arXiv:2603.27462v1 [cs.DS] 29 Mar 2026

# RSR-core：面向低比特矩阵-向量乘法的高性能引擎

Mohsen Dehghankar 伊利诺伊大学芝加哥分校
mdehgh2@uic.edu

Abolfazl Asudeh 伊利诺伊大学芝加哥分校
asudeh@uic.edu

## 摘要

矩阵-向量乘法是神经网络、向量数据库和大语言模型中的基础构件，尤其是在推理过程中。因此，高效的矩阵-向量乘法引擎能够直接提升推理效率。

近期研究探索了模型权重的低比特量化，即使用二进制（1 位）或三元（1.58 位）数值表示矩阵，同时将激活值保持在较高精度。这些表示方式能够实现高效的硬件级计算。与此同时，冗余分段归约（RSR）等算法为加速低比特矩阵-向量乘法提供了理论保证。然而，现有实现运行于应用层，无法高效集成到硬件内核中，因而限制了实际性能。

为弥合这一差距，我们推出了 RSR-core。这是一款高性能引擎，可在 CPU 和 CUDA 环境中将 RSR 算法实现为经过优化的底层内核。RSR-core 支持对二进制和三元权重矩阵与通用向量进行高效的矩阵-向量乘法，同时使 RSR 算法能够实际部署到真实的推理流水线中。

RSR-core 是一款可用于生产环境的引擎，集成了 HuggingFace，可对低比特模型进行预处理并运行加速推理。

实验结果表明，与 HuggingFace PyTorch 基线乘法相比，性能显著提升：CPU 上最高可实现 62× 倍加速；对于热门三值大语言模型，在 CUDA 上进行令牌生成时最高可实现 1。9× 倍加速。源代码已公开发布于 RSR-core 仓库[1]。

## 1 引言

推理是神经网络中的关键阶段，对于大语言模型（LLM）尤为如此，因为它在很大程度上决定了能耗和资源分配 [7, 12]。推理过程中的一个关键计算瓶颈是矩阵-向量乘法 $Mv$，其中，矩阵 $M$ 表示训练后固定的模型权重，而向量 $v$ 则对应执行过程中生成的输入激活值。因此，提高矩阵-向量乘法的效率可直接加快神经模型推理速度，并提升资源利用效率。

高效的矩阵-向量乘法也是许多神经推理之外的数据管理和检索工作负载的核心 [1, 3]。例如，向量数据库系统会计算查询向量与索引数据点之间的内积，以检索最近邻；其中，索引数据集可视为矩阵，而查询则可视为输入向量 [4]。类似地，对表格数据集进行排序时，会将用户定义的权重向量应用于属性列，以计算用于排序和筛选的分数；其中，表格的行构成矩阵，而权重向量则定义排序函数 [9, 10]。

