模型压缩与蒸馏：剪枝、量化与 TinyLLM 边缘部署

2025 年 10 月 25 日

1 剪枝（Pruning）、量化（Quantization）

1.1 压缩路线图

图?? 展示了从全精度 LLM 到轻量化部署模型的典型路径：先通过剪枝降低参数冗余，再通过量化减少数值精度，最后可结合蒸馏形成新的学生模型。

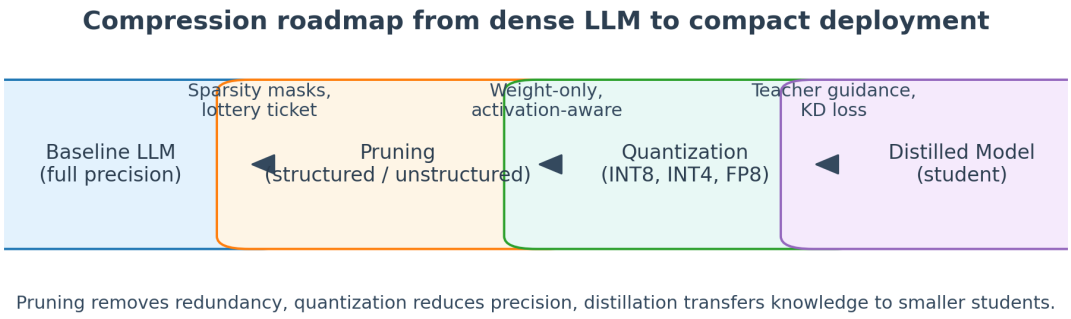

图 1: 压缩路线图：剪枝、量化与蒸馏协同减少模型体积与计算量。

1.2 剪枝策略

- **非结构化剪枝**：以权重幅值或梯度为指标，将小权重置零；灵活但硬件友好度较低。
- **结构化剪枝**：去除整列、整头或通道，便于 GPU/CPU 加速；常见于注意力头剪枝、FFN 通道剪枝。
- **动态稀疏**：训练过程中动态更新稀疏模式（RigL、SNIP），在保持精度的同时提升收敛速度。

- **彩票假说**：寻找可独立训练的稀疏子网络，为后续微调提供轻量结构。

1.3 量化方法

类别	位宽	特点	典型工具
动态量化	INT8	推理阶段按批次重新统计量化参数	PyTorch Dynamic Quantization
静态量化	INT8	预先收集校准数据计算 scale/zero-point	TensorRT, ONNX Runtime
权重量化	INT4/INT3	仅量化权重，激活保持 FP16	GPTQ, AWQ
激活感知量化	INT8/INT4	同时考虑激活分布，减少误差	SmoothQuant, AQLM
混合精度	FP8/INT8	利用最新硬件（H100、Gaudi2）支持的低精度格式	TransformerEngine

1.4 量化示例：GPTQ

Listing 1: 使用 AutoGPTQ 对 LLaMA 权重量化

```
from auto_gptq import AutoGPTQForCausalLM, BaseQuantizeConfig
from transformers import AutoTokenizer

model_name = "meta-llama/Llama-2-7b-hf"
quant_config = BaseQuantizeConfig(
    bits=4,
    group_size=128,
    desc_act=False  # 关闭激活量化
)

model = AutoGPTQForCausalLM.from_pretrained(model_name, quantize_config=quant_config)
tokenizer = AutoTokenizer.from_pretrained(model_name, use_fast=True)

model.quantize(
    examples=["Hello world!", "Quantization reduces memory footprint."],
    batch_size=8,
    use_triton=True,
```

```
)

model.save_quantized("llama2-7b-gptq")
tokenizer.save_pretrained("llama2-7b-gptq")
```

2 知识蒸馏（Knowledge Distillation）

2.1 蒸馏流程

知识蒸馏通过教师模型（Teacher）向学生模型（Student）传递知识，常见目标包括：

- **软标签（Soft Targets）**：使用温度系数 T 放大概率分布，学生最小化 KL 散度。
- **中间层对齐**：蒸馏注意力、隐藏状态、梯度统计等中间特征，增强学生表达力。
- **任务蒸馏**：将教师模型在真实任务上的输出作为监督信号（SFT+KD）。

2.2 损失函数设计

综合蒸馏损失可写为：

$$\mathcal{L} = \alpha \cdot \mathcal{L}_{\text{KD}}(p_s, p_t) + \beta \cdot \mathcal{L}_{\text{task}}(y_s, y_{\text{true}}) + \gamma \cdot \mathcal{L}_{\text{feature}}(h_s, h_t), \tag{1}$$

其中 p_s、p_t 分别为学生与教师的 softmax 输出，h 代表中间特征，α, β, γ 控制权重。

2.3 蒸馏案例

- **TinyLlama**：基于 3 亿参数的小模型，通过蒸馏大模型指令数据获得接近 7B 模型的能力。
- **MiniLM**：利用深度自注意力蒸馏，让 384 维模型达到 BERT-base 的效果。
- **LLaDA**：在多语言任务中将多模态知识蒸馏到紧凑模型。

