# 《人工智能原理与实践》期末大作业说明书

**项目名称**：NanoGPT-Edge：构建端侧可部署的微型大语言模型 **适用对象**：计算机科学/人工智能专业本科生 **算力要求**：个人笔记本电脑 (CPU / NVIDIA GPU 4GB+)

## 1. 项目概述 (Project Overview)

### 1.1 项目背景

当前，以 GPT-4 为代表的大语言模型（LLM）展现了惊人的能力，但其庞大的参数量极其依赖昂贵的集群算力，难以在边缘设备（如手机、IoT 设备、个人 PC）上普及。**端侧 AI (On-Device AI)** 正成为行业的新热点。

本项目旨在打破“大模型只能在服务器上跑”的思维定势。你将亲手从 0 到 1 构建一个参数量小于 50M 的微型 LLM，并攻克**架构优化（MoE）**、**指令微调（SFT）**、**模型压缩（Quantization）** 和 **知识增强（RAG）** 四大核心技术难点。

### 1.2 项目目标

基于 PyTorch 框架，构建一个具备指令跟随能力的 Decoder-only Transformer 模型。该模型需在保持轻量级（<50M 参数）的同时，通过混合专家（MoE）架构提升容量，通过量化适配端侧推理，并通过 RAG 外挂知识库解决“幻觉”问题。

## 2. 核心技术指标 (Key Performance Indicators)

学生需在代码中实现以下 5 个关键技术模块（必做）：

1. **原生 Transformer 实现**：
   * 禁止直接调用 HuggingFace Transformers 库中的现成模型（如 AutoModel）。
   * 需手动基于 torch.nn 编写 Decoder-only 架构。
2. **混合专家模型 (MoE)**：
   * 引入稀疏门控机制（Sparse Gating），在不显著增加推理 FLOPs 的前提下增加模型总参数量。
3. **指令微调 (SFT)**：
   * 实现从 Pre-training（续写）到 Instruction Tuning（对话）的范式转变。
4. **训练后动态量化 (PTQ)**：
   * 将模型权重从 FP32 压缩至 INT8，实现体积缩小 4 倍以上。
5. **检索增强生成 (RAG)**：
   * 实现向量检索系统，赋予小模型查阅外部知识库的能力。

## 3. 具体实施步骤 (Implementation Steps)

### 阶段一：模型架构搭建 (Architecture Design)

**目标**：构建符合 LLaMA/PaLM 等现代架构标准的“基座模型”。

* **基础组件要求**：
  + **RMSNorm**：替代 LayerNorm，提升训练稳定性。
  + **RoPE (Rotary Embedding)**：实现旋转位置编码，提升外推能力。
  + **SwiGLU**：替代 ReLU/GELU，提升 FFN 表达能力。
* **MoE 模块实现**：
  + **结构替换**：将 Transformer Block 中的标准 FFN 层替换为 MoE 层。
  + **Router (门控网络)**：实现 Top-2 路由机制（torch.topk），从 N 个专家中激活权重最高的 2 个。
  + **配置建议**：专家数量 4-8 个，总参数量 \le 50M。
* **验收标准**：模型能完成一次完整的前向传播（Forward Pass），参数统计正确（推荐使用 torchinfo 或 torchsummary 验证）。

### 阶段二：全量预训练 (Pre-training)

**目标**：让模型掌握基础的词法、句法和逻辑生成能力。

* **数据集**：[roneneldan/TinyStories](https://huggingface.co/datasets/roneneldan/TinyStories)
  + *说明*：该数据集由 GPT-4 生成，词汇简单、逻辑清晰，极易在小参数模型上收敛。
* **核心任务**：
  + **Tokenizer**：推荐使用 tiktoken (GPT-2/4 preset) 或使用 tokenizers 库训练 BPE (Vocab size: 4096-8192)。
  + **训练循环**：实现 Next Token Prediction 任务（CrossEntropyLoss）。
  + **超参建议**：Optimizer: AdamW; LR: 1e-3 ~ 1e-4; Schedule: Cosine Decay。
* **验收标准**：
  + Training Loss 呈现明显下降趋势（目标 Loss < 2.5）。
  + 给定开头 "Once upon a time"，模型能续写出一段语法正确、逻辑通顺的英文短故事。

### 阶段三：指令微调 (Supervised Fine-Tuning, SFT)

**目标**：将模型从“故事续写者”转变为“对话助手”。

* **数据集构建**：
  + 构造 500-1000 条 JSON 格式的指令数据（参考 Alpaca 格式）。
  + *数据样例*：  
    {"instruction": "翻译成英文", "input": "你好", "output": "Hello"}
* **微调策略**：
  + 可以通过修改nanoGPT中的train.py代码来实现微调。（建议）
  + 也可以将经过全量预训练的模型转换成llamafactory可识别的格式，利用llamafactory进行微调。
* **验收标准**：模型能准确响应指令（如“Who are you?”），而不是对问题进行续写或重复问题，微调之后展示结果。

