PR #24907 完整报告

sgl-project/sglang

[MLX] Add on-the-fly --quantization mlx_q4 / mlx_q8 for Apple Silicon

合并时间：2026-05-14 02:06

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24907

执行摘要

- 一句话：MLX 后端新增 mlx_q4/q8 即时量化支持
- 推荐动作：建议精读此 PR，尤其关注以下设计决策：
  1. 如何通过标记配置类（MlxQuantizationConfig）避免后端代码侵入并行配置架构；
  2. 如何利用 MLX 元数据在 _load_model 中计算内存节省而不强制实例化权重；
  3. 如何通过 is_mps() 条件注册实现平台感知的量化方法集合；
  4. 测试文件的注册与自动跳过模式，可作为跨平台测试的范式。

功能与动机

Apple Silicon 支持已通过 #19549 和 #20342 落地，但量化（roadmap 中 TBD）是缺失的关键一环。PR 数据显示，对 Qwen3-0.6B 使用 mlx_q4 可将活跃内存从 1.110 GB 降至 0.312 GB（-71.9%），使 Qwen3-32B 等大模型能在 64 GB Mac 上运行。用户无需等待社区发布预量化版本，可直接从官方 HF 仓库加载 fp16 模型并即时量化。

实现拆解

1. 注册量化名称：在 server_args.py 的 QUANTIZATION_CHOICES 中追加 mlx_q4 和 mlx_q8，使命令行参数可被识别。
2. 创建标记配置类：新增 layers/quantization/mlx.py，定义 MlxQuantizationConfig 继承自 QuantizationConfig，其 from_config 和 get_quant_method 抛出清晰错误提示用户设置 SGLANG_USE_MLX=1。此类仅在 MPS 后端通过 is_mps() 条件注册到 BASE_QUANTIZATION_METHODS，确保非 Apple 平台快速失败。
3. 集成到模型加载流程：修改 hardware_backend/mlx/model_runner.py，在 MlxModelRunner.__init__ 中接受 quantization 参数并传入 _load_model；在 _load_model 中，首先通过 mlx_lm.load(..., return_config=True) 加载模型和配置，然后根据预设（_MLX_QUANTIZATION_PRESETS）调用 mlx_lm.utils.quantize_model 对未量化模型进行原地量化，并利用 tree_flatten 元数据计算量化前后的内存占用并记录日志。
4. 传递参数：在 hardware_backend/mlx/tp_worker.py 中将 self.server_args.quantization 传递给 MlxModelRunner 的 quantization 参数，完成完整参数链路。
5. 测试与文档：新增 test/registered/unit/hardware_backend/mlx/test_quantization.py（6 个测试用例），使用 register_cpu_ci 注册并通过 @unittest.skipUnless 在非 Apple

Silicon 平台自动跳过；文档更新 apple_metal.mdx 新增 'Quantization' 小节。

关键文件：

- python/sglang/srt/layers/quantization/mlx.py（模块 量化配置；类别 source；类型 dependency-wiring；符号 MlxQuantizationConfig, init, get_name, get_supported_act_dtypes）：新增标记配置类 MlxQuantizationConfig，使量化名称可被注册识别，但不执行实际量化，是后端隔离的关键桥梁。
- python/sglang/srt/hardware_backend/mlx/model_runner.py（模块 模型运行器；类别 source；类型 data-contract）：核心改动：添加 quantization 参数，在 _load_model 中调用 mlx_lm.quantize_model 进行原地量化，并通过元数据计算内存节省日志。
- test/registered/unit/hardware_backend/mlx/test_quantization.py（模块 测试；类别 test；类型 test-coverage；符号 TestMlxQuantization, _module_counts, _reset_mlx_memory, _build_runner）：新增 6 个单元测试覆盖量化模块创建、内存减少、生成验证、预量化模型回归等，通过 register_cpu_ci 注册并在非 Apple 平台自动跳过。
- python/sglang/srt/layers/quantization/__init__.py（模块 注册入口；类别 source；类型 dependency-wiring）：导入 MlxQuantizationConfig 并通过 is_mps() 条件注册到 BASE_QUANTIZATION_METHODS，与现有 mxfp4 注册模式一致。
- python/sglang/srt/server_args.py（模块 参数配置；类别 source；类型 core-logic）：在 QUANTIZATION_CHOICES 中添加 mlx_q4 和 mlx_q8，使命令行参数可被框架解析。
- python/sglang/srt/hardware_backend/mlx/tp_worker.py（模块 工作器；类别 source；类型 core-logic）：将 server_args.quantization 传递给 MlxModelRunner 的 quantization 参数，完成参数链路。
- docs_new/docs/hardware-platforms/apple_metal.mdx（模块 文档；类别 other；类型 core-logic）：新增 'Quantization' 文档小节，说明两种量化路径并给出示例命令。

