# GEM-KMEANS: MEMORY-EFFICIENT AND ACCURATE CLUSTERING ON MASSIVE SCALE WITH GPU OPTIMIZATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Memory-efficient scaling on clustering problems without sacrificing statistical accuracy is of central interest for large-scale data analysis and machine learning problems. Nonnegative low-rank (NLR) matrix factorization for $K$-means is a scalable clustering method, which connects to semidefinite relaxations with optimal average-case exact recovery guarantees. However, a direct GPU implementation of NLR requires multiple large factor-sized buffers and substantial data movements that are essentially memory-bound. In this paper, we introduce GEM-KMEANS, a spectrally normalized yet mathematically equivalent NLR formulation that fuses the gradient update, nonnegative projection, and sufficient statistics for normalization and iterate movement into a matrix-multiplication epilogue. Instead of retaining three massive factor-sized arrays, our IO-aware GPU implementation materializes only one single factor with small tile-reduction arrays as additional storage in the High Bandwidth Memory (HBM). We derive explicit memory costs and spectrally normalized smoothness bounds for optimizing the clustering objective function. Accurate clustering is demonstrated at massive scales on synthetic and real datasets, where performance gains of GEM-KMeans over existing GPU-accelerated Lloyd's algorithms involve data-dependent runtime tradeoffs.

## 1 INTRODUCTION

Clustering is a core technique in modern machine learning, underpinning a broad range of applications across data analysis, representation learning, and large-scale AI systems. For example, semantic deduplication groups related training examples before selecting data (Abbas et al., 2023); clustering groups sources with similar observed properties in massive astronomical catalogs (Hunt & Reffert, 2023); retrieval systems organize representations around centroids (Santhanam et al., 2022); and clustering allows selective cache access and sparse attention for large language model inference (Liu et al., 2025; Zhu et al., 2026). The $K$-means method is arguably one of the most widely used clustering methods, partially due to the fast Lloyd's algorithm iterating between nearest-centroid assignment and mean updates (MacQueen, 1967; Lloyd, 1982). Despite being scalable with highly efficient GPU implementations (Johnson et al., 2021; Yang et al., 2026), this heuristic approach can be sensitive to initialization and it only enjoys suitable theoretical guarantees in recovering cluster labels when carefully initialized (Zhuang et al., 2023; Qian et al., 2022; Lu & Zhou, 2016).

Since the exact solution of the $K$-means problem, as a combinatorial optimization problem, is computationally intractable in general (Dasgupta, 2008; Aloise et al., 2009), there are many other relaxed $K$-means formulations, including spectral clustering (von Luxburg, 2007; Ng et al., 2001), nonnegative matrix factorization (NMF) (He et al., 2011; Kuang et al., 2015; Wang & Zhang, 2013), (convex) semidefinite programming (SDP) (Peng & Wei, 2007; Mixon et al., 2017; Giraud & Verzelen, 2018), and nonconvex low-rank SDPs (Zhuang et al., 2024; Xu et al., 2026; Carson et al., 2017). Nevertheless, unlike Lloyd's algorithm, it is extremely challenging to adopt those $K$-means variants in massive-scale problems with hundreds of millions of data points, due to hardware (e.g., memory and data movement) or algorithmic (e.g., high computational complexity) constraints. This raises the following natural tradeoff question between statistical accuracy and computational efficiency:



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

*Can we develop a statistically accurate, or even optimal, $K$-means implementation that is hardware efficient and scalable to a massive number of data points?*

In this paper, we introduce the GEM-KMEANS (*GEneral Matrix multiplication $K$-Means*) method, a **GPU-native**, **highly scalable**, *and* **statistically exact** $K$-means approach based on the nonnegative low-rank (NLR) algorithm, originally introduced by (Zhuang et al., 2024). Unlike existing GPU-accelerated Lloyd's algorithm with min-assignment and centroid update kernels (Yang et al., 2026; Johnson et al., 2021; Faiss developers, 2026; NVIDIA, 2026c), our implementation is gradient-based that is naturally compatible with modern GPU hardware design and especially suitable for dense computes via Tensor Core kernels. With fused gradient kernels based on low-level NVIDIA libraries such as cuBLAS (NVIDIA, 2026a) and CUTLASS GEMM (NVIDIA, 2026b), we demonstrate that GEM-KMEANS, on a single H200 NVIDIA GPU, can effectively scale to hundreds of millions of data points, outperforming state-of-the-art GPU implementations of Lloyd's algorithm on comparable scales in terms of clustering accuracy.

