vLLM 二次开发

vLLM 是由加州大学伯克利分校 LMSYS 团队开发的一个高性能大语言模型（LLM）服务系统，以其高吞吐、低延迟和高效的内存管理（基于 PagedAttention 技术）而广受关注。如果你希望对 vLLM 进行二次开发，以下是一些关键方向和建议：

一、理解 vLLM 的核心架构

在开始二次开发前，建议深入理解 vLLM 的核心组件：

- **PagedAttention**：一种受操作系统虚拟内存分页启发的注意力机制，能有效减少显存碎片，提升吞吐量。
- **连续批处理（Continuous Batching）**：动态合并不同长度的请求，提高 GPU 利用率。
- **模型执行引擎**：基于 Hugging Face Transformers 和 PyTorch，但做了大量底层优化。
- **服务接口**：支持 OpenAI 兼容的 API、gRPC 和 WebSocket。

二、常见二次开发方向

1. 支持新模型架构

vLLM 官方支持主流模型（如 Llama、ChatGLM、Qwen、Baichuan 等），但如果你有自定义模型或未被支持的架构（如 MoE 模型、特定微调模型），可以：

- 继承 vllm.model_executor.models.BaseModel
- 实现 forward 和 prepare_inputs 方法
- 注册模型到 vLLM 的模型注册表中

示例（简化）：

```
from vllm.model_executor.models import register_model
from vllm.model_executor.models.llama import LlamaForCausalLM
```

```
@register_model("my-custom-model")
class MyCustomModel(LlamaForCausalLM):
    def __init__(self, config):
        super().__init__(config)
        # 自定义修改
```

2. 添加自定义推理逻辑

- **修改采样策略**：如加入自定义的 temperature 调度、Top-k/Top-p 变体、重复惩罚等。
- **实现新的生成控制**：如基于规则的停止条件、动态提示工程。
- **集成外部工具**：如调用数据库、搜索引擎（可通过中间件或后处理实现）。

3. 优化性能或适配新硬件

- 针对特定 GPU（如 H100、A100）进行 kernel 优化。
- 支持量化模型（如 AWQ、GPTQ）——vLLM 已部分支持，可扩展。
- 集成 FlashAttention 2 或其他高效注意力实现。

4. 扩展服务功能

- 添加用户认证、请求限流、日志记录。
- 支持流式输出（vLLM 已支持，但可定制格式）。
- 与 LangChain、 LlamaIndex 等框架集成。

5. 监控与可观测性

- 集成 Prometheus / Grafana 监控 GPU 使用率、请求延迟、队列长度。
- 添加推理性能分析工具（如 PyTorch Profiler）。

三、开发建议

1. **从官方 GitHub 克隆最新代码**

https://github.com/vllm-project/vllm

2. **使用 Docker 或 Conda 环境**

官方提供 Dockerfile，便于环境复现。

3. **阅读官方文档中的“自定义模型”和“开发指南”章节**

https://vllm.readthedocs.io/en/latest/

4. **先通过 vllm.entrypoints.openai.api_server 或 vllm.LLM 类做快速验证**
5. **贡献代码**：如果你开发的功能具有通用性，可考虑向官方提交 PR。

四、注意事项

- vLLM 依赖 PyTorch 2.x 和 CUDA，需确保环境兼容。
- 底层 C++/CUDA 代码较复杂，如需修改 PagedAttention 等核心模块，需具备 CUDA 开发经验。
- 二次开发后需重新编译（尤其修改了 .cu 或 .cpp 文件）。