

2025



面向openKylin智能引擎的开源 大模型推理优化

提升端侧大模型推理效率，探索智能引擎优化新路径

 开源大模型 →  高效推理 →  技术创新

— OPENATOM OPENKYLIN —



开放原子开源基金会
OPENATOM FOUNDATION

| openKYLIn

1 赛事介绍

-  赛事背景
-  端侧大模型推理面临的挑战
-  openKylin智能引擎概述
-  openKylin智能引擎架构
-  openKylin智能引擎工作流程

2 赛题详情

-  赛题任务
-  提交成果要求
-  赛题技术方向

3 评审与环境

-  评审标准
-  环境配置指南
-  基础环境准备
-  llm-agent及依赖安装
-  运行与验证

4 技术路径与总结

-  技术路径示例：大模型压缩优化
-  模型量化实现方案
-  模型蒸馏技术路径
-  实验效果评估
-  总结与展望

openKylin AIPC版本是我国首个全开源的端侧操作系统智能引擎，它深度融合了AI技术，旨在为用户提供 在本地PC上运行AI助手、智能生成图像、模糊搜索等功能，提升用户体验和办公效率。



端侧设备的挑战

- 端侧设备的CPU/GPU/NPU算力和内存远低于数据中心
- 难以原生支撑参数规模庞大的大模型高效推理
- 全精度推理在端侧几乎不现实，亟需"瘦身"以适配端侧硬件



优化方向

- 大模型压缩技术：低比特量化、剪枝、蒸馏等
- 大模型推理加速算法：高效运算内核设计、并行流水线优化
- 异构协同推理：探索在多种计算单元间协同调度
- RISC-V架构优化：挖掘RISC-V架构中向量、矩阵等AI加速算力