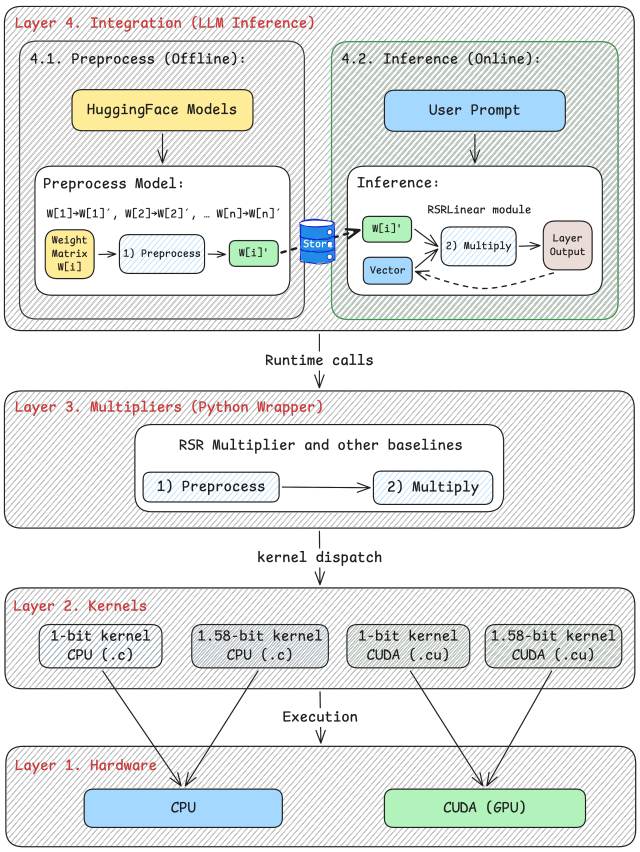



**图 1：RSR-core 系统的分层架构。**

在模型推理中，加速矩阵–向量乘法的一种常见方法是对权重矩阵进行低比特量化 [6, 11]。在低比特量化大语言模型中，矩阵以二进制或三值表示，而激活向量则保持较高精度，从而能够采用专门的硬件高效实现，降低推理过程中的计算成本。

与硬件感知量化方法不同，算法方法旨在进一步加速矩阵-向量乘法。具体而言，冗余分段归约（*RSR*） [2] 利用矩阵结构中的冗余，将低比特矩阵-向量乘法的计算成本降低一个对数因子。

[1]https://github.com/UIC-InDeXLab/RSR-core

具体来说，相同的列模式[2] 会产生相同的部分结果，因此可以在乘法前进行聚合，从而减少所需的总运算次数，且不影响推理精度。然而，现有的 RSR 实现运行在应用层，无法高效集成到硬件内核级执行中，限制了其在实际推理系统中的性能收益。

在这项工作中，我们通过提供 RSR 算法的内核级实现，并使其能够部署于实际的推理流水线中，弥补了这一空白。据我们所知，这是首个同时针对 CPU 和 CUDA 推理环境进行优化的 RSR 实现。

我们推出 **RSR-core**，这是一款高性能引擎，通过优化的 CPU 和 CUDA 内核实现 RSR，从而高效执行低比特矩阵-向量乘法。RSR-core 提供可用于生产环境的 Python 接口，并集成至 HuggingFace 工作流 [5]，让用户能够一次性预处理量化模型，并通过端到端工作流高效执行加速推理。

我们将 RSR-core 与 PyTorch [8]后端提供、并在 HuggingFace 推理接口中使用的广泛应用的硬件优化矩阵乘法实现，以及 BitNet [11]进行了基准测试。实验结果表明，对于常见的三值 LLM 推理工作负载，与 PyTorch bfloat16 计算相比，CPU 上的加速比最高可达 62×，CUDA 上最高可达 1.9×，且推理精度完全不变。

## 2 系统概述

我们考虑矩阵-向量乘法 $M \cdot v$，其中 $M$ 是一个元素为 $\{0,1\}$ 的二值矩阵，或一个元素为 $\{-1, 0, +1\}$ 的三值矩阵。该矩阵对应量化模型权重，在预处理后保持固定；而向量 $v$ 表示推理过程中输入的激活值，正如引言部分所述。我们的目标是在保持精确计算的同时，加速实际推理流水线中的这一乘法。

**算法概述**。冗余分段归约（RSR）算法 [2] 包含两个阶段：首先对矩阵 $M$ 执行一次离线预处理，随后在推理过程中反复执行快速的在线乘法阶段。在预处理期间，RSR 将矩阵划分为若干个包含 $k$ 行的水平块，并在每个块中识别相同的列分段，将具有相同位模式的列分组。预处理阶段会生成辅助数据结构，包括列置换、分组边界和散射索引，以便在推理过程中高效复用中间结果。在推理期间，给定输入向量 $v$，RSR 会在乘法前汇总对应相同列分段的向量元素，并且每个唯一分段只分配一次贡献。这样可以减少计算 $M \cdot v$ 所需的算术运算次数，同时保持乘法结果精确且不增加空间占用。

### 2.1 系统架构

RSR-core 是一个采用图 1 所示分层架构的优化型 RSR 执行引擎。其核心是提供在 CPU（C 内核）和 CUDA（GPU 内核）上实现 RSR 乘法的内核级实现。这些内核通过中间乘法器抽象对外提供，并集成到 Python 接口中，以支持端到端的预处理和推理工作流。

