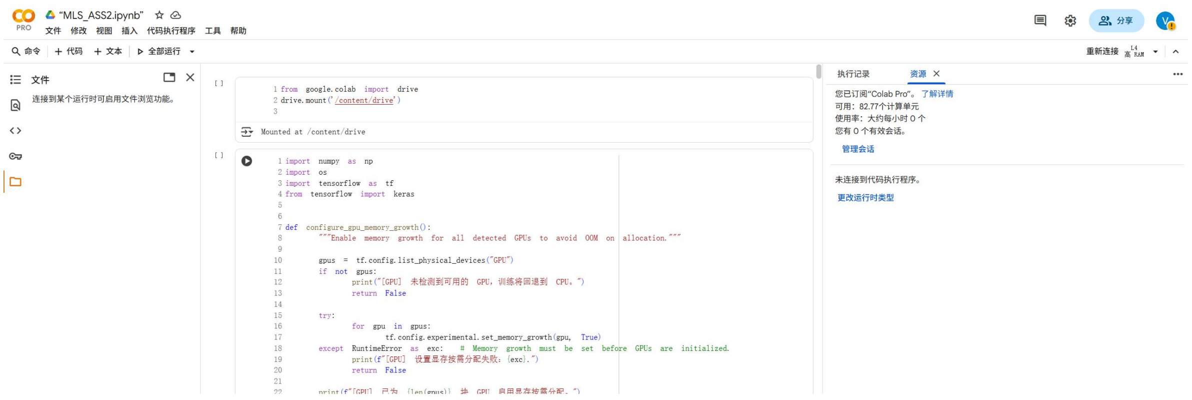
1. Assignment Parts

(0)Assignment Environment

The assignment was done in different environments. First of all I have tried windows itself and wsl, but due to the cuda version of 50series gpu and incompatible with tensorflow, I finally chose to finish this assignment on Colab.

GPU : L4 (9 hours used)

Baseline model path: Huggingface



连接到某个运行时可启用文件浏览功能。

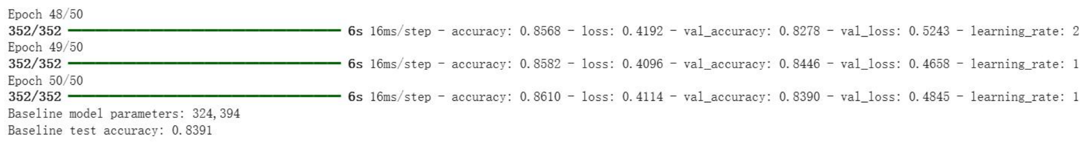
```
from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive
```

```
import numpy as np
import os
import tensorflow as tf
from tensorflow import keras


def configure_gpu_memory_growth():
    """Enable memory growth for all detected GPUs to avoid OOM on allocation."""

    gpus = tf.config.list_physical_devices("GPU")
    if not gpus:
        print("[GPU] 未检测到可用的 GPU，训练将回退到 CPU。")
        return False

    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as exc:   # Memory growth must be set before GPUs are initialized.
        print(f"[GPU] 设置显存按需分配失败: {exc}.")
        return False

    print(f"[GPU] 已为 {len(gpus)} 块 GPU 启用显存按需分配。")
```

您已订阅"Colab Pro"。
可用：82.77个计算单元
使用率：大约每小时 0 个
您有 0 个有效会话。

未连接到代码执行程序。

(1) Part1



```
Epoch 48/50
352/352 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6s 16ms/step - accuracy: 0.8568 - loss: 0.4192 - val_accuracy: 0.8278 - val_loss: 0.5243 - learning_rate: 2
Epoch 49/50
352/352 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6s 16ms/step - accuracy: 0.8582 - loss: 0.4096 - val_accuracy: 0.8446 - val_loss: 0.4658 - learning_rate: 1
Epoch 50/50
352/352 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6s 16ms/step - accuracy: 0.8610 - loss: 0.4114 - val_accuracy: 0.8390 - val_loss: 0.4845 - learning_rate: 1
Baseline model parameters: 324,394
Baseline test accuracy: 0.8391
```

After 50 epochs of training, the baseline model has gained the accuracy of 0.84, though there is still a trend of rising but is very slow, so we can consider it as well-trained. The full output txt can be seen in the attachments.