本赛题旨在探索如何通过各种技术手段，提升大模型在openKylin智能引擎上的推理效率，使AI技术能够在资源受限的端侧设备上更好地发挥作用

端侧大模型推理面临的挑战



硬件资源受限

端侧设备的CPU/GPU/NPU算力和内存远低于数据中心，难以原生支撑参数规模庞大的大模型高效推理。



模型规模庞大

大模型通常需要数十GB显存和大量算力，全精度推理在端侧几乎不现实，亟需"瘦身"以适配端侧硬件。



实时交互需求

智能助手等应用对延迟要求极高，高延迟会严重影响用户体验，因此需要多层次优化以保证实时性。



能耗与散热

端侧设备长时间高负载运行大模型会导致功耗和发热问题，优化不仅要追求速度，更要注重能效比。



软硬件协同

端侧环境多样（x86、ARM、RISC-V等），充分利用不同架构的特色（如RISC-V的向量扩展）进行异构加速是一大挑战。



本地AI助手

提供智能对话、问答、文档处理等本地化AI服务，无需上传云端，保护用户隐私。



智能图像生成

基于文本描述生成相应的图像内容，支持创意设计和图像编辑等操作。



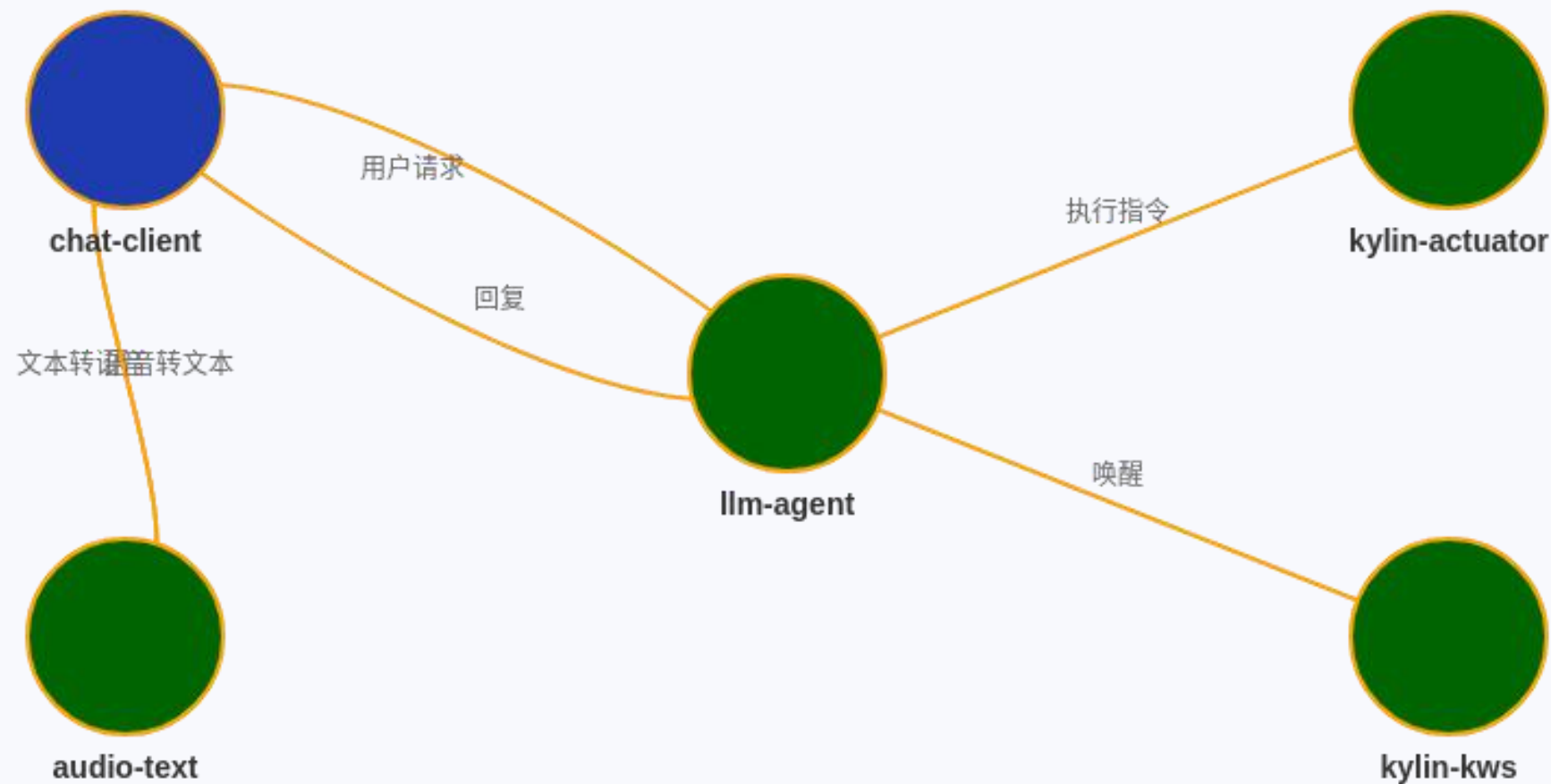
模糊搜索

通过自然语言描述进行精准搜索，支持复杂查询和智能推荐。

核心技术支持

openKylin智能引擎由五个核心子项目构成，共同协作实现端侧AI能力：

- ✔ **llm-agent**：端侧智能引擎核心，提供模型推理和用户意图识别
- ✔ **kylin-actuator**：系统执行引擎，接收指令并执行系统操作
- ✔ **chat-client**：智能助手应用前端，提供交互界面
- ② **audio-text**：语音输入输出技术，支持实时语音识别和合成
- ② **kylin-kws**：提供文本及语音唤醒服务



● 核心组件 数据/指令流 ● 用户交互

llm-agent：端侧智能引擎



开放原子开源基金会
OPENATOM FOUNDATION

openKyLin

概述

llm-agent是openKyLin智能引擎的核心组件，提供模型推理和用户意图识别服务。作为端侧智能引擎，它在openKyLinAIPC版本中扮演着关键角色，支持多种本地大模型并具有良好的扩展性。

核心功能

- 📌 模型推理：调用大语言模型进行理解和决策
- 🔍 用户意图识别：分析用户请求并识别操作意图
- ↔️ 命令转发：将识别出的指令发送给kylin-actuator执行
- 💬 回复生成：根据执行结果生成文字回复

代码仓库

<https://gitee.com/openkylin/llm-agent>

系统地位



支持的模型

九格 miniCPM DeepSeek 其他十余种

- ➕ 用户可根据需求选择与添加最合适的大模型
- 🔧 具有良好的扩展性，可动态加载不同模型

kylin-actuator：系统执行引擎

组件概述

kylin-actuator是openKylin智能引擎的系统执行引擎，作为连接智能助手与系统操作的桥梁，具有以下关键特性：

- ✓ 接收llm-agent的指令判断结果并执行相应系统操作
- ✓ 支持蓝牙控制、音量控制、系统设置等数百种系统操作指令
- ✓ 具备动态扩展能力，可适应未来新任务类型或系统操作需求
- ✓ 通过系统API完成相应操作，如打开蓝牙、降低屏幕亮度等

实现技术



开放原子开源基金会
OPENATOM FOUNDATION

openKyLin

指令执行流程



kylin-actuator接收到来自llm-agent的指令后，通过内部逻辑判断指令类型，并调用相应的系统API执行操作，如蓝牙控制、音量调整、系统设置等。

系统集成



作为openKylin智能引擎的关键组件，kylin-actuator与llm-agent紧密协作，形成从用户请求理解到系统操作执行的完整闭环，提供无缝的智能交互体验。

动态扩展能力



kylin-actuator支持动态加载新的指令处理模块，无需重启服务即可扩展功能，适应不断变化的系统操作需求和用户场景，具有极高的灵活性和可扩展性。

开源实现



代码地址：<https://gitee.com/openkylin/kylin-actuator>，采用C++/Shell实现，提供系统调用接口，具有跨平台兼容性。



系统智能助手应用前端

chat-client是openKylin智能引擎的前端界面，以llm-agent为后端服务，提供智能交互界面。用户可通过语音或文本输入与系统进行交互，实现多种智能操作。