该架构使用户能够对量化模型进行一次预处理，并在多次推理调用中重复使用生成的产物。完整源代码已公开发布于此代码库。

### 2.2 内核实现

尽管 RSR 算法在概念上十分简单，但直接使用 Python 实现无法带来实际的加速，这是因为解释器开销、通用排序，以及在采集、聚合和散射阶段低效的内存访问模式。为此，RSR-core 针对二进制（1 位）和三元（1.58 位）矩阵，分别提供了以 C（CPU）和 CUDA（GPU）实现的原生内核。所有后端均采用一项关键设计：预处理时在离散模式空间（$2^k$ 个桶用于二进制，$4^k$ 个桶用于三元）上使用计数排序，使每个数据块的复杂度达到 $O$（$n$ + 个桶），而非基于比较的排序所需的 $O$（$n \log n$）。元数据统一压缩为紧凑类型——置换索引和分组边界以 16 位整数存储，散射目标则编码为固定大小的位掩码，而非可变长度索引数组——从而降低推理热路径中的带宽压力。

在 CPU 上，C 内核将收集阶段与聚合阶段融合为一次对输入向量的遍历，直接累加部分和，而无需生成中间数组。对于 RSR 固有的随机访问模式，二进制内核采用带有软件预取提示的标量展开加载，其性能优于硬件向量收集指令。对于三值矩阵，每组都需要散射一个正向贡献和一个负向贡献，因此内核以每组两个紧凑的位掩码取代可变长度的有符号散射列表，并直接遍历其中的置位比特，使每组的散射开销不受活跃输出行数量的影响。所有 CPU 内核都按行块进行并行化。

对于 CPU 上的端到端模型推理，RSR-core 进一步将激活量化与 RSR 矩阵向量乘法融合为一次原生调用，并将共享同一输入向量的同级线性层（例如 $W_q$、$W_k$、$W_v$）批处理为一次组合操作。这消除了重复的 Python 调度和冗余量化，并提供了更多可分配到各核心的行块——这一层优化在内核本身带来加速之外，也显著促成了观测到的实际运行时间加速。

在 CUDA 上，每个行块会分配给一个线程块，块内的线程束并行处理各组。每组的元数据都会打包到一个 64 位字中，其中编码了置换范围、长度和散射掩码，因此内核每组只需加载一个值即可立即开始工作。预处理期间会对每组内的置换索引进行排序，这不会影响计算出的总和，但能提升收集读取时的缓存局部性。每个线程束使用独立的共享内存部分缓冲区，以避免输出写入时发生争用；预处理还会丢弃零贡献组，消除无用计算。针对 $k$ 的编译期特化使编译器能够为每种受支持的块高度[3]完全展开散射循环。

[2]在原始论文中，考虑的是乘法 $v \cdot M$，而不是 $Mv$，因此分组的是行（而非列）。

[3]更多技术实现细节请参阅代码仓库。

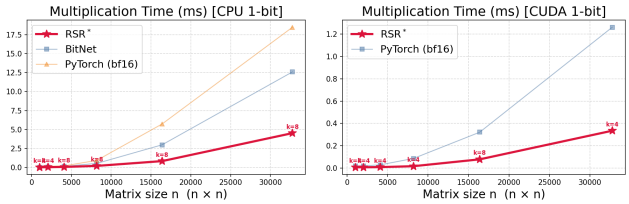



**图 2：CUDA 和 CPU 上的矩阵乘法基准测试。矩阵为二进制，向量为 float32。**

## 2.3 乘法器

除了内核实现之外，RSR-core 还提供了一组名为 Multiplier 的 Python 类。这些类封装了乘法器特有的逻辑，包括基于 RSR 的乘法以及 BitNet 等基准乘法器。每个乘法器都提供两项主要操作：预处理和乘法。

预处理阶段对给定的低比特矩阵进行处理，并生成可复用的产物，用于编码分段级结构，以便在推理时高效执行。随后，乘法阶段将这些产物与输入向量 $v$ 结合，计算精确的矩阵-向量乘积 $M \cdot v$。[4]