关键符号：MlxQuantizationConfig.init, MlxQuantizationConfig.from_config, MlxModelRunner.init, MlxModelRunner._load_model, TestMlxQuantization.test_mlx_q4_creates_quantized_linear_modules, TestMlxQuantization.test_mlx_q4_reduces_memory_vs_fp16, TestMlxQuantization.test_mlx_q8_creates_quantized_linear_modules, TestMlxQuantization.test_mlx_q4_generates_text

关键源码片段

python/sglang/srt/hardware_backend/mlx/model_runner.py

核心改动：添加 quantization 参数，在 _load_model 中调用 mlx_lm.quantize_model 进行原地量化，并通过元数据计算内存节省日志。

```
# 定义支持的量化预设：name -> (bits, group_size)
_MLX_QUANTIZATION_PRESETS: dict[str, tuple[int, int]] = {
    "mlx_q4": (4, 64),
    "mlx_q8": (8, 64),
}

def _load_model(self) -> None:
```

```
    """加载模型，若指定量化预设则原地量化 fp16 权重。"""
    logger.info(f"Loading MLX model: {self.model_path}")
    start_time = time.time()

    # 加载模型及配置，return_config=True 可获取 HF 配置
    loaded = mlx_lm_load(
        self.model_path,
        tokenizer_config={"trust_remote_code": self.trust_remote_code},
        return_config=True,
    )
    self.model, _tokenizer, config = loaded

    if self._quantization in _MLX_QUANTIZATION_PRESETS:
        bits, group_size = _MLX_QUANTIZATION_PRESETS[self._quantization]
        # 若模型已是量化版本（config 包含 "quantization" 键）则跳过
        if "quantization" in (config or {}):
            logger.info(
                "Model already pre-quantized; "
                f"ignoring --quantization={self._quantization}"
            )
        else:
            # 通过 tree_flatten 获取权重元数据，不实际 materialize 权重
            from mlx.utils import tree_flatten
            bytes_before = sum(
                p.size * p.itemsize
                for _, p in tree_flatten(self.model.parameters())
            )
            # 执行原地量化
            mlx_lm_quantize_model(
                self.model, config, bits=bits, group_size=group_size
            )
            bytes_after = sum(
                p.size * p.itemsize
                for _, p in tree_flatten(self.model.parameters())
            )
            logger.info(
                f"Quantized {bytes_before/1024**3:.2f} GB -> "
                f"{bytes_after/1024**3:.2f} GB "
                f"({(1 - bytes_after/bytes_before)*100:.1f}% reduction)"
            )
    # 其余初始化 ...
```