会话管理

管理当前和历史会话，支持任务处理和对话内容富文本显示



模型切换

支持切换不同的大语言模型，根据需求选择最适合的AI助手



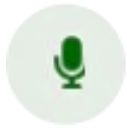
上下文选择

选择和管理对话上下文，使AI能更好地理解用户意图



语音播报

将文字回复转换为语音，通过音频设备播放，增强交互体验



语音输入

提供便捷的语音输入功能，实现免手写交互，支持多种语言

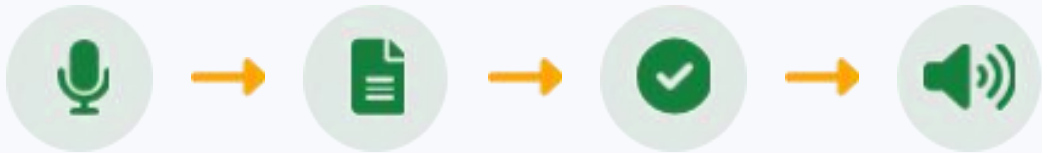


音色选择

可调整语音合成的音色和风格，提供个性化交互体验

 代码地址：<https://gitee.com/openkylin/chat-client>

audio-text为麒智-灵语智能助手提供语音输入输出技术支撑，实现实时语音识别转写、文本自动纠错和高质量自然人声语音生成，增强人机交互体验。



实时语音识别转写

将用户语音输入实时转换为文本，支持自然流畅的语音交互，降低用户使用门槛，提升交互效率。

AB 文本自动纠错

智能识别和纠正语音转写中的错误，提高文本识别的准确度，确保用户意图得到准确理解。

高质量语音生成

将智能助手的回复合成为自然人声语音，通过音频设备播放，提供更加自然、沉浸式的交互体验。

系统集成

audio-text模块与chat-client前端和llm-agent后端紧密集成，形成完整的语音输入输出闭环，支持用户通过语音与智能助手进行自然流畅的交互。

kylin-kws提供文本及语音唤醒服务，是openKylin智能引擎的关键组件之一

文本唤醒

功能描述

当用户选择文本内容时，kylin-kws会检测并触发相应操作菜单弹出，提供快捷操作选项。

应用场景

用户在使用"麒智-灵语"智能助手时，可以通过文本选择触发上下文菜单，快速切换模型、清除上下文或调整设置。

工作流程

- 用户在对话界面中选择特定文本
- kylin-kws检测到文本选择事件
- 分析上下文并确定合适的操作菜单
- 弹出操作菜单，用户可直接点击选项

语音唤醒

功能描述

kylin-kws能够检测并响应预设的语音唤醒词，唤醒"麒智-灵语"智能助手，激活AI交互。

应用场景

用户可以通过语音命令（如"你好麒麟"）唤醒智能助手，开始一次新的对话交互，系统会自动弹出对话界面。

工作流程

- 用户说出预定义唤醒词
- kylin-kws进行语音特征提取和比对
- 识别成功后，激活chat-client对话界面
- 用户可直接开始与AI助手对话



启动阶段

系统初始化，各组件准备就绪



唤醒交互

用户唤醒词触发，弹出对话界面



意图识别与决策

模型理解用户请求并生成操作命令



指令执行

系统执行操作并准备文字回复



语音反馈

文字回复转换为语音并播放



多轮对话支持

系统内置记忆模块，存储对话上下文，支持连续交互。用户可查看历史对话、切换模型或清除上下文，提高交互灵活性。

🎯 核心目标

提升大模型在**openKylin**智能引擎上的推理效率

针对端侧场景优化大模型的推理，使其在有限资源上运行更快，同时尽量保持模型精度。

参赛者需解决的问题

- 🔧 如何定制化大模型压缩策略，在降低模型规模的同时将精度损失降到最低?
- 👥 如何综合利用数据并行、张量并行、流水线并行等方法，设计最优并行方案?
- 📊 如何根据推理任务特征，动态将计算负载分配到最合适的计算单元?



✅ 提交要求

- 1 详细的技术方案描述
阐述所用方法（如算法流程、架构设计、压缩策略等）
- 2 实现代码
包含优化算法的源码，需可编译/运行
- 3 验证结果
通过实验对比展示优化效果，如推理吞吐量提升或延迟降低的数据，以及模型精度变化情况
- 4 可复现性
提供完整的环境配置说明和测试方法，确保评审者能够复现和检验优化效果



参赛队伍需提交以下四项成果，确保评审者能够复现和检验优化效果



详细的技术方案描述

- 阐述所用方法（如算法流程、架构设计、压缩策略等）
- 清晰说明技术路线和关键步骤
- 突出创新点和优化思路



实现代码

- 包含优化算法的源码
- 确保代码可编译/运行
- 提供完整的开发环境配置说明



验证结果

- 通过实验对比展示优化效果
- 提供推理吞吐量提升或延迟降低的数据
- 展示模型精度变化情况



可复现性

- 提供完整的环境配置说明
- 详细描述测试方法
- 确保评审者能够复现和检验优化效果

 注：提交的成果应确保完整性，所有材料需使用压缩包形式提交，文件命名为"团队名称-成果.zip"