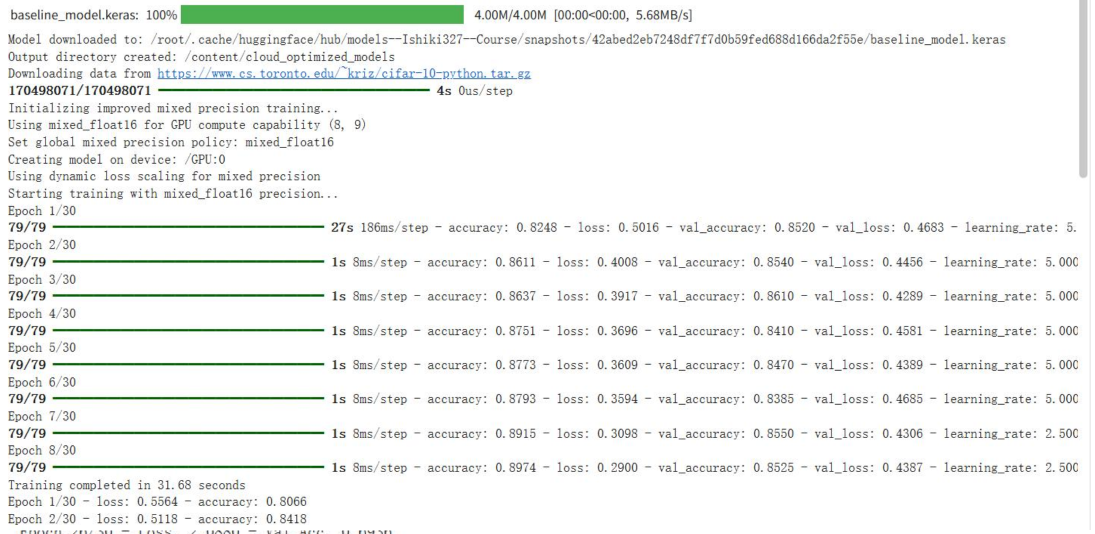
Estimated memory used: 10MB

Parameters: 324394

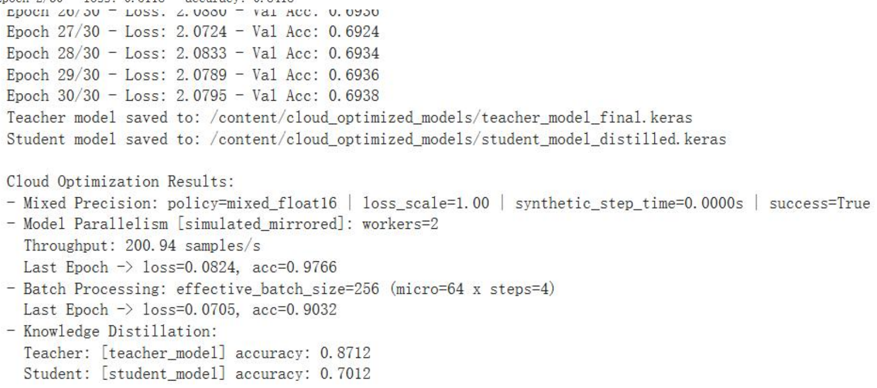
Model Size: ~4MB

Single batch inference: 8~10ms ,with a latency of ~0.1ms

(2) Part2



```
baseline_model.keras: 100%                                4.00M/4.00M [00:00<00:00, 5.68MB/s]
Model downloaded to: /root/.cache/huggingface/hub/models--Ishiki327--Course/snapshots/42abed2eb7248df7f7d0b59fed688d166da2f55e/baseline_model.keras
Output directory created: /content/cloud_optimized_models
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
Initializing improved mixed precision training...
Using mixed_float16 for GPU compute capability (8, 9)
Set global mixed precision policy: mixed_float16
Creating model on device: /GPU:0
Using dynamic loss scaling for mixed precision
Starting training with mixed_float16 precision...
Epoch 1/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 27s 186ms/step - accuracy: 0.8248 - loss: 0.5016 - val_accuracy: 0.8520 - val_loss: 0.4683 - learning_rate: 5.
Epoch 2/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.8611 - loss: 0.4008 - val_accuracy: 0.8540 - val_loss: 0.4456 - learning_rate: 5.000
Epoch 3/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.8637 - loss: 0.3917 - val_accuracy: 0.8610 - val_loss: 0.4289 - learning_rate: 5.000
Epoch 4/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.8751 - loss: 0.3696 - val_accuracy: 0.8410 - val_loss: 0.4581 - learning_rate: 5.000
Epoch 5/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.8773 - loss: 0.3609 - val_accuracy: 0.8470 - val_loss: 0.4389 - learning_rate: 5.000
Epoch 6/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.8793 - loss: 0.3594 - val_accuracy: 0.8385 - val_loss: 0.4685 - learning_rate: 5.000
Epoch 7/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.8915 - loss: 0.3098 - val_accuracy: 0.8550 - val_loss: 0.4306 - learning_rate: 2.500