## 2.4 集成

RSR-core 提供了使用三值大语言模型执行推理的接口，并可直接集成 HuggingFace 模型库中的模型[5]。使用三元（BitLinear）层的模型（例如 BitNet [11] 及其变体）可与 RSR-core 配合使用，无需修改现有推理流水线[6]。

用户只需对从 HuggingFace 模型库下载的选定模型进行一次预处理。生成的预处理产物会存储在本地，其占用空间最多与模型本身相同，并可在后续推理执行中重复使用。在推理过程中，用户可以通过标准提示工作流与模型交互，并使用 RSR-core 提供的加速矩阵-向量乘法获取响应（见演示部分）。

RSR-core 还包含一个轻量级用户界面，可简化模型选择、预处理管理、生成产物的存储监控以及与模型进行交互式提示。

通过名为 RSRLinear 的自定义 PyTorch 模块，可将其集成到基于 PyTorch 的推理流水线中。该模块使用支持 RSR 的实现替换三值线性层。任何使用 *PyTorch* 进行神经网络推理的系统都可以通过该模块以最小改动接入 RSR-core。

## 2.5 基准测试结果

为评估 RSR-core 的性能，我们将其乘法耗时与 PyTorch 提供的基线实现进行比较；这些实现已在 CPU 和

[4]请注意，只有基于 RSR 的乘法器需要进行预处理。[5] https://huggingface.co/models [6]例如，microsoft/bitnet-b1.58-xxx 模型。

**表 1：相较于 HuggingFace PyTorch bfloat16，三元（1.58 位）大语言模型推理加速**

| 模型 | HF（Tok/s） | RSR（Tok/s） | 加速比 |
|---|---|---|---|
| *CPU* | | | |
| Falcon3-10B-1.58b | 0.2 | **11.3** | **62.0x** |
| Llama3-8B-1.58b | 0.2 | **13.4** | **53.8x** |
| BitNet-2B-4T-bf16 | 2.1 | **28.8** | **13.9x** |
| BitNet-2B-4T | 14.2 | **29.3** | **2.1x** |
| *CUDA* | | | |
| Falcon3-10B-1.58b | 25.2 | **47.4** | **1.9x** |
| Llama3-8B-1.58b | 31.9 | **59.3** | **1.9x** |
| BitNet-2B-4T-bf16 | 33.1 | **57.4** | **1.7x** |
| BitNet-2B-4T | 41.6 | **57.1** | **1.4x** |

CUDA 平台上采用经过硬件优化的内核。我们还将其与 BitNet [11]进行比较，将其作为低比特矩阵乘法的另一项基线。结果（图 2）显示，RSR-core 在 CPU 和 CUDA 上的矩阵-向量乘法均实现了显著加速。

我们进一步评估了解码（令牌生成）阶段的大语言模型推理端到端性能，并将其与 CPU 和 CUDA 平台上的 HuggingFace PyTorch 推理进行比较。结果（表 1）表明，使用 RSR-core 可显著提高令牌生成吞吐量。更多基准测试结果请参阅项目代码库。