### 1.1 OUR CONTRIBUTIONS

Our contributions center on reducing the memory footprint to execute the full-data NLR formulation while retaining its statistical structure and demonstrating clustering accuracy at massive scale.

1. **Memory-efficient GPU execution through fused updates and statistics.** We develop new customized cuBLAS/CUTLASS kernels tailored to fuse (dense) gradient updates, nonnegative projection, and accumulation of normalization and primal-movement statistics into a GEMM epilogue (Section 4). By collecting three sufficient statistics before overwriting the factor, GEM-KMEANS eliminates both the objective-gradient buffer and the previous-factor copy. This reduces factor-sized storage from three $n \times r$ arrays to one, saving $2nr$ FP32 entries ($8nr$ bytes), with small tile-reduction arrays as additional storage. We give explicit buffer accounting and retain $O(ndr)$ arithmetic per iteration without subsampling the data or weakening the NLR constraints.
2. **Normalized optimization with explicit smoothness and descent guarantees.** We combine spectral data normalization with a scaled constraint residual to control the gradient scale (Section 3). For a fixed dual variable, we prove the convergence of projected gradient descent for an appropriately fixed step size.
3. **Accuracy at hundred-million-sample scale.** On a single NVIDIA H200 GPU, we evaluate balanced Gaussian mixtures, CyTOF cell profiles, and 118 million Gaia DR3 sources. GEM-KMEANS attains near-perfect Adjusted Rand Index (ARI) in most synthetic runs (Figure 1). On 104184 CyTOF cells, its median Normalized Mutual Information (NMI) is approximately $0.82$, versus $0.79$ for the three baselines (Figure 3). On Gaia, its mean NMI is $0.315$, compared with $0.304$ for FLASH-KMEANS and $0.305$ for FAST-PYTORCH-KMEANS (Figure 4).

To our knowledge, GEM-KMEANS is the first $K$-means clustering solver that scales to **hundreds of millions** points with **guaranteed statistical accuracy** in the exact recovery regime under standard statistical sampling models such as Gaussian mixtures.

### 1.2 RELATED WORK

Existing GPU accelerations for $K$-means are exclusively based on Llyod's algorithm. Using the FlashAttention mechanism (Dao et al., 2022), FLASH-KMEANS reduces data movement for the Lloyd assignment and centroid update operations (Yang et al., 2026). FAISS and cuML provide additional practical GPU baselines (Johnson et al., 2021; Faiss developers, 2026; NVIDIA, 2026c). $K$-means++ and scalable $K$-means++ improve initialization, while mini-batch $K$-means trades full data updates for smaller batches (Arthur & Vassilvitskii, 2007; Bahmani et al., 2012; Sculley, 2010).

Up to a data spectral normalization, our GEM-KMEANS formulation is mathematically equivalent to the NLR method of Zhuang et al. (2024), whose unfused CPU implementation is limited to the order of a few tens of thousands due to memory constraints. The second-order Riemannian method of Xu et al. (2026) addresses the same NLR problem with a smooth log-barrier penalty and a different optimization geometry. Our GPU implementation shares the same general principle as FlashAttention and FLASH-KMEANS by consuming a matrix-product tile in its epilogue based on

the fusion mechanism of CUTLASS (NVIDIA, 2026b). However, our computation has an extra projection operation which still requires a global norm reduction.