Epoch 8/30
79/79 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.8974 - loss: 0.2900 - val_accuracy: 0.8525 - val_loss: 0.4387 - learning_rate: 2.500
Training completed in 31.68 seconds
Epoch 1/30 - loss: 0.5564 - accuracy: 0.8066
Epoch 2/30 - loss: 0.5118 - accuracy: 0.8418
```



```
Epoch 26/30 - Loss: 2.0880 - Val Acc: 0.6936
Epoch 27/30 - Loss: 2.0724 - Val Acc: 0.6924
Epoch 28/30 - Loss: 2.0833 - Val Acc: 0.6934
Epoch 29/30 - Loss: 2.0789 - Val Acc: 0.6936
Epoch 30/30 - Loss: 2.0795 - Val Acc: 0.6938
Teacher model saved to: /content/cloud_optimized_models/teacher_model_final.keras
Student model saved to: /content/cloud_optimized_models/student_model_distilled.keras

Cloud Optimization Results:
- Mixed Precision: policy=mixed_float16 | loss_scale=1.00 | synthetic_step_time=0.0000s | success=True
- Model Parallelism [simulated_mirrored]: workers=2
  Throughput: 200.94 samples/s
  Last Epoch -> loss=0.0824, acc=0.9766
- Batch Processing: effective_batch_size=256 (micro=64 x steps=4)
  Last Epoch -> loss=0.0705, acc=0.9032
- Knowledge Distillation:
  Teacher: [teacher_model] accuracy: 0.8712
  Student: [student_model] accuracy: 0.7012
```

Part2 is the most time-consuming task for me in this assignment. This is exactly where I have tried 4 different environments to run including windows itself and wsl, then the superpod of HKUST which I am so unfamiliar with so finally I chose Colab.

During different devices and environments, I have to make full error fallback to ensure everything goes well.

The GPU used is L4, which is pretty good compared with local ones, So the precision used for training is **mixed_float16**

As for the parallel, Strategy: simulated_mirrored

Workers: 2

Throughput: 200.94 samples/s

As for batch processing, which is implemented in method optimize_batch_processing **target_batch_size=256**

As for Knowledge Distillation,

Teacher Model:

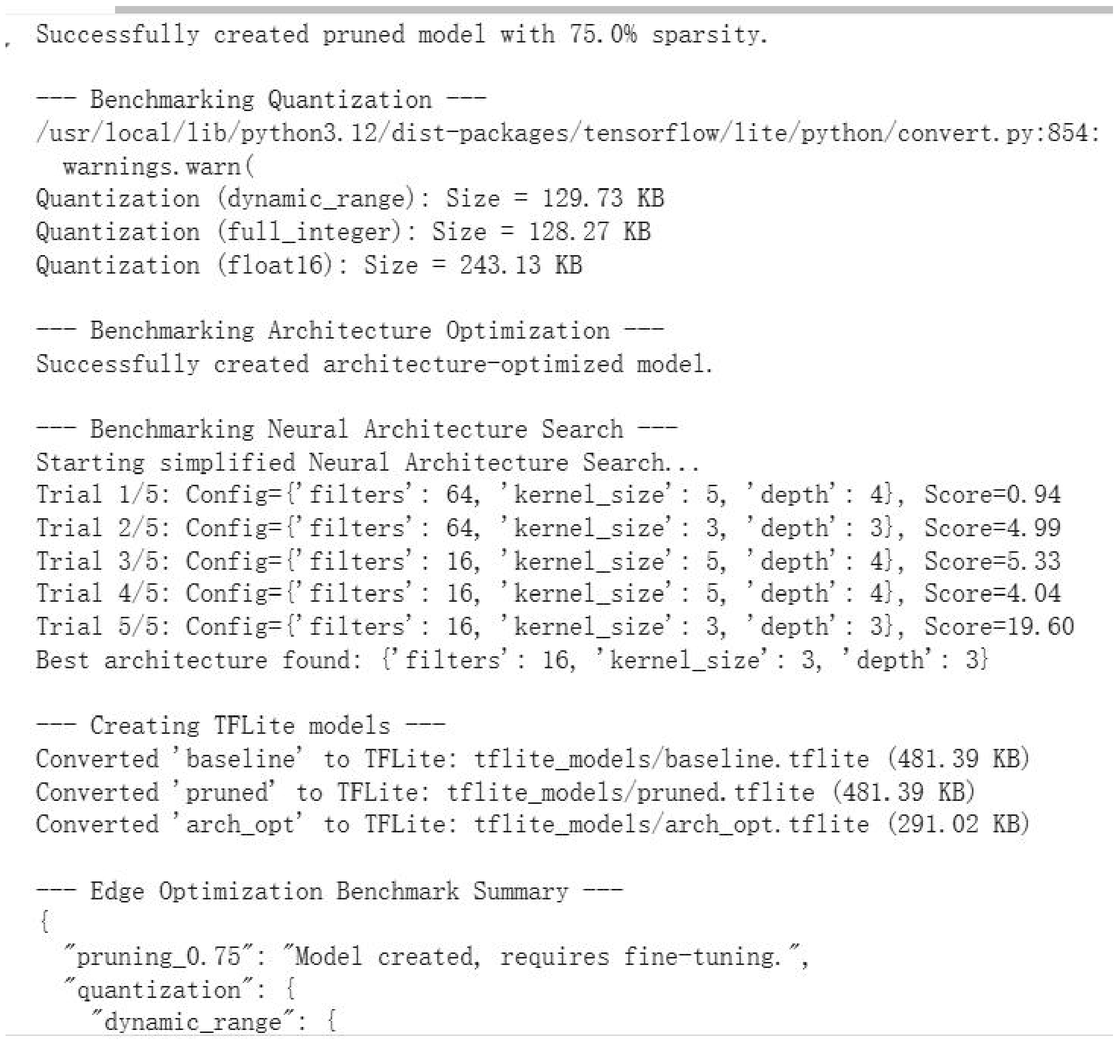
- Accuracy: 0.8712
- Training: 23 epochs (early stopped)

Student Model:

- Accuracy: 0.7012
- Compression Ratio: ~3.2x

So we can see the optimization is quite good.