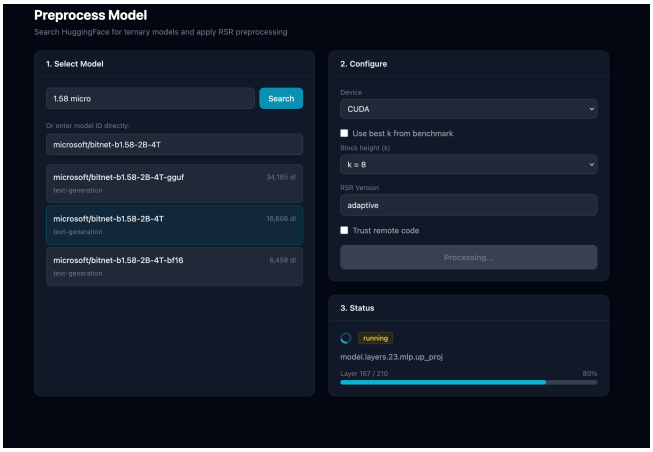
## 3 演示场景

演示过程中，参与者通过集成的基于 Web 的界面与 RSR-core 交互，该界面支持模型预处理、加速推理和内核级性能探索工作流程。

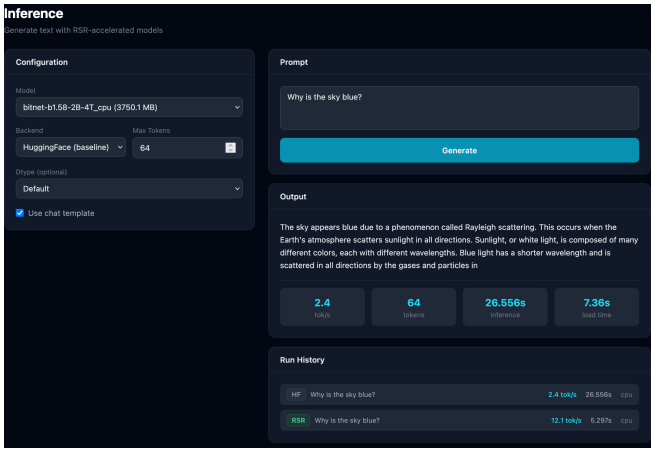
**场景 1：模型接入。**参与者在 HuggingFace Hub 上搜索三值模型，选中模型后配置 RSR 预处理参数，包括块高度 $k$、乘数版本和目标设备。启动预处理后，界面会实时显示各层的处理进度。处理完成后，模型会显示在模型列表中，并附带大小、层数和所选 $k$等元数据。RSR 预处理只需执行一次；该界面让非专业人士也能轻松使用。图 3a 展示了预处理界面。

**场景 2：并排推理。**参与者选择预处理后的模型，输入提示词，并分别使用 RSR 后端和 HuggingFace bfloat16 基线进行文本生成。界面会并排显示生成文本和性能指标——每秒生成令牌数、生成延迟及模型加载时间——并计算端到端加速比。该场景展示了 RSR 在实际大语言模型推理中的加速效果。图 3b 展示了推理对比界面。

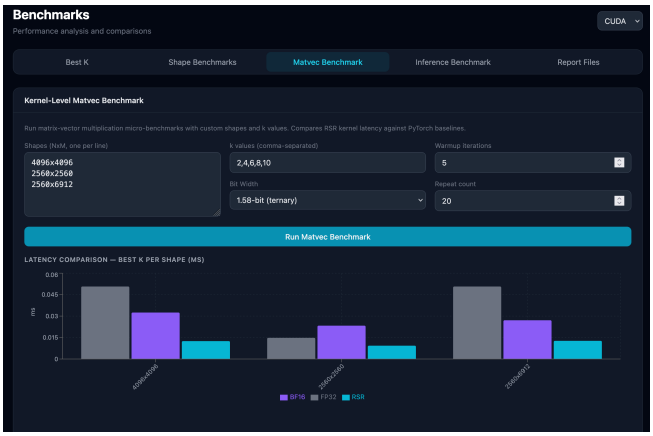
**场景 3：内核级探索。**参与者进入“矩阵向量乘法基准测试”选项卡，输入自定义矩阵形状（例如，与特定模型的权重维度相匹配），选择一系列 $k$ 值和位宽（1 位或 1.58 位），并启动内核级微基准测试。界面会生成柱状图，按每种形状的最佳 $k$ 值比较 RSR 与 FP32 和 BF16 基线，并生成折线图，展示 RSR 延迟如何随 $k$ 值变化。参与者