## 2 $K$-MEANS FROM PARTITION TO NONNEGATIVE FACTORIZATION

Let $X \in \mathbb{R}^{n\times d}$ contain $d$-dimensional observations $x_1, \dots, x_n$ as rows, and let $K$ be the true number of nonempty clusters. For any given partition $\mathcal{G} = (G_1, \dots, G_K)$ of the data index $[n] \coloneqq \{1, \dots, n\}$, the total intra-cluster sum of squares is

$$J(\mathcal{G}) = \sum_{k=1}^{K} \sum_{i\in G_k} \|x_i - \bar{x}_k\|_2^2 = \|X\|_F^2 - \langle XX^\top, Z_{\mathcal{G}}\rangle, \qquad Z_{\mathcal{G}} = HH^\top, \tag{1}$$

where $\bar{x}_k$ is the $k$-th cluster mean and $H_{n\times K}$ is the associated assignment matrix with entries $H_{ik} = \mathbf{1}\{i \in G_k\}/\sqrt{|G_k|}$. Then, the partition-based formulation of the $K$-means clustering is to solve

$$\min_{\mathcal{G}} J(\mathcal{G}) \tag{2}$$

over all possible partitions $\mathcal{G}$ of $[n]$. Note that the $n\times n$ membership matrix $Z_{\mathcal{G}}$ is positive-semidefinite, entrywise nonnegative, has trace $K$, and satisfies $Z_{\mathcal{G}}\mathbf{1}_n = \mathbf{1}_n$. Relaxing the partition structure that $H$ is an $n \times K$ "binary" assignment matrix yields the Peng–Wei SDP (Peng & Wei, 2007):

$$\min_{Z\in\mathbb{R}^{n\times n}} \{\langle A, Z\rangle : Z \succeq 0,\ Z \geq 0,\ \mathrm{Tr}(Z) = K,\ Z\mathbf{1}_n = \mathbf{1}_n\}, \tag{3}$$

where $A = -XX^\top$ and $Z \geq 0$ denotes entrywise nonnegativity of the matrix $Z$.

In general, the SDP relaxation (3) to a continuous optimization problem may lose the integrality nature of cluster labels (Fei & Chen, 2022). Nevertheless, a well-known information-theoretical optimality result for the Peng–Wei SDP states that the unique SDP solution (3) achieves the *exact recovery* of cluster labels, as soon as this becomes possible under the standard Gaussian mixture model (GMM) (Chen & Yang, 2021). Specifically, consider the standard GMM $x_i = \mu_{z_i^*} + \varepsilon_i$ with i.i.d. sampling noises $\varepsilon_i \sim N(0, \sigma^2 I_d)$, equal cluster sizes, and the minimal centroid separation $\Delta = \min_{k\neq\ell\in[K]} \|\mu_k - \mu_\ell\|_2$. Here, $z_i^* = k$ if $i \in G_k^*$ means that the data point $x_i$ belongs to the true $k$-th cluster $G_k^*$. Then, there is a phase transition on the *centroid-to-noise ratio* (CNR), with the sharp threshold given by

$$\overline{\mathrm{CNR}}^2 \coloneqq 4\left(1 + \sqrt{1 + \frac{Kd}{n\log n}}\right)\log n, \tag{4}$$

in the sense that: for any $\alpha > 0$ and $K = O(\log n/\log\log n)$, (i) when the minimal centroid-separation $\Delta \geq (1+\alpha)\sigma\cdot\overline{\mathrm{CNR}}$, the SDP solution of (3) has a block diagonal structure (after data rearrangement), which perfectly recovers the true cluster structure $\mathcal{G}^* = (G_1^*, \dots, G_K^*)$ with high probability; (ii) when $\Delta \leq (1-\alpha)\sigma\cdot\overline{\mathrm{CNR}}$, then the maximum likelihood training, and therefore any other method regardless of computational cost, fails to exactly recover the true cluster structure.