test/registered/unit/hardware_backend/mlx/test_quantization.py

新增 6 个单元测试覆盖量化模块创建、内存减少、生成验证、预量化模型回归等，通过 register_cpu_ci 注册并在非 Apple 平台自动跳过。

```
@unittest.skipUnless(
    platform.system() == "Darwin" and platform.machine() == "arm64"
    and importlib.util.find_spec("mlx") and importlib.util.find_spec("mlx_lm"),
```

```
    "Apple-Silicon-only test (requires Darwin/arm64 + mlx + mlx_lm)"
)
class TestMlxQuantization(unittest.TestCase):
    """MLX 量化冒烟测试集合。"""

    @staticmethod
    def _module_counts(model) -> tuple[int, int]:
        """统计模型中 QuantizedLinear 和 Linear 模块数量。"""
        n_quant = n_linear = 0
        for _, m in model.named_modules():
            cls_name = type(m).__name__
            if cls_name == "QuantizedLinear":
                n_quant += 1
            elif cls_name == "Linear":
                n_linear += 1
        return n_quant, n_linear

    def test_mlx_q4_creates_quantized_linear_modules(self):
        """加载 mlx_q4 后所有 Linear 应被替换为 QuantizedLinear。"""
        self._reset_mlx_memory()
        runner = self._build_runner(_TEST_MODEL, "mlx_q4")
        try:
            n_quant, n_linear = self._module_counts(runner.model)
            self.assertGreater(n_quant, 0) # 至少有一个 QuantizedLinear
            self.assertEqual(n_linear, 0) # 不应存在未量化的 Linear
        finally:
            del runner
            self._reset_mlx_memory()
```

评论区精华

review 中三个核心讨论：

- mx.eval 与内存统计正确性：gemini-code-assist 指出 MLX 权重是惰性加载的，直接读取 mx.get_active_memory() 可能得到近零值，建议先调用 mx.eval(self.model.parameters()) 再统计。作者在 v2 中加入该调用，但在 v3 中改用 tree_flatten 读取元数据（p.size * p.itemsize）避免将全部 fp16 权重强制实例化导致 OOM。最终方案是零开销的元数据路径，日志信息形式不变。
- 注册方式选择：yeahdongcn 建议将 MLX 量化名称的注册从硬编码在 ModelConfig._verify_quantization 改为使用 if is_mps() 条件门控，与 mxfp4 的注册模式一致。作者采纳并移除了内联注释，使代码块结构完全镜像。
- 量化日志必要性：yeahdongcn 质疑量化前后的内存日志是否必要；作者解释量化过程（如 Qwen3-32B）可能耗时近 8 分钟，无进度条情况下用户会以为服务挂起，日志是唯一反馈。最终保留日志。
  - mx.eval 与内存统计正确性 (performance): 作者先采用 mx.eval，后改用 tree_flatten 读取元数据避免强制实例化，最终保留元数据方案并记录内存日志。
  - 注册方式选择 (design): 作者采纳并移除了内联注释，使代码块结构完全镜像。

- 量化日志必要性 (question): 作者解释量化过程可能耗时近 8 分钟，日志是唯一反馈，最终保留日志。

风险与影响

- 风险：
  1. 精度损失：4/8 比特权重量化会带来约 0.5%/ 层的累积精度损失，虽然符合预期但用户需要知晓。
  2. 平台隔离：mlx_q4/q8 仅在 is_mps() 时注册，但若用户在非 Apple 环境通过 PyTorch 路径错误实例化 MlxQuantizationConfig 会得到清晰错误。
  3. 测试覆盖：单元测试仅在 Apple Silicon + mlx 可用时运行，CI 自动跳过，可能遗漏跨平台回归。
  4. 量化过程无进度：长时间无输出可能被误判为挂死，日志行虽缓解但不如进度条直观。 - 影响：对用户：Apple Silicon 用户现在可以通过 --quantization mlx_q4/q8 直接加载任意 HF 仓库的 fp16 模型并即时量化，可运行原本超内存的模型（如 Qwen3-32B）。无需社区预量化版本，降低了使用门槛。对系统：量化后模型内存占用大幅降低，但量化过程 CPU/GPU 开销较大，启动时间延长。对团队：MLX 量化逻辑完全隔离在 hardware_backend/mlx/ 中，后端无关代码无需修改，降低维护负担。 - 风险标记：新增后端能力，精度损失可预期，测试仅限 Apple Silicon，量化过程无进度反馈

关联脉络

- PR #19549 Initial macOS plumbing + stubs: 此 PR 是 Apple Silicon 支持的第一层，为后续 MLX 后端落地奠定基础。
- PR #20342 Native MLX execution backend: 此 PR 实现了完整的 MLX 模型执行后端（MlxModelRunner、KV cache、调度器集成），是当前量化功能的前置依赖。