

PR #1950 完整报告

radixark/miles

Support non-finite floats across the wire

合并时间: 2026-09-04 13:27

原文链接: <http://prhub.com.cn/radixark/miles/pull/1950>

执行摘要

- 一句话: 引入 `_NonFiniteFloatCodec`, 修复 NaN/Inf 在 RPC 线上丢失的问题。
- 推荐动作: 该 PR 值得精读。其核心设计——通过引入一个轻量级递归编解码器来处理 JSON 序列化的固有缺陷——清晰、有效且可扩展。实现集中、测试完备, 是处理类似数据完整性问题的优秀范例。关注点应放在 `_NonFiniteFloatCodec` 的递归实现和其在 `RpcSerializer` 中的无缝集成上。

功能与动机

PR 标题为 'Support non-finite floats across the wire', PR body 为空。结合实现上下文 (如测试用例 `test_non_finite_result_survives_the_wire`), 核心动机是修复 RPC 通信中因 JSON 不允许 NaN/Inf 导致的特殊浮点值丢失问题, 确保如损失值、梯度范数等训练指标在 Worker 和客户端间传输的完整性。

实现拆解

1. 引入非有限浮点数编解码器: 在 `miles/utils/workers/rpc/common/serialization.py` 中新增 `_NonFiniteFloatCodec` 类。该类提供 `encode` 和 `decode` 两个类方法, 分别负责在序列化前将非有限浮点数 (NaN, Inf, -Inf) 递归转换为特定字典格式 (如 `{'__miles_non_finite_float__': 'nan'}`), 并在反序列化后还原。原因: 标准 JSON 库的 `allow_nan=False` 无法处理这些值, 此编解码器提供了安全、递归的转换层。影响: 为 RPC 序列化核心路径增加了数据保真度。
2. 更新 RPC 序列化器: 修改 `RpcSerializer` 的 `encode_query`、`decode_query`、`encode_result`、`decode_result` 四个方法, 在原有的 Pydantic 模型序列化 / 反序列化步骤前后, 分别调用 `_NonFiniteFloatCodec.encode` 和 `_NonFiniteFloatCodec.decode`。原因: 确保所有进出 RPC 系统的负载都经过编解码器处理, 无论其类型注解如何。影响: 这是变更的核心集成点, 将编解码器应用于所有 RPC 通信。
3. 调整 Pydantic 序列化配置: 将全局 `_WIRE_CONFIG` 增加 `ser_json_inf_nan='constants'` 配置项。原因: 配合 `_NonFiniteFloatCodec`, 确保 Pydantic 层面也能识别这些特殊值, 而非在 dump 阶段就出错。影响: 配置与编解码器协同工作, 共同完成对非有限浮点数的支持。
4. 增加单元测试与端到端测试: 在 `tests/fast/utils/workers/rpc/common/test_serialization.py` 中新增 `TestNonFiniteFloatRoundtrip`、`TestNonFiniteFloatValues` 等测试类, 覆盖各种数据结构 (标量、字典、列表、模型、数据类) 中的非有限浮点数往返。在

tests/fast/utils/workers/e2e/test_complex_types.py 中新增 TestNonFiniteFloats 测试类，通过模拟真实的 RPC Worker 通信验证端到端正确性。原因：确保新机制的正确性和鲁棒性。影响：提供了坚实的测试保障。

5. 扩展端到端测试工作器：在 tests/fast/utils/workers/e2e/e2e_worker.py 中新增 Metric 模型和一系列演示方法（如 demo_nan_result, demo_float），为端到端测试提供支持。原因：为端到端测试用例提供必要的 Worker 端逻辑。影响：使集成测试成为可能。

关键文件：

- miles/utils/workers/rpc/common/serialization.py（模块 序列化；类别 source；类型 dependency-wiring；符号 _NonFiniteFloatCodec, encode, decode）：核心变更文件，引入 _NonFiniteFloatCodec 并集成到 RpcSerializer，直接修复非有限浮点数传输问题。
- tests/fast/utils/workers/rpc/common/test_serialization.py（模块 序列化测试；类别 test；类型 test-coverage；符号 Reading, _through_the_strict_wire, _equal_including_non_finite_floats, TestNonFiniteFloatRoundtrip）：关键测试文件，新增全面覆盖非有限浮点数往返的单元测试，验证编解码器的正确性。
- tests/fast/utils/workers/e2e/test_complex_types.py（模块 复杂类型测试；类别 test；类型 test-coverage；符号 test_non_utc_datetime_keeps_its_offset, TestNonFiniteFloats, test_nan_result_arrives_as_nan, test_float_argument_and_result_roundtrip）：端到端测试文件，验证非有限浮点数在真实 RPC 通信中的完整性。
- tests/fast/utils/workers/e2e/e2e_worker.py（模块 测试工作器；类别 test；类型 test-coverage；符号 Metric, demo_nan_result, demo_float, demo_optional_float）：端到端测试工作器，新增支持非有限浮点数测试的演示方法和数据模型。