Despite the appealing statistical optimality and robust empirical performance, SDP is often criticized for having poor scalability. Even for a small dataset with sample size $n = 1000$, the SDP requires $n^2 = 10^6$ optimization variables that are at the very boundary of state-of-the-art general-purpose SDP solvers such as SDPNAL+ (Yang et al., 2015). In the clustering problem when the SDP solution is expected to be low-rank, nonnegative low-rank (NLR) matrix factorization is a computationally thrifty nonconvex approach by reparametrizing $Z = UU^T$, where $U \geq 0$ is an $n \times r$ nonnegative matrix factor with a rank parameter $r \geq K$ (Burer & Monteiro, 2003). This substantially reduces the number of variables from $n^2$ to $nr$ at the cost of forfeiting the SDP convexity. Specifically, for the $K$-means SDP (3), Zhuang et al. (2024) proposed to solve the following nonconvex NLR problem:

$$\min_{U\in\Omega}\big\{\langle A, UU^\top\rangle : g(U) = \mathbf{0}_n\big\}, \tag{5}$$

where $\Omega = \{U \in \mathbb{R}^{n\times r} : U \geq 0, \|U\|_F^2 = K\}$ and $g(U) = UU^T\mathbf{1}_n - \mathbf{1}_n$. Note that problem (5) is a tighter NMF-style formulation than the SDP (3) because it imposes the entrywise nonnegative constraint $U \geq 0$ (rather than $UU^T \geq 0$) that is easier to enforce (Kulis et al., 2007). Tailoring

the standard primal-dual algorithm (cf. Algorithm 3 in Appendix) via the augmented Lagrangian method (Nocedal & Wright, 2006) to minimize

$$\min_{U\in\Omega}\Big\{f_y(U) \coloneqq \langle A, UU^\top\rangle + y^\top g(U) + \frac{\beta}{2}\|g(U)\|_2^2\Big\}, \tag{6}$$

it is shown in (Zhuang et al., 2024) that with a random initialization, the NLR algorithm enjoys the local linear convergence to a stationary point in the exact recovery regime $\Delta \geq (1+\alpha)\sigma \cdot \overline{\mathrm{CNR}}$.

## 3 Scaling Issues: Gradient Blowup and Massive Materialization

When scaling the NLR algorithm to massive problem sizes, there are serious optimization and storage obstacles for a direct implementation. First, in the objective function $f_y(U)$, gradient of the penalty term is given by $(\mathbf{1}_n\overline{y}^\top + \overline{y}\mathbf{1}_n^\top)U$ with $\overline{y} = y + \beta(UU^\top\mathbf{1}_n - \mathbf{1}_n)$, whose Frobenius norm increases with $n$. To avoid gradient blowup, we propose to center and spectrally normalize the data as $X_c = X - \mathbf{1}_n\bar{x}^\top$, $a = \|X_c\|_2 > 0$, $\hat{X} = X_c/a$, and $\hat{A} = -\hat{X}\hat{X}^\top$. Accordingly, we also normalize the constraint residual and define

$$\hat{f}_y(U) = \langle \hat{A}, UU^\top\rangle + y^\top h(U) + \frac{\beta}{2}\|h(U)\|_2^2, \qquad h(U) = \frac{g(U)}{\sqrt{n}}. \tag{7}$$

Here $y$ is the multiplier for $h$ rather than $g$. Then the gradient with respect to $U$ is given by

$$\nabla_U \hat{f}_y(U) = 2\hat{A}U + \bar{y}_{\text{eff}}(U^\top\mathbf{1})^\top + \mathbf{1}(U^\top\bar{y}_{\text{eff}})^\top, \tag{8}$$

where the effective multiplier and the dual ascent update are

$$\bar{y}_{\text{eff}} = \frac{y}{\sqrt{n}} + \frac{\beta}{n}g(U), \qquad y^+ = y + \frac{\beta}{\sqrt{n}}g(U). \tag{9}$$

The implementation stores $y/\sqrt{n}$ in its dual buffer and uses $\beta/n$ as the penalty coefficient. Consequently, the buffer is updated by $(\beta/n)g(U)$, which is equivalent to the update of $y$ in (9). All formulas below retain $y$ as the multiplier for $h$. We remark that centering and positive rescaling preserve optimal clustering partitions, but they may change the algorithmic path of infeasible iterates.