2.4 蒸馏伪代码

Listing 2: Teacher-Student 蒸馏训练循环简例

```
for batch in dataloader:
    with torch.no_grad():
        teacher_logits, teacher_hidden = teacher(**batch,
           output_hidden_states=True)
```

```
    student_logits, student_hidden = student(**batch,
        output_hidden_states=True)

    loss_kd = kl_divergence(
        F.log_softmax(student_logits / T, dim=-1),
        F.softmax(teacher_logits / T, dim=-1)
    ) * (T * T)

    loss_task = cross_entropy(student_logits, batch["labels"])
    loss_hidden = mse_loss(student_hidden[-1], projector(teacher_hidden
        [-1]))

    loss = alpha * loss_kd + beta * loss_task + gamma * loss_hidden
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

3 TinyLLM 与边缘设备部署

3.1 部署栈概览

图?? 描述了 TinyLLM 在边缘侧部署的核心组件：上游压缩流水线产生量化模型，TinyLLM Runtime 利用特定硬件内核执行，遥测与编排层实现反馈闭环。

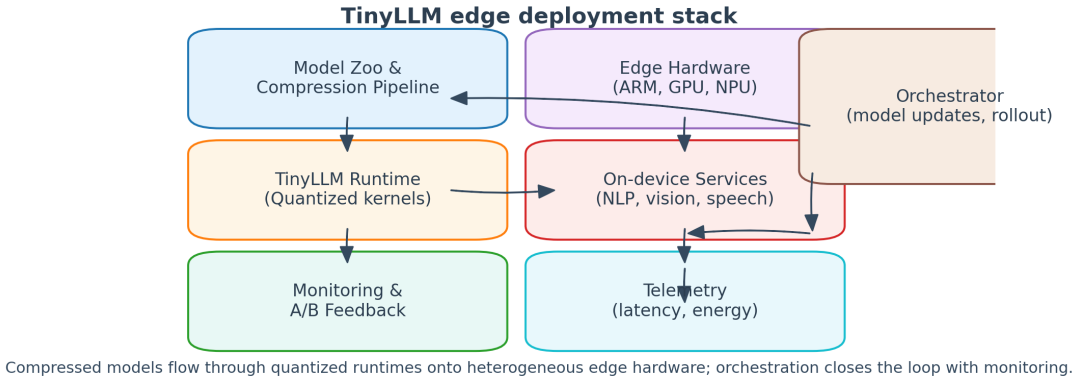

图 2: TinyLLM 边缘部署栈：压缩流水线、量化运行时、异构硬件与编排治理。

3.2 TinyLLM Runtime

- **内核优化**：针对 INT4/INT8 张量核、张量切片、流水线并行进行优化；

- **内存管理**：采用静态内存池、KV Cache 压缩（grouped-query attention）；
- **调度策略**：支持多租户、批处理、自适应延迟控制。

3.3 边缘硬件适配

硬件	精度支持	典型场景	工具栈
ARM CPU (Neon)	INT8/INT4	移动端对话、离线助手	MNN, NCNN, llama.cpp
NVIDIA Jetson	FP16/INT8	机器人、工业检测	TensorRT, FasterTransformer
Apple Neural Engine	8-bit	iOS 应用内推理	Core ML, Metal Performance Shaders
定制 NPU	INT4/INT2	智能家居、车载系统	ONNX Runtime EP, TVM

3.4 部署流水线

1. **模型准备**：对基础模型执行剪枝 + 量化 + 蒸馏形成学生模型；
2. **格式转换**：导出为 ONNX、TensorRT engine、Core ML、GGUF 等；
3. **运行时集成**：嵌入 TinyLLM Runtime 或各类推理框架（MNN、llama.cpp）；
4. **监控闭环**：收集延迟、能耗、质量指标，回传云端重新蒸馏或更新模型。

3.5 部署脚本示例

Listing 3: 使用 llama.cpp 推理量化模型

```
import subprocess
import pathlib

MODEL_PATH = pathlib.Path("models/tinyllm-q4_0.gguf")
PROMPT = "Summarize the daily production report in Chinese."

cmd = [
    "./main",
    "-m", str(MODEL_PATH),
    "-p", PROMPT,
    "-n", "128",
```

```
    "--temp", "0.8",
    "--batch-size", "32",
    "--threads", "8"
]

result = subprocess.run(cmd, capture_output=True, text=True, check=True)
print(result.stdout)
```

实践建议

- 将剪枝、量化、蒸馏视为组合拳，先评估精度敏感度再选择策略；
- 对压缩模型进行系统化评测：困惑度、下游任务、延迟、能耗、安全性；
- 建立持续学习与反馈回路，利用边缘日志改进蒸馏数据与模型版本；
- 注重安全与隐私：在边缘端启用本地推理可减少数据外泄风险，但仍需加密存储和访问控制。

参考文献

- Frantar et al. “GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers.” NeurIPS, 2022.
- Dettmers et al. “QLoRA: Efficient Finetuning of Quantized LLMs.” NeurIPS, 2023.
- Sanh et al. “DistilBERT, a distilled version of BERT.” NeurIPS, 2019.
- Zhang et al. “TinyLlama: Tiny Language Models for Edge AI.” arXiv, 2024.
- Li et al. “AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration.” arXiv, 2023.