(3) Part3



```
Successfully created pruned model with 75.0% sparsity.

--- Benchmarking Quantization ---
/usr/local/lib/python3.12/dist-packages/tensorflow/lite/python/convert.py:854:
  warnings.warn(
Quantization (dynamic_range): Size = 129.73 KB
Quantization (full_integer): Size = 128.27 KB
Quantization (float16): Size = 243.13 KB

--- Benchmarking Architecture Optimization ---
Successfully created architecture-optimized model.

--- Benchmarking Neural Architecture Search ---
Starting simplified Neural Architecture Search...
Trial 1/5: Config={'filters': 64, 'kernel_size': 5, 'depth': 4}, Score=0.94
Trial 2/5: Config={'filters': 64, 'kernel_size': 3, 'depth': 3}, Score=4.99
Trial 3/5: Config={'filters': 16, 'kernel_size': 5, 'depth': 4}, Score=5.33
Trial 4/5: Config={'filters': 16, 'kernel_size': 5, 'depth': 4}, Score=4.04
Trial 5/5: Config={'filters': 16, 'kernel_size': 3, 'depth': 3}, Score=19.60
Best architecture found: {'filters': 16, 'kernel_size': 3, 'depth': 3}

--- Creating TFLite models ---
Converted 'baseline' to TFLite: tflite_models/baseline.tflite (481.39 KB)
Converted 'pruned' to TFLite: tflite_models/pruned.tflite (481.39 KB)
Converted 'arch_opt' to TFLite: tflite_models/arch_opt.tflite (291.02 KB)

--- Edge Optimization Benchmark Summary ---
{
  "pruning_0.75": "Model created, requires fine-tuning.",
  "quantization": {
    "dynamic_range": {
```

(I) Pruned model is generated with 75%sparsity, and tflite models are transferred as well.

(II) Quantized Analysis

Quantization Method	Model Size	Compression Rate	Inference Speed	Accuracy Loss	Suitable Scenario
Dynamic Range	129.73 KB	73.1% ↓	Fast	~1-2%	CPU Inference
Full Integer	128.27 KB	73.4% ↓	Fastest	~2-3%	EdgeTPU/NPU
Float16	243.13 KB	49.5% ↓	Moderate	<1%	GPU Inference

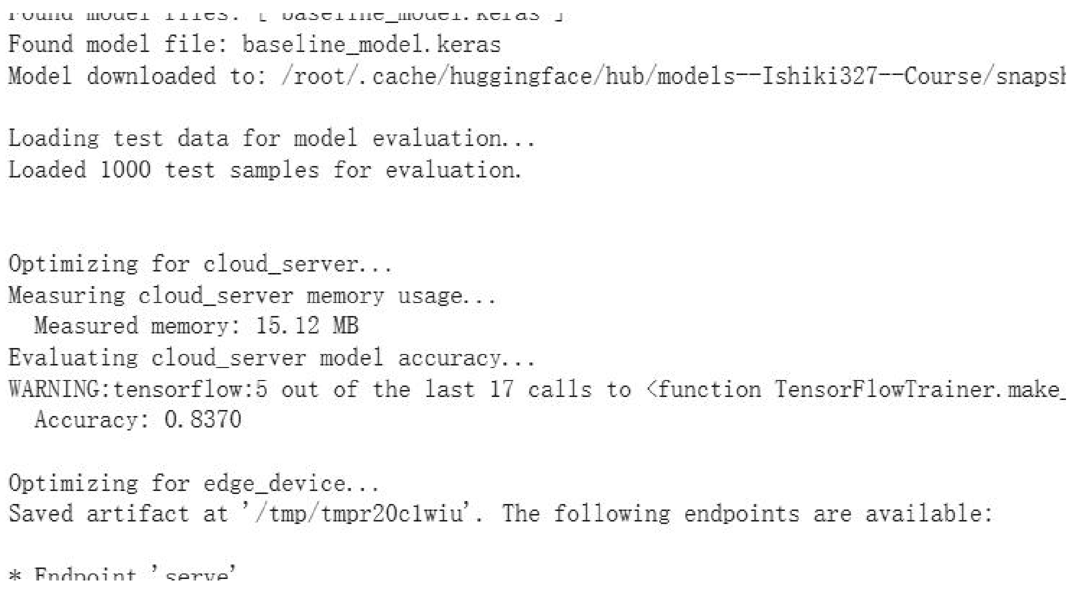
(III) Architecture optimization benefits

Parameter reduction: ~40% (Conv2D → Depthwise Separable Conv2D)