本赛题涵盖面向openKylin端侧智能引擎的大模型推理优化的各种技术方向，参赛团队可以聚焦于一个或多个优化方向



大模型压缩技术

通过低比特量化、剪枝、蒸馏等方法，减少模型参数规模和计算量，加速推理



推理加速算法

高效运算内核设计、并行流水线优化、缓存调度策略，加快推理速度



异构协同推理

探索在多种计算单元间协同调度，将任务分配到不同硬件资源上并行计算



RISC-V架构优化

挖掘RISC-V架构中向量、矩阵等AI加速算力，构建易于使用的AI开发框架



其他创新方向

针对特定模型或应用场景的专用优化、自研轻量化推理框架等

注：以上方向并非相互独立，参赛者可结合多种方法形成混合优化方案

通过压缩技术减少模型参数规模和计算量，提升端侧推理效率



低比特量化

将模型权重从高精度（如FP16）转换为低精度（如INT8）表示

- 8-bit数值只需32-bit的1/4存储空间
- 使用高效整数指令，大幅提升推理速度
- 硬件（CPU/GPU）对int8运算有优化（如张量核心）



模型剪枝

移除模型中不必要的部分，保留关键结构

- 按权重重要性或连接强度移除部分网络结构
- 保持网络整体拓扑结构，减少参数量
- 可针对不同层使用不同剪枝比例



模型蒸馏

训练"小模型"（学生）来逼近"大模型"（教师）的行为

- 保持原始大模型输出，训练参数量更小的学生模型
- 成功案例：DistilBERT将参数量减少40%，推理速度提升60%
- 可引入多任务损失，同时最小化输出差异和提高学生表现

压缩技术效益

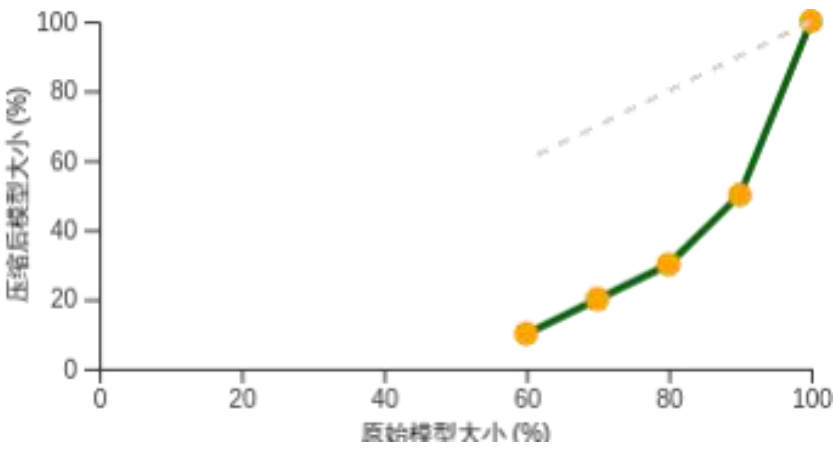
 推理速度提升

 显存占用大幅降低

 能耗降低

 精度损失可控

模型压缩效果示意



⚡ 通过优化算法设计与实现，充分利用硬件加速能力，提升大模型推理速度，降低延迟，提高吞吐量。



高效运算内核设计

- ✓ 针对Transformer架构优化的矩阵乘法内核
- ✓ 融合注意力机制的低精度计算策略
- ✓ 自适应计算图优化



并行流水线优化

- ✓ 数据并行与张量并行策略
- ✓ 深度流水线架构设计
- ✓ 多头注意力并行计算优化



缓存调度策略

- ✓ 分层注意力机制
- ✓ 高效中间结果复用策略
- ✓ 智能内存调度算法

💡 优化目标： 在最大化GPU吞吐量的同时满足部署资源约束和内存限制，提升推理速度。

异构协同推理方向探索在多种计算单元（ CPU、GPU、NPU ）间实现**协同调度**，根据任务特点将计算负载动态分配到最合适的计算单元并行执行，充分利用各类加速资源，提高整体能效。

关键技术要点

-  **动态任务分配**
根据推理任务的实时特征，将计算负载分配到最合适的计算单元
-  **负载均衡优化**
充分利用各类加速器资源，避免资源浪费和瓶颈
-  **高效通信机制**
减少不必要的数据传输开销，优化单元间数据流通

 异构协同的核心是“根据任务特点选择最优计算资源”，实现资源的智能调度与高效利用



RISC-V架构凭借其指令集灵活性和可定制性，在端侧AI领域具有独特优势，可通过优化提升openKylin智能引擎在资源受限设备上的性能。



RISC-V架构优化方向

- ✔ 挖掘RISC-V向量指令集加速潜力
- ✔ 利用可定制指令优势（如增加矩阵乘法指令）加速Transformer模型
- ✔ 构建针对AI计算优化的指令流
- ✔ 优化内存访问模式，提高cache效率



AI加速能力

- ✔ 向量扩展（Vector Extensions）加速矩阵运算
- ✔ 矩阵乘法硬件加速
- ✔ 低精度计算支持（如INT8/INT4）
- ✔ 多核并行计算能力优化