**Proposition 1** (A global fixed-dual smoothness bound)**.** *For fixed $y$ and $\beta > 0$, the gradient of (7) is Lipschitz on $\{U : \|U\|_F \leq \sqrt{K}\}$, with the valid bound*

$$\mathsf{L} = 2 + 2\|y\|_2 + \beta(6K + 2). \tag{10}$$

**Corollary 1.** *If $U \in \Omega$, $U^+ \in \Pi_\Omega(U - \alpha\nabla\widehat{f}_y(U))$ is an exact Euclidean projection and $0 < \alpha < 1/\mathsf{L}$, then*

$$\widehat{f}_y(U^+) \leq \widehat{f}_y(U) - \frac{1-\alpha\mathsf{L}}{2\alpha}\|U^+ - U\|_F^2. \tag{11}$$

As a direct consequence by summing over (11) and using the boundedness below of $\hat{f}_y$ on $\Omega$, we have for fixed $y$ and a constant step size $0 < \alpha < 1/\mathsf{L}$, the projected gradient iterates satisfy

$$\sum_{t=0}^{\infty}\|U_{t+1} - U_t\|_F^2 < \infty, \qquad \|U_{t+1} - U_t\|_F \to 0.$$

Second, a direct implementation would store the objective-gradient product, compute the projected update, and then compare successive iterates in separate operations. Projection enforces nonnegativity and the prescribed Frobenius norm, while the augmented-Lagrangian terms address $g(U) = UU^\top\mathbf{1}_n - \mathbf{1}_n$. Each primal iteration takes a gradient step with $y$ fixed and projects the result onto $\Omega$:

$$U^+ \in \Pi_\Omega\big(U - \alpha\nabla_U\hat{f}_y(U)\big), \quad \text{where } \Pi_\Omega(W) = \sqrt{K}\,W/\|W\|_F. \tag{12}$$

The implementation then recomputes the constraint residual and takes the dual ascent step in (9) when relative primal movement is sufficiently small. In a nutshell, a direct IO-agnostic implementation to minimize (7) over $\Omega$ with spectral normalization requires materializing the whole gradient matrix $U_{n\times r}$, which is memory-bounded when the sample size $n$ is very large.

# 4 OUR METHOD: EFFICIENT GPU IMPLEMENTATION

Since our method GEM-KMEANS is a memory-efficient GPU implementation of the $K$-means NLR algorithm and they are mathematically equivalent between (6) and (7), we first list the key updating formula, where a naive CPU/GPU implementation requires materialization of the full gradient with memory-bounded matrix multiplications. Then, we propose a new IO-aware implementation with fused gradient kernels based on the cuBLAS and CUTLASS GEMM libraries, which combine the entrywise update with the matrix multiplication used to evaluate that product. A key component in our gradient kernels is to accumulate the statistics needed for online projection and data movement monitoring before overwriting the current factor. This eliminates both the full objective-gradient buffer and a second copy of the factor.

## 4.1 GEM-KMEANS UPDATING FORMULA

Define

$$p = U^\top \mathbf{1}_n, \qquad q = Up - \mathbf{1}_n = g(U), \qquad \bar{y} = \bar{y}_{\text{eff}} = \frac{y}{\sqrt{n}} + \frac{\beta}{n} q. \tag{13}$$

Using the gradient in (8) with $\Theta = \hat{X}^\top U$ and $s = U^\top \bar{y}$, the positive part of the tentative update is

$$W = \left[U + 2\alpha \hat{X}\Theta - \alpha\left(\bar{y}p^\top + \mathbf{1}_n s^\top\right)\right]_+. \tag{14}$$

Note that the Euclidean projection in (12) is given by $U^+ = \sqrt{K}W/\|W\|_F$ when $\|W\|_F > 0$, where the positive-part operation is entrywise and can be fused with the gradient computation. The normalization depends on all entries and follows a global reduction. Associating the objective-gradient product as $\hat{X}(\hat{X}^\top U)$ avoids materializing $\hat{X}\hat{X}^\top$. The two constraint-gradient terms are generated from vectors rather than stored as $n \times r$ matrices.

## 4.2 FUSED GRADIENT UPDATE AND STATISTICS

We first compute $\Theta = \hat{X}^\top U$, using cuBLAS. The second product, $\hat{X}\Theta$, is computed by a CUTLASS GEMM whose epilogue applies the entrywise update and accumulates the statistics needed for normalization and convergence criterion. The kernel configuration is described in Appendix C.