关键符号：_NonFiniteFloatCodec.encode, _NonFiniteFloatCodec.decode, RpcSerializer.encode_query, RpcSerializer.decode_query, RpcSerializer.encode_result, RpcSerializer.decode_result

关键源码片段

miles/utils/workers/rpc/common/serialization.py

核心变更文件，引入 _NonFiniteFloatCodec 并集成到 RpcSerializer，直接修复非有限浮点数传输问题。

```
class _NonFiniteFloatCodec:
    """递归编解码器，用于在 JSON 序列化中安全地传输非有限浮点数 (NaN, Inf, -Inf)。"""

    @classmethod
    def encode(cls, value: Any) -> Any:
        """递归编码：将数据结构中的非有限浮点数替换为特定字典标记。"""
        if isinstance(value, float) and not math.isfinite(value):
            # 将 NaN/Inf/-Inf 映射到对应的字符串 token，并包装在特定字典中
            return {NON_FINITE_FLOAT_TAG: _TOKEN_BY_FLOAT.get(value, "nan")}
        if isinstance(value, dict):
            # 保留键冲突检测：如果用户数据中包含标记键，则抛出异常
            if NON_FINITE_FLOAT_TAG in value:
                raise ValueError(f"rpc payloads must not contain the reserved key {NON_FINITE_
```

```

        FLOAT_TAG!r}")
    # 递归编码字典中的每个值
    return {key: cls.encode(item) for key, item in value.items()}
if isinstance(value, list):
    # 递归编码列表中的每个元素
    return [cls.encode(item) for item in value]
# 其他类型（如 int, str, None）直接返回，无需处理
return value

@classmethod
def decode(cls, value: Any) -> Any:
    """递归解码：将特定字典标记还原为对应的非有限浮点数。"""
    if isinstance(value, dict):
        token = value.get(NON_FINITE_FLOAT_TAG)
        # 判断当前字典是否恰好是标记格式：只有一个键且键值是有效 token
        if len(value) == 1 and token in _FLOAT_BY_TOKEN:
            return _FLOAT_BY_TOKEN[token]
        # 否则，递归解码字典中的每个值
        return {key: cls.decode(item) for key, item in value.items()}
    if isinstance(value, list):
        # 递归解码列表中的每个元素
        return [cls.decode(item) for item in value]
    # 其他类型直接返回
    return value

```

评论区精华

Review 审核由 guapisolo 在 stack 上下文下完成，通过了 safe-merge checkpoint #2176。评审意见为：'The serialization design holds, and no end-to-end-backed blocker remains. LGTM.' 这表明评审认可了序列化设计，并确认没有端到端测试覆盖的阻塞性问题。本次 PR 未发现公开的争议或设计权衡讨论。

- PR 审核 (other): PR 被批准并合并。

风险与影响

- 风险：
 1. 性能开销：_NonFiniteFloatCodec.encode/decode 在每次 RPC 编解码时都会递归遍历整个数据结构。对于包含大量基本类型的大型负载，这可能引入额外的性能开销。具体文件 / 逻辑：serialization.py 中的递归遍历逻辑。
 2. 保留键冲突：编解码器使用固定键 __miles_non_finite_float__ 作为标记。如果用户数据中意外包含此键（尽管概率低），编解码器会抛出 ValueError。具体文件 / 逻辑：encode 方法中的冲突检测逻辑。
 3. 与现有逻辑的兼容性：此次变更为 RPC 序列化的全局行为引入了新的处理层，可能影响所有使用 RpcSerializer 的路径，需要确保无回归。具体文件 / 逻辑：RpcSerializer 的四个方法。- 影响：对用户：确保了通过 RPC 传输的训练指标（如 NaN/Inf 损失值）能被正确接收和处理，避免了数据静默丢失。对系统：这是 RPC 序列化模块（

miles.utils.workers.rpc.common.serialization) 的核心变更，直接影响 Worker 与客户端间所有类型化数据的传输，影响范围广但精确。对团队：引入了新的设计模式（编解码器递归处理），需要团队成员理解并维护；测试覆盖全面，降低了引入缺陷的风险。- 风险标记：序列化路径变更，递归性能开销，保留键冲突

关联脉络

- PR #2006 Launch inference engines through the inference engine worker spec: 两者均涉及 RPC 工作器基础设施的改进与规范化，本 PR 修复了其中的序列化数据完整性问题。