RISC-V AI开发框架

- ✔ 构建易于使用的RISC-V AI开发工具链
- ✔ 优化编译器以生成高效AI代码
- ✔ 提供模型转换与优化工具
- ✔ 封装常用AI算子库

通过RISC-V架构优化，可弥补生态不足，发挥其指令集灵活性的优势，为openKylin智能引擎提供性能提升。

除大模型压缩、推理加速、异构协同和RISC-V架构优化外，参赛者还可探索以下创新路径



针对特定模型的专用优化

针对九格、MiniCPM、DeepSeek等特定大模型的特点，设计定制化的推理优化方案，如模型特定的剪枝策略、量化方案或混合精度策略，以最大化各模型的性能潜力。



针对应用场景的优化

针对特定应用场景（如对话、生成、搜索等）的计算特点，设计针对性的推理路径和优化策略，如任务特定的上下文处理、多轮对话优化等，以提升在特定场景下的推理效率。



自研轻量化推理框架

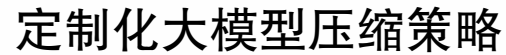
构建一个轻量级的推理框架，针对端侧资源受限的特点，从底层优化内存布局、计算流程和调度策略，实现更高效的推理。该框架可支持多种大模型格式和推理模式。



其他创新技术路径

探索其他创新技术如模型早停推理、自适应精度调整、多模型协同推理、内存优化策略等，或结合多种方法形成混合优化方案，以实现大模型在端侧环境下的高效推理。

注：以上方向并非相互独立，参赛者可结合多种方法形成混合优化方案



如何通过模型剪枝、蒸馏、低秩分解等技术，在保证用户体验不受明显影响的前提下，尽量保持精度的同时大幅减小大模型的计算和存储开销，使其适配资源受限的端侧设备？

评审标准



推理性能

30

评估指标：吞吐量(samples/s)、延迟(ms)、实时性
要求：提供优化前后的对比数据，提升幅度越大得分越高



精度保持

30

评估指标：分类准确率、召回率、 BLEU、ROUGE等
要求：在提速的同时，模型原有功能准确率不能大幅退化



适配性与部署性

20

评估内容：方案在不同硬件环境和系统架构上的适配性、部署流程的便捷性
要求：跨平台、易用性强的方案将更受青睐



创新性

10

要求：原创性强、具有首创性的方案可得高分

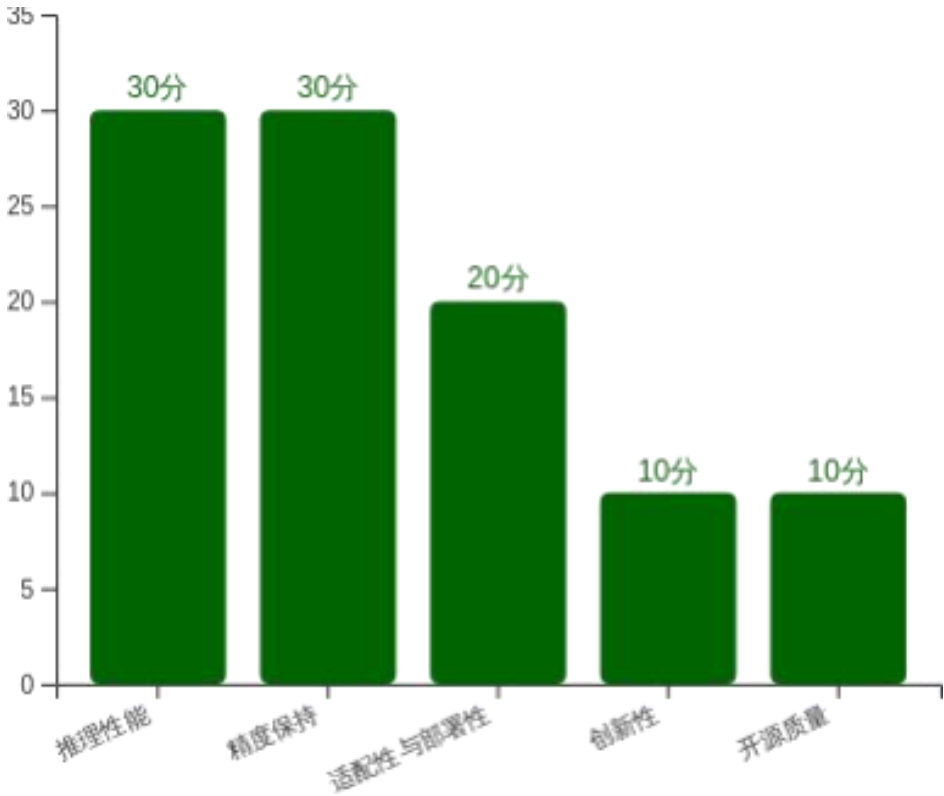


开源质量

10

要求：代码风格整洁、注释和文档完善、测试用例覆盖率高

评分分布



❗ 评审将综合平衡各项指标。例如，某方案性能提升极高但精度稍有下降，评委会将考虑其综合效益酌情给分。

★ 鼓励参赛者在性能与精度间寻找最优点。

要从零开始部署和运行openKylin智能引擎及大模型推理环境，请按照以下步骤配置：

基本环境要求

-  **操作系统**
建议使用最新版openKylin 2.0+操作系统，以获得完整的AI子系统支持
-  **硬件要求**
为了运行大模型并获得较好性能，GPU几乎是必备的。请准备带NVIDIA GPU的机器，显存容量取决于模型大小（至少8GB起，推荐16GB或更高）
-  **软件环境**
需要安装CUDA和cuDNN库，配置Python环境（推荐使用Anaconda或Miniconda创建隔离环境）