For each valid output entry $(i, j)$, let $a_{ij}$ denote the GEMM accumulator for $(\hat{X}\Theta)_{ij}$ and $w_{ij}$ denote the entry of the nonnegative update $W$ in (14) before normalization:

$$w_{ij} = \max\{U_{ij} + 2\alpha a_{ij} - \alpha \bar{y}_i p_j - \alpha s_j,\ 0\}. \tag{15}$$

The epilogue combines the GEMM accumulator with the old factor entry and the vectors $\bar{y}, p, s$, producing $w_{ij}$ without storing intermediate gradient matrices of size $n \times r$. Once the normalization multiplier $c$ is available, the squared movement follows from

$$\|cW - U_{\text{prev}}\|_F^2 = c^2\|W\|_F^2 + \|U_{\text{prev}}\|_F^2 - 2c\langle W, U_{\text{prev}}\rangle_F. \tag{16}$$

Before overwriting the old factor entries, the epilogue accumulates partial sums for

$$a = \|W\|_F^2, \qquad b = \|U_{\text{prev}}\|_F^2, \qquad h = \langle W, U_{\text{prev}}\rangle_F. \tag{17}$$

Here, $a$ is the squared Frobenius norm of the unnormalized update $W$, $b$ is the squared Frobenius norm of the previous factor, and $h$ is their Frobenius inner product. The epilogue accumulates these statistics in FP64 before overwriting the old factor entries with $W$. This allows us to estimate primal movement without retaining a second copy of $U$. Finally, the partial sums per tile are reduced to the global scalars $a, b, h$ and are used to normalize the factor using $U \leftarrow cW$, where $c = \text{FP32}(\sqrt{K/a})$. This scaling pass follows the global reduction because $c$ depends on all output tiles.

## 4.3 PRIMAL MOVEMENT AND STOPPING

Using the accumulated statistics, we estimate relative primal movement as

$$\hat{\delta} = \sqrt{\frac{\max\{c^2 a + b - 2ch, 0\}}{K}}. \tag{18}$$

**Algorithm 1** GEM iteration with fused update and statistics

**Input:** $\hat{X}, U \in \Omega, y = 0, \alpha, \beta > 0, T, T_{\min}, \epsilon_d, \epsilon_c$.

$p \leftarrow U^\top \mathbf{1}$; $q \leftarrow Up - \mathbf{1}$;
**for** $t = 0, \ldots, T-1$ **do**
  $\Theta \leftarrow \hat{X}^\top U$;
  $\bar{y} \leftarrow y/\sqrt{n} + (\beta/n)q$; $s \leftarrow U^\top \bar{y}$;
  Compute $W$ by (14) and tile partials of $a, b, h$ in one GEMM epilogue;
  Reduce partials and transfer $a, b, h$ to the host;
  **if** $\sqrt{a} \le 10^{-20}$ **then**
    Report normalization breakdown and **break**;
  $c \leftarrow \mathrm{float}(\sqrt{K/a})$;
  $U \leftarrow cW$;
  Compute $\hat{\delta}$ by (18);
  $p \leftarrow U^\top \mathbf{1}$; $q \leftarrow Up - \mathbf{1}$;
  **if** $\hat{\delta} < \epsilon_d$ **then**
    $y \leftarrow y + (\beta/\sqrt{n})q$;
  $\rho \leftarrow \|q\|_2/\sqrt{n}$;
  **if** $\hat{\delta} < \min\{\epsilon_d, \epsilon_c\}$ and $\rho < \epsilon_c$ and $t + 1 \ge T_{\min}$ **then**
    **break**;
**return** $(U, y)$.

The clamp removes negative values caused by floating-point cancellation. The estimate uses the rounded multiplier $c$, but does not account for entrywise rounding when $cW$ is stored.

We use the accumulated squared norms $a$ and $b$ rather than replacing $c^2 a + b$ with $2K$. Floating-point normalization does not enforce the target squared norm $K$ exactly, and this difference can affect the movement estimate near convergence.