### 阶段四：模型量化 (Model Quantization)

**目标**：极致压缩模型体积，模拟移动端部署环境。

* **量化方案**：
  + 使用 torch.quantization.quantize\_dynamic 实现 **W8A16** 策略（权重 INT8，激活 FP32/FP16）。
  + *注意*：仅针对 nn.Linear 层进行量化。
* **对比分析**：
  + **体积**：计算文件大小压缩比（预期接近 4:1）。
  + **精度**：在测试集上计算量化前后的 **Perplexity (PPL)** 变化，增长率需尽量小。
* **验收标准**：量化模型加载成功，推理无乱码，且体积达标。

### 阶段五：检索增强生成 (RAG)

**目标**：外挂知识库，解决小模型“知识储备不足”和“幻觉”问题。

* **知识库以及环境准备**
  + 整理你的微调模型
  + 准备知识库文档
  + 基础的框架使用 Lanchain，向量模型使用 sentence-transfomers，向量数据库使用 chromadb，基础文本的处理库有 pypdf2（pdf 文档），python-docx（处理 word），pands（excel）。使用 transformers 库来进行推理并结合以上的库来完成整个 RAG 的流程（以下任务，达成验收标准）
* **任务**
  + 用户提出问题
  + 将用户问题向量化
  + 在知识库中检索相关的文档
  + 构建提示词并拼接相关文档成指令
  + 让你的模型根据拼装好的指令生成答案
  + 返回结果
* **验收标准：**模型能基于知识库回答问题，你可以在自己创建的知识库中存储自己的相关信息，向模型提问有关自己的问题，如果模型能回答出来，说明知识库应用成功。

## 4. 交付物清单 (Deliverables)

请将以下内容打包为 学号\_姓名\_NanoGPT.zip 提交：

### 1. 源代码 (/src)

* model.py: Transformer Block, MoE Layer, RoPE 等核心组件定义。
* train.py: 包含 Pre-train 和 SFT 的完整训练流水线。
* inference.py: 最终的交互脚本，需集成 RAG 功能和模型加载逻辑。
* quantize.py: 执行模型量化并保存权重的脚本。
* requirements.txt: 项目依赖库列表。

### 2. 模型权重 (/weights)

* best\_sft\_model.pth: SFT 后的 FP32 模型权重。
* quantized\_model.pt: 量化后的 INT8 模型权重。
  + *注：若文件过大，请在文档中提供网盘下载链接。*

### 3. 实验报告 (Report.pdf) **[核心评分依据]**

需包含以下图表与分析：

* **架构图解**：绘制 MoE 路由逻辑图及整体架构图。
* **训练日志**：Loss 曲线图（Pre-train & SFT）。
* **MoE 分析**：**专家负载可视化**（统计推理过程中各专家的被选中频率，分析是否存在“专家坍缩”现象）。
* **量化报告**：FP32 vs INT8 的大小、PPL、推理延迟对比表。
* **Case Study**：展示 RAG 开启前后的回答对比截图。

## 5. 评分标准 (Grading Criteria)

| 维度 | 权重 | 评分细则 |
| --- | --- | --- |
| **基础架构** | 20% | 正确实现 RMSNorm, RoPE, SwiGLU；代码结构清晰，模块化程度高。 |
| **MoE 实现** | 25% | 门控网络逻辑正确；Top-k 路由有效；有针对专家负载均衡的分析或优化。 |
| **SFT 效果** | 20% | Loss 计算正确（正确处理 Mask）；模型能遵循指令对话，无明显重复生成。 |
| **量化优化** | 15% | 成功实现动态量化；压缩比达标；量化后模型输出逻辑不崩坏。 |
| **RAG 系统** | 20% | 向量检索逻辑跑通；Prompt 拼接正确；能显著修正幻觉问题。 |
| **报告质量** | +10% | **(附加分)** 分析深入（如对比不同专家数量的影响），图表精美，有独到见解。 |

## 6. 注意事项与提示 (Notes & Tips)

1. **显存管理**：
   * 在个人电脑上训练时，若显存（VRAM）不足，请减小 batch\_size（如设为 4 或 8）并使用 gradient\_accumulation\_steps（梯度累积）来模拟大 Batch。
   * 善用 torch.cuda.empty\_cache()。
2. **MoE 调试**：
   * MoE 模型容易出现“训练不稳定”或“专家坍缩”（即所有 Token 都只选同一个专家）。建议在 Loss 中加入**负载均衡辅助损失 (Load Balancing Loss)** 可作为进阶优化。
3. **量化陷阱**：
   * PyTorch 的动态量化主要加速 **CPU** 推理。在 GPU 上测试量化模型时，速度可能不会提升甚至变慢，这是正常的，重点关注**模型体积**和**CPU推理延迟**。
4. **学术诚信**：
   * 允许参考开源代码（如 Andrej Karpathy 的 nanoGPT），但必须在代码注释中注明引用来源，并需自己重构核心模块。直接 Copy-Paste 将被视为作弊。

### [附录] 推荐参考资源

* **Transformer 基础**: *"Attention Is All You Need"* (Vaswani et al.)
* **MoE 原理**: *"Switch Transformers"* (Google) 或 Mixtral 8x7B 技术报告。
* **代码参考**:
  + [Andrej Karpathy's nanoGPT](https://github.com/karpathy/nanoGPT) (极简实现，强烈推荐)
  + [PyTorch Quantization Documentation](https://pytorch.org/docs/stable/quantization.html)
  + [LangChain RAG Tutorial](https://python.langchain.com/docs/use_cases/question_answering/)