- 基础环境准备**
安装操作系统、硬件驱动和必要的开发工具，确保系统稳定运行
- 2 安装CUDA和cuDNN**
安装CUDA 11.8或以上版本和cuDNN 9.x，配置环境变量确保可用
- 3 安装llm-agent及依赖**
获取源码，创建Conda环境，安装PyTorch和项目依赖
- 4 运行与验证**
启动服务，测试接口，观察资源占用，记录Baseline数据



操作系统

- 1 建议使用最新版openKylin 2.0+操作系
- 2 统 确保系统已更新到最新状态
- 3 安装必要的开发工具和库



GPU硬件要求

- 1 准备带NVIDIA GPU的机器
- 2 显存容量取决于模型大小（至少8GB起）
- 3 推荐显存16GB或更高以获得较好性能



GPU驱动

- 1 安装对应GPU型号的NVIDIA驱动
- 2 确保CUDA环境可用
- 3 配置环境变量，确保驱动正确安装



CUDA和cuDNN

- 1 安装CUDA 11.8或以上版本
- 2 安装cuDNN 9.x
- 3 配置环境变量，确保nvcc等命令可用，并CUDA库路径加入 LD_LIBRARY_PATH



Python环境

- 1 安装Anaconda或Miniconda
- 2 创建隔离的Python环境运行llm-agent

 推荐使用Python 3.10版本，与llm-agent兼容

llm-agent及依赖安装



开放原子开源基金会
OPENATOM FOUNDATION

openKYLin

1 获取源码

克隆llm-agent项目仓库：

```
git clone https://gitee.com/openkylin/llm-agent.git
cd llm-agent
```

2 创建Conda环境

创建并激活Python 3.10环境：

```
conda create -n llm_env python=3.10 -y
conda activate llm_env
```

注：llm-agent当前适配Python 3.10，使用其他版本可能遇到兼容问题。

3 安装PyTorch

根据CUDA版本安装对应版本的PyTorch：

```
pip install torch==2.2.1 -f
https://download.pytorch.org/whl/cu118/torch_stable.htm
```

若官方源下载慢，可切换国内源或使用清华TUNA源。

4 安装大模型推理库

推荐使用AutoGPTQ库加载量化后的本地大模型：

```
pip install auto-gptq --no-build-isolation -f
https://huggingface.github.io/autogptq-
index/whl/cu118/
pip install pytrie
```

上述库有预编译的CUDA扩展，可高效加载GPTQ格式的4bit/8bit模型。

5 安装项目依赖

安装剩余项目依赖：

```
pip install -r requirements.txt
```

注：如部分依赖需编译，可能耗时较长，请耐心等待或提前安装好可选依赖。

💡 安装提示

- 确保CUDA和cuDNN版本与PyTorch兼容
- 使用虚拟环境隔离依赖，避免与其他项目冲突
- 若遇到权限问题，可尝试使用sudo或管理员权限

服务运行与测试

1 启动服务

在llm-agent项目目录下，直接运行主程序：

```
python main.py
```

首次运行会加载默认的大语言模型（如九格9B模型），这可能需要下载模型权重或从缓存加载，耗时较长。

2 测试接口

llm-agent提供开放的HTTP和WebSocket API。可通过HTTP POST请求 发送测试指令：

```
curl -X POST http://127.0.0.1:8080/reply -d  
'{"query": "今天天气怎么样? "}'
```

或使用项目自带的测试脚本：

3 多模型切换

llm-agent支持加载多个模型并动态切换。可尝试将默认模型换成更小的 模型以测试性能差异：

```
python main.py --model-name new-model-name
```