After scaling, we recompute $p = (U^+)^\top \mathbf{1}_n$ and $q = U^+ p - \mathbf{1}_n$. When $\hat{\delta} < \epsilon_d$, we take the dual step $y \leftarrow y + (\beta/\sqrt{n})q$. We terminate when

$$\hat{\delta} < \min\{\epsilon_d, \epsilon_c\}, \qquad \frac{\|q\|_2}{\sqrt{n}} < \epsilon_c, \qquad t + 1 \ge T_{\min}.$$

The iteration limit is $T$. At each iteration, primal movement determines whether to update the dual variable, while both movement and constraint feasibility determine whether to terminate.

The dual ascent tolerance $\epsilon_d$ and convergence tolerance $\epsilon_c$ are configurable parameters and could vary based on the clustering problem.

### 4.4 Arithmetic, storage, and synchronization

The two matrix products require approximately $4ndr$ floating-point operations per primal iteration, counting a multiply and an add separately. The matrix-vector products, entrywise updates, and reductions add $O(nr)$ work. The total is therefore $4ndr + O(nr)$ per iteration. These counts describe useful arithmetic per iteration and exclude work on padded tile entries, preprocessing, and initialization. For output tiles of size $B_M \times B_N$, the number of tiles is

$$N_{\text{tile}} = \left\lceil \frac{n}{B_M} \right\rceil \left\lceil \frac{r}{B_N} \right\rceil. \tag{19}$$

The reduction workspace stores three FP64 partial sums per GEMM output tile, requiring $24N_{\text{tile}}$ bytes, where $N_{\text{tile}} = \lceil n/B_M \rceil \lceil r/B_N \rceil$. Here, $B_M$ and $B_N$ are the output tile dimensions.

Table 1 lists the device buffers used during iterations. Their total storage is

$$b_f(nd + nr + dr + 4n + 2r) + 3b_s(N_{\text{tile}} + 1), \tag{20}$$

where $b_f = 4$ bytes for FP32 entries and $b_s = 8$ bytes for FP64 statistics.

Since $W$ overwrites $U$, the iteration requires only one $n \times r$ factor buffer. Fusing the gradient update and statistics into the GEMM epilogue removes the need to store the objective-gradient matrix and a separate copy of the previous factor, saving two $n \times r$ buffers, or $8nr$ bytes in FP32.

Table 1: Device buffers used during the iteration, excluding library workspaces and preprocessing temporaries.

| Buffer | Type | Elements |
|---|---|---|
| Input $\hat{X}$ | FP32 | $nd$ |
| Factor $U$, overwritten by $W$ | FP32 | $nr$ |
| Product $\Theta$ | FP32 | $dr$ |
| $\mathbf{1}, q, y/\sqrt{n}, \bar{y}$ | FP32 | $4n$ |
| $p, s$ | FP32 | $2r$ |
| Three tile-statistic arrays | FP64 | $3N_{\text{tile}}$ |
| Three reduced statistics | FP64 | 3 |

Table 2: Experimental settings in Section 5.1 with the NLR factorization rank $r$.

| $n$ | $d$ | $K$ | $r$ | Seeds |
|---|---|---|---|---|
| $10^6$ | 50 | 20 | 30 | 20 |
| $10^6$ | 200 | 100 | 300 | 5 |
| $10^7$ | 50 | 20 | 30 | 20 |
| $10^7$ | 200 | 100 | 300 | 5 |
| $10^8$ | 50 | 20 | 30 | 5 |
| $2 \times 10^8$ | 50 | 20 | 60 | 5 |

The host waits for the reduced statistics before computing the normalization multiplier, and for the feasibility norm before checking termination.

**Relation to the FlashAttention mechanism.** FlashAttention (Dao et al., 2022) computes attention in tiles without storing the full attention matrix in global memory. Our GEM-KMEANS follows the same principle of consuming matrix-product intermediates on chip. The CUTLASS epilogue uses the GEMM accumulators directly to update the factor and collect statistics, avoiding a separate objective-gradient buffer. Its projection depends, however, on the global norm $\|W\|_F$. Moreover, FlashAttention updates its softmax normalization incrementally as tiles are processed. In GEM-KMEANS, the normalization multiplier also depends on the global Frobenius norm of $W$, so our implementation reduces the tile statistics before applying a separate scaling pass.

## 5 EXPERIMENTS

We compare GEM-KMEANS (GEM hereafter) with three GPU implementations of Lloyd's algorithm: FLASH-KMEANS (Yang et al., 2026), FAST-PYTORCH-KMEANS (Omer, 2020), and FASTKMEANS (Clavié & Warner, 2025). All baselines use one random initialization per seed and a stopping tolerance of $10^{-4}$. These tolerances are not directly equivalent because the implementations aggregate centroid movements differently; in particular, FAST-PYTORCH uses squared movements. FASTKMEANS is additionally capped at 2000 iterations. We evaluate clustering quality against available labels and total runtime on a synthetic GMM dataset and two real-world datasets: mass cytometry (CyTOF) cell profiles and Gaia stellar observations. We report ARI for balanced datasets and NMI otherwise. All experiments run on a single NVIDIA H200 GPU; GEM uses a CUTLASS SM80 Tensor Core kernel with the customized fused epilogue described in Section 4.

### 5.1 SIMULATION BENCHMARKS

We consider three synthetic data regimes: large $n$ with small $K$, large $n$ with large $K$, and moderate $n$ with large $K$. We investigate two questions: (i) does solving the NLR relaxation recover the planted partition more reliably than Lloyd-type heuristics, and (ii) what does this cost in computation time as $n$ grows to $\sim 10^8$? In each setup, the dataset is generated from a balanced $K$-component GMM in $\mathbb{R}^d$ with centroid separation $\Delta = \gamma\overline{\text{CNR}}$, $\gamma = 1.0$. The settings are listed in Table 2. For each seed, we generate a new dataset and use it for all methods. Figure 1 shows the comparison results. GEM attains ARI close to one (i.e., exact recovery) in nearly every run, whereas the $K$-means baselines generally show lower accuracy and greater variation across seeds. A few isolated GEM runs fall below its typical performance, so the result is not uniform across initializations. Figure 1 also illustrates how the runtime–accuracy trade-off changes with scale. FAST-PYTORCH-KMEANS is the fastest method in every displayed configuration. GEM-KMEANS incurs greater computational cost to optimize a nonconvex factorization of the SDP relaxation, but achieves higher clustering accuracy. At $n = 2 \times 10^8$, GPU memory becomes a limiting factor: FASTKMEANS runs out of memory, while FLASH-KMEANS requires CPU–GPU block-streaming, whose transfer overhead makes it slower than GEM-KMEANS.

**Recovery versus runtime.** We compare GEM with FLASH-KMEANS on GMMs with $K = 20$, $d = 50$, and different $(n, \gamma)$ combinations. FLASH-KMEANS is run on 50 independently seeded datasets; on

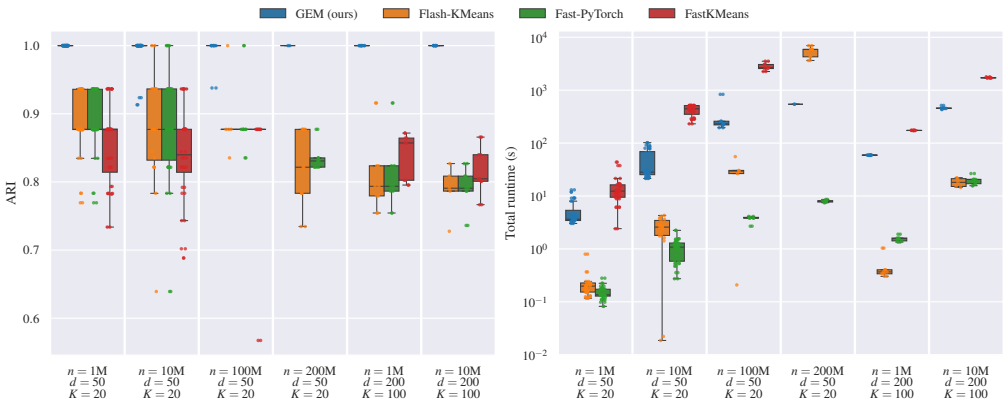


Figure 1