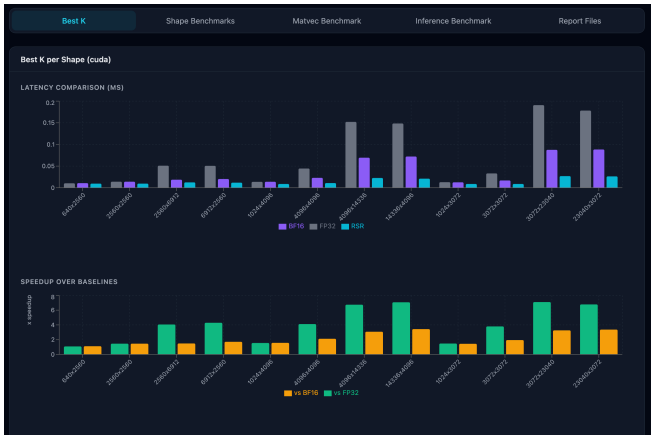

**(a) 模型预处理。**




**(b) 模型推理。**




**(c) 内核级矩阵向量乘法基准测试。**




**(d) 跨配置比较。**

**图 3：RSR-core 网页界面演示场景。**

可以在演示硬件上探索 RSR 的参数敏感性，并验证冗余结构会随形状变化。图 3c 展示了内核级基准测试界面。

**场景 4：跨配置比较。**参与者使用设备选择器在 CPU 和 CUDA 之间切换，并比较不同位宽（1 位与 1.58 位）下预先计算的形状基准测试结果。Best K 选项卡展示了最佳块高度如何随设备和矩阵形状而变化，说明 RSR 的优势取决于硬件。该仪表板支持快速探索不同配置，无需重新运行实验。图 3d 展示了跨配置比较界面。

## 参考文献

[1] Tingyang Chen、Cong Fu、Kun Wang、Xiangyu Ke、Yunjun Gao、Wenchao Zhou、Yabo Ni 和 Anxiang Zeng。2025年。最大内积就是查询缩放最近邻。*arXiv* 预印本 *arXiv:2503.06882*（2025年）。[2] Mohsen Dehghankar、Mahdi Erfanian 和 Abolfazl Asudeh。2024年。一种用于加速二进制和三值神经网络推理的高效矩阵乘法算法。*arXiv* 预印本 *arXiv:2411.06360*（2024年）。[3] Matthijs Douze。2025年。机器学习与高维向量搜索。*arXiv* 预印本 *arXiv:2502.16931*（2025年）。

[4] Matthijs Douze、Alexandr Guzhva、Chengqi Deng、Jeff Johnson、Gergely Szilvasy、Pierre-Emmanuel Mazaré、Maria Lomeli、Lucas Hosseini 和 Hervé Jégou。2024年。Faiss 库。（ 2024 2401 08281 [ ] ）。arXiv：。cs.LG Inc.

[5] 2024 Hugging Face。。Hugging Face 模型中心和 Transformers 库。https://huggingface.co。[6] 等 Ma, Shuming。2024年。1 位大语言模型的时代：所有大语言模型都只有 1.58 位。*arXiv* 预印本 *arXiv:2402.17764*（2024年）。[7] Xupeng Miao、Gabriele Oliaro、Zhihao Zhang、Xinhao Cheng、Hongyi Jin、Tianqi Chen 和 Zhihao Jia。2025年。迈向高效生成式大语言模型服务：从算法到系统的综述。《计算机调查》58, 1（2025年），1–37。[8] 等 Paszke, Adam。2019年。PyTorch：一种命令式风格、高性能的深度学习库。神经信息处理系统进展32（2019年）。[9] Thomas Rölleke、Theodora Tsikrika 和 Gabriella Kazai。2006年。用于信息检索建模的通用矩阵框架。《信息处理与管理》42, 1（2006年），4–30。[10] Mohamed Trabelsi、Zhiyu Chen、Brian D Davison 和 Jeff Heflin。2021年。用于文档检索的神经排序模型。《信息检索期刊》24, 6（2021年），400–444。[11] 等 Wang, Hongyu。2023年。BitNet：为大语言模型扩展 1 位 Transformer。*arXiv* 预印本 *arXiv:2310.11453*（2023年）。[12] Zixuan Zhou、Xuefei Ning、Ke Hong、Tianyu Fu、Jiaming Xu、Shiyao Li、Yuming Lou、Luning Wang、Zhihang Yuan、Xiuhong Li 等。2024年。大语言模型高效推理综述。*arXiv* 预印本 *arXiv:2404.14294*（2024年）。