资源监控与基准测试

4 观察资源占用

使用以下工具监测GPU显存和显卡利用率，以及CPU占用：

```
nvidia-smi
```

记录Baseline（优化前）的吞吐量和延迟，以便后续优化时对比验证。

5 性能基准测试

在开始优化前，建议进行一次性能基准测试，记录原始性能数据：

```
python benchmarks/benchmark.py
```

该脚本会自动运行一系列测试并收集性能数据，为后续优化提供参考点。

6 关闭服务

测试完成后，可通过以下方式停止llm-agent服务：

```
Ctrl+C
```

或使用以下命令强制终止：

```
pkill -f main.py
```

示例：大模型压缩优化



开放原子开源基金会
OPENATOM FOUNDATION

openKyLin

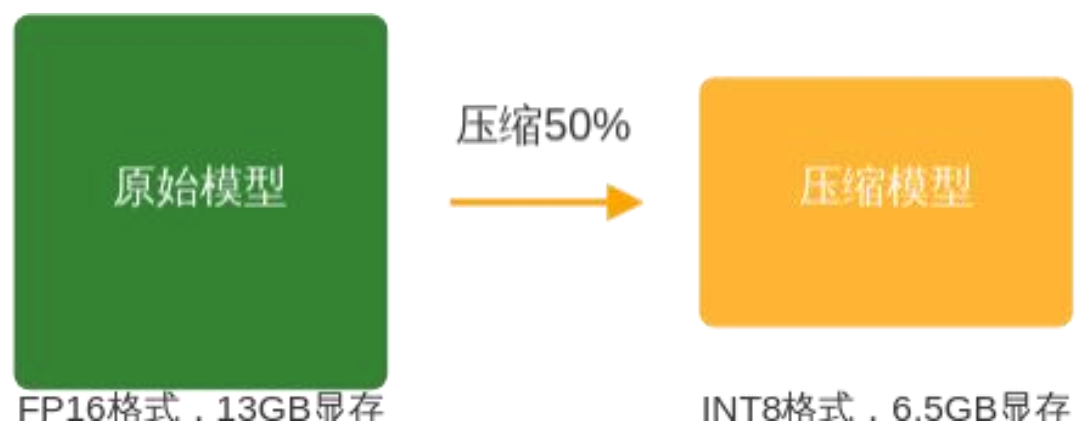
? 问题与目标

问题定义

- > 开源大语言模型（如7亿参数的中文GPT模型）在CPU上推理非常缓慢
- > GPU显存占用高，能耗大
- > 端侧设备资源受限，难以高效运行

优化目标

- > 通过压缩加速推理至少2倍
- > 保持输出效果接近原模型
- > 平衡精度损失与推理速度提升



压缩优化方案

模型压缩策略

低比特量化

将模型权重从FP16减少到INT8甚至更低，换取速度提升

模型蒸馏

训练"小模型"（学生模型）来逼近"大模型"（教师模型）

实施流程



基本原理

模型量化是将神经网络权重从高精度（如32位浮点）转换为低精度（如8位整数）表示，以减少内存占用和计算量。

- ✓ 8-bit数值只需32-bit的1/4存储空间
- ✓ 计算时使用高效的整数指令，大幅提升速度
- ✓ 许多硬件（CPU/GPU）对int8运算做了优化

工具选型

利用业界成熟工具AutoGPTQ进行权重量化：

- ✓ 实现GPTQ（高效近似量化算法）
- ✓ 支持将Transformer模型压缩为4-bit/8-bit表示
- ✓ 保持精度下降很小

实施步骤

1 离线量化

使用AutoGPTQ对模型进行离线量化

2 加载量化模型

修改llm-agent加载量化后的模型文件

3 性能测试

测试推理性能，与优化前对比

4 精度弥补

若精度下降明显，考虑结合微调或半精度混合策略

实现示例

利用Hugging Face Transformers库加载8-bit量化模型：

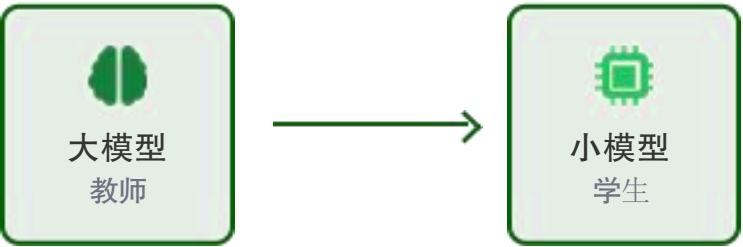
```
from transformers import AutoTokenizer

model_name = "THUDM/chatglm2-6b"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)
```

只需在加载时指定参数`load_in_8bit=True`

基本原理

- ✓ 知识蒸馏是通过训练"小模型"（学生模型）来逼近"大模型"（教师模型）行为的压缩技术
- ✓ 在保持原始大模型输出的前提下，训练参数量更小的学生模型，使其输出尽可能接近教师
- ✓ 蒸馏后，学生模型推理速度更快，但性能接近老师



应用建议

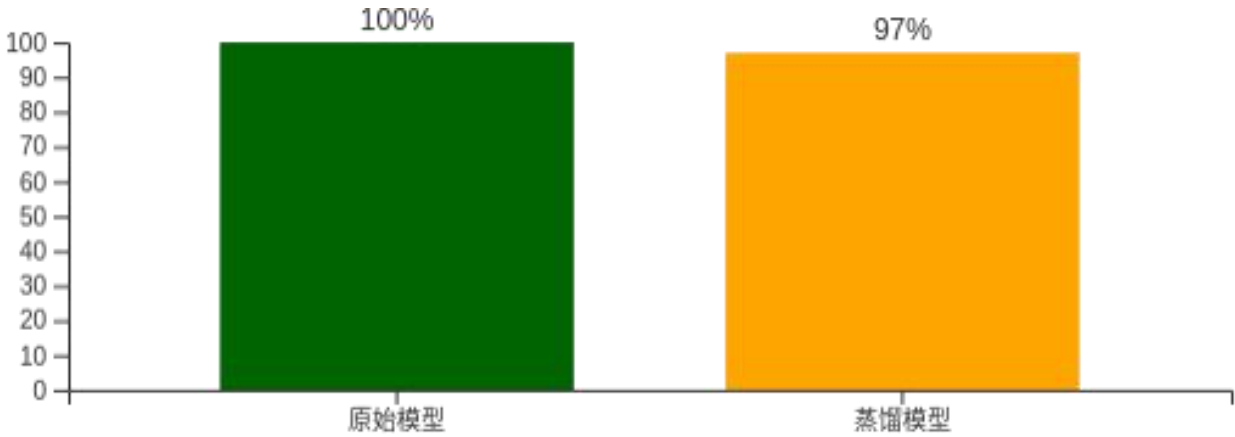
- 先用大模型跑出结果（作为教师），再训练小模型去学习这些结果
- 引入多任务损失：同时最小化输出logits差异和提高学生的自有任务表现
- 蒸馏过程需要一定训练成本，但可离线完成
- 离线训练完成后，线上推理效率提升巨大