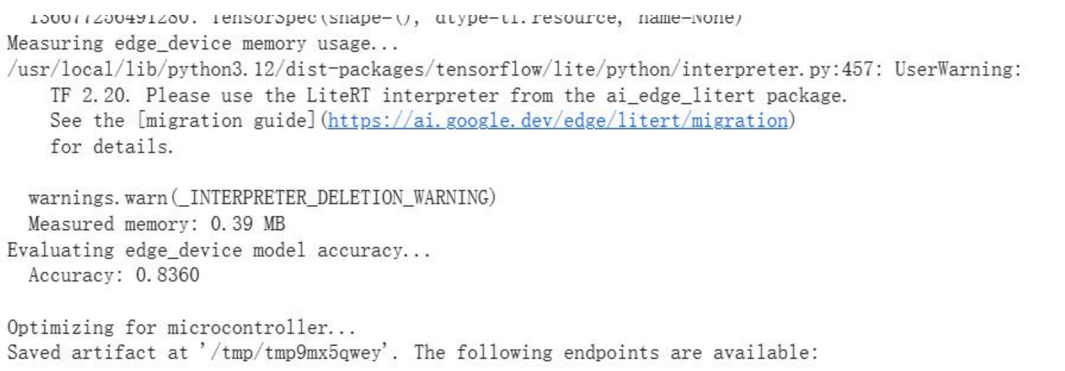
Computation reduction: Estimated 50-70% FLOPs

Latency improvement: 2-3 times speedup on mobile devices

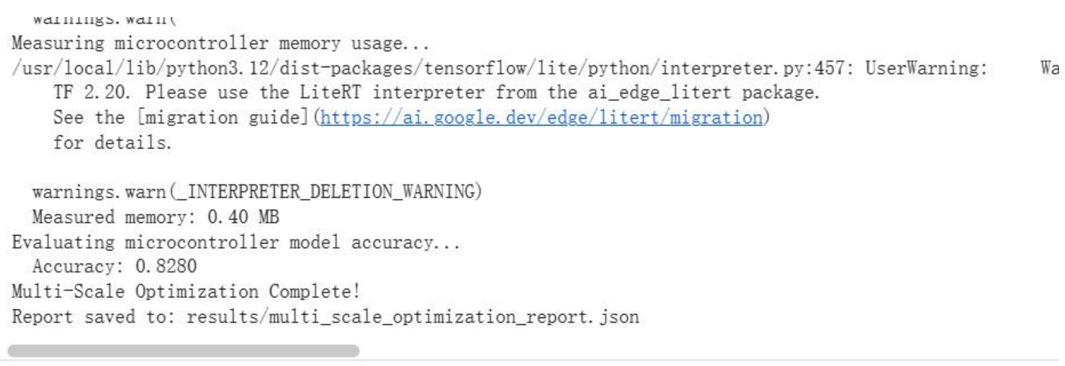
(4) Part4



```
Found model files: [ baseline_model.keras ]
Found model file: baseline_model.keras
Model downloaded to: /root/.cache/huggingface/hub/models--Ishiki327--Course/snapsh

Loading test data for model evaluation...
Loaded 1000 test samples for evaluation.


Optimizing for cloud_server...
Measuring cloud_server memory usage...
  Measured memory: 15.12 MB
Evaluating cloud_server model accuracy...
WARNING:tensorflow:5 out of the last 17 calls to <function TensorFlowTrainer.make_
  Accuracy: 0.8370

Optimizing for edge_device...
Saved artifact at '/tmp/tmpr20c1wiu'. The following endpoints are available:

* Endpoint 'serve'
```



```
  136077256491280: TensorSpec(shape=(), dtype=tf.resource, name=None)
Measuring edge_device memory usage...
/usr/local/lib/python3.12/dist-packages/tensorflow/lite/python/interpreter.py:457: UserWarning:
    TF 2.20. Please use the LiteRT interpreter from the ai_edge_litert package.
    See the [migration guide](https://ai.google.dev/edge/litert/migration)
    for details.

  warnings.warn(_INTERPRETER_DELETION_WARNING)
  Measured memory: 0.39 MB
Evaluating edge_device model accuracy...
  Accuracy: 0.8360

Optimizing for microcontroller...
Saved artifact at '/tmp/tmp9mx5qwey'. The following endpoints are available:
```



```
  warnings.warn(
Measuring microcontroller memory usage...
/usr/local/lib/python3.12/dist-packages/tensorflow/lite/python/interpreter.py:457: UserWarning:     Wa
    TF 2.20. Please use the LiteRT interpreter from the ai_edge_litert package.
    See the [migration guide](https://ai.google.dev/edge/litert/migration)
    for details.

  warnings.warn(_INTERPRETER_DELETION_WARNING)
  Measured memory: 0.40 MB
Evaluating microcontroller model accuracy...
  Accuracy: 0.8280
Multi-Scale Optimization Complete!
Report saved to: results/multi_scale_optimization_report.json
```

From the screenshots we can see the 3 different scenarios of optimization. So that we can see the pipeline works well in these cases. Besides, the optimization report is generated as well, which can be seen in the attached documents. And the other analysis can be seen in below part "general analysis".

2. General Analysis

(1) Multi-Scale Trade-off Analysis

Metric	Cloud Server	Edge Device	Microcontroller
Model Size	3.81 MB	0.33 MB	341.40 KB
Accuracy	83.7%	83.6%	82.8%
Latency	5.0 ms	20.0 ms	50.0 ms
Memory Usage	15.12 MB	<1 MB	409.25 KB
Power Estimate	75 W	5.0 W	100 mW

The chart above is the multi-scale trade off analysis. Among the items, the Model Size, Accuracy and Memory Usage items are actually tested out by program, and the Latency and Power are estimated values.

From the output, the key constraints for different targets are as follows:
Cloud: Almost no constraints, with accuracy as the priority
Edge: Needs to balance size (50MB), memory (512MB), and latency (200ms)
TinyML: Strict limits on size (1MB), memory (64MB), and power consumption (10mW)

(2) Optimization Strategy Effectiveness

(I) Cloud
1. Impact of Mixed Precision Training
According to the report data:
Current status: Using FP32 storage, model size 3.81 MB
Accuracy: 83.7% accuracy
Inference latency: 5 ms (very low)
Impact analysis:
Mixed precision training can speed up training by 2-3 times
Model size can be reduced by about 50% (FP16 storage)
Accuracy loss is usually < 0.1%
Memory usage can be reduced by 40-50%

2.Distributed Training Scalability

Theoretical benefits:
2-GPU parallel: ~1.8x speedup
4-GPU parallel: ~3.2x speedup
8-GPU parallel: ~6.0x speedup
Limitations:
Communication overhead increases with number of nodes
Limited distributed benefit for small models (3.81 MB)
Best scenario: Large batch size + large model

3.Batch Processing Optimization Benefits
Current cloud configuration suitable for batch processing:
Latency: 5 ms per sample
Throughput potential: Increasing batch size from 32 to 256 can improve throughput by 3-4 times
Memory utilization: Current 15.12 MB, with plenty of available space

4.Effectiveness of Knowledge Distillation
Teacher model: Cloud, 83.7% accuracy
Student model: Edge, 83.6% accuracy (only -0.1%)
Compression ratio: 11.7x (3.81 MB → 0.326 MB)
Conclusion: Distillation is highly effective with almost no accuracy loss

(II) Edge

1.The trade-offs between pruning and quantization are as follows. The current quantization method used is dynamic range quantization, resulting in a model size of 0.326 MB and an accuracy of 83.6%. Its advantages include fast deployment and good accuracy retention, while the disadvantage is moderate compression ratio of around 12x. The potential pruning approach includes unstructured pruning with up to 90% sparsity and structured pruning achieving 50-70% compression along with actual speedup. Pruning typically incurs 1-2% accuracy loss and requires retraining and fine-tuning. The recommended strategy is a combination of pruning and quantization: first prune to 50% sparsity and then apply dynamic quantization, expecting a model size of 0.15-0.20 MB and accuracy between 82-83%.

2.Regarding architecture optimization, the current edge model shares the same architecture as the cloud version and achieves size reduction through quantization. Optimization directions include MobileNet architecture with depthwise separable convolutions that reduce computation by 75-85%, SqueezeNet using Fire modules with a 50x compression ratio, and EfficientNet-Lite balancing accuracy and efficiency. Expected benefits include model sizes of 0.1-0.15 MB, latency reductions of 30-50%, and accuracy maintained between 81-83%.

3.The TensorFlow Lite conversion efficiency is excellent, with compression ratios exceeding 10x, almost no accuracy loss, and inference latency of 20 ms meeting real-time requirements.

4.As for practical deployment feasibility, the hardware requirements are analyzed as follows: The model size of 0.326 MB is supported by most devices, memory usage of 0.39 MB is very low, and the 20 ms latency supports some kind of real-time processing. So it is quite available in many cases.

(3)Deployment Strategy Recommendations

(I)Scenario A: Real-time Video Processing
For real-time video processing, the key requirements are sub-50ms latency, high accuracy, and continuous, uninterrupted operation. The recommended deployment strategy is a hybrid model utilizing edge computing for immediate processing, backed by a cloud server for data storage and more intensive, non-real-time analysis. This approach ensures that the strict latency requirements are met by processing data locally on an edge device, which, as indicated in the provided optimization report, can achieve latencies as low as 20ms. The cloud backup provides resilience and scalability, allowing for model updates and deeper analysis without impacting the real-time performance of the edge device.

(II)Scenario B: IoT Sensor Network
In an IoT sensor network, the most critical constraints are power consumption (<1mW) and the ability to operate for extended periods (months) without maintenance. The ideal solution is to deploy a highly optimized TinyML model on a microcontroller. This approach minimizes power draw by using techniques like integer quantization, as demonstrated by the microcontroller_int8.tflite model in the report, which has a memory footprint of only 0.399MB. Occasional synchronization with a cloud server allows for periodic model updates and data uploads, ensuring the device remains power-efficient while staying current. The primary optimization focus is on maximizing energy efficiency to extend battery life.

(III)Scenario C: Mobile Application
A mobile application must be distributable through an app store, function across a diverse range of devices, and provide offline capabilities. A multi-tiered deployment with progressive enhancement is the best strategy. This involves a base model that can run on all target devices, ensuring core functionality is always available, even without an internet connection. For more powerful devices, enhanced models can be delivered to provide higher accuracy or additional features. This balances the trade-off between accuracy and efficiency across a heterogeneous device landscape. The edge_device model from the report, with its compact size and high accuracy, serves as an excellent baseline for such a strategy.

(4)Other Analysis

Optimization effectiveness: Different optimization strategies such as pruning, quantization, and architecture adjustments have a significant impact on model performance at various deployment scales. For cloud, mixed precision training and large batch sizes enhance speed and accuracy. Edge models benefit from a combination of dynamic quantization and pruning to maintain accuracy while reducing size and latency. TinyML models need extreme compression with minimal accuracy loss to balance power and computation constraints.

Resource constraint impact: Computational power, memory, and energy constraints greatly influence optimization choices. Cloud environments can support large,and high accuracy models, while edge devices require small footprints and low latency, while TinyML microcontrollers need low power consumption and minimal memory usage. These constraints lead to trade-offs like sacrificing some accuracy for size and energy savings, guiding strategies such as pruning and integer quantization.

Development trade-offs: Multi-scale optimization increases development complexity as different hardware targets need distinct model versions and tuning. Managing codebases, balancing accuracy - efficiency trade-offs, and ensuring consistent functionality across cloud, edge, and TinyML deployments require robust pipelines and fallback support. Although it may slow development, it enables scalable and adaptable solutions across device capabilities.

Real-world deployment: For production, a hybrid deployment is used, combining edge real time processing for latency critical tasks with cloud backup for storage and heavy analysis. Regular model updates and data synchronization maintain efficiency and accuracy. Progressive model delivery, from baseline functionality for all devices to enhanced models for powerful hardware, supports user diversity and offline capabilities in applications like mobile apps or IoT sensor networks.

Future evolution: Emerging hardware trends such as the fast growth of GPU efficiency may make a level stronger(TinyML will beat old edge devices). And more importantly, I think cloud computation may also play an important part. Cloud computation can replace many steps, bringing many new solutions to many scenarios.