成功案例：DistilBERT

DistilBERT是BERT-base的蒸馏版，通过精心设计的蒸馏过程，取得了显著的压缩效果：



性能对比



精心设计的蒸馏可以"大幅压缩模型且只带来极小的性能损失"，在实际应用中取得了显著的推理效率提升。

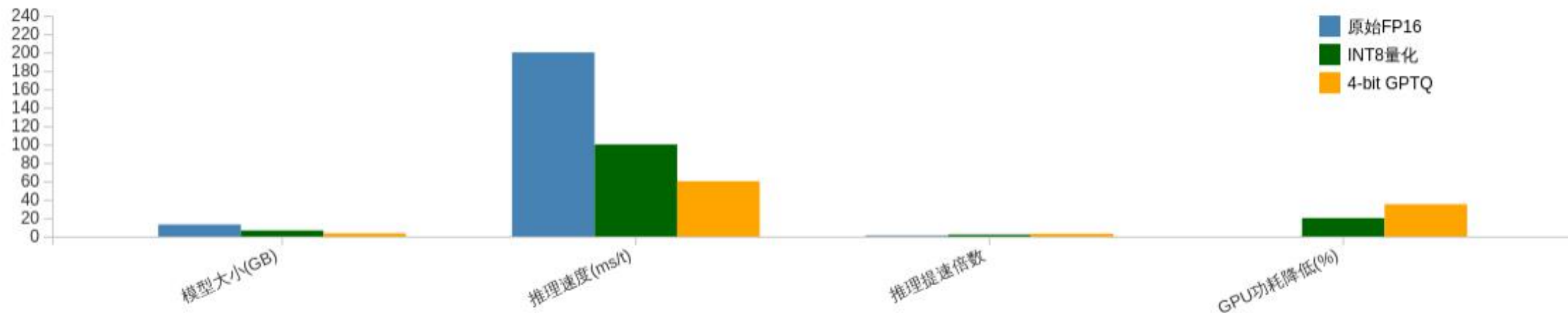
实验效果评估



开放原子开源基金会
OPENATOM FOUNDATION

openKYLIN

对中文7B参数模型进行量化实验，对比不同量化精度下模型性能：



性能提升

- INT8量化：推理速度提升约**2**倍
- 4-bit GPTQ：推理速度提升**3**倍
- + INT8量化：模型大小减少

50%

能耗与精度

- INT8量化：GPU功耗降低约**20%**
- 4-bit GPTQ：功耗进一步降低
- 精度影响：INT8量化模型语义正确率**>98%**

实验结论

模型压缩方案成功提升了大模型在端侧的推理效率，通过合理选择量化位宽，可实现性能与精度的平衡，使大模型在资源受限的环境中高效运行。

端侧AI的未来

openKylin智能引擎的出现标志着端侧设备AI能力的飞跃，从本地运行大模型助手，到跨应用智能处理，开源操作系统正在成为AI创新的重要土壤。

- ✓ 随着社区协作与本次大赛的推进，端侧大模型技术将快速发展
- ✓ 开源开放原子基金会将持续支持AI操作系统创新

优化挑战与机遇

挑战

- ❗ 硬件资源受限
- ❗ 模型规模庞大
- ❗ 实时交互需求

机遇

- ★ 多技术方向协同优化
- ★ 软硬件协同加速
- ★ 开源生态共建

资源支持

-  **开源代码与文档**
openKylin社区提供完整的源码和详细文档，支持参赛者理解系统架构和优化方向。
-  **算力支持**
开放原子开源基金会提供必要的算力资源支持，保障大赛顺利进行。
-  **社区交流**
社区讨论区欢迎交流想法、分享困难，相互启发，共同推进端侧AI发展。

■ 期待您的成果

希望通过这次比赛，能涌现一批优秀的优化方案，推动openKylin智能引擎更加高效实用。也期待同学们、开发者们从中收获成长，在国产开源AI操作系统领域留下自己的贡献！

 预祝各位参赛队伍取得优异成绩！

2025



开放原子开源基金会
OPENATOM FOUNDATION

| openKyLin

THANKS



www.openKylin.top



contact@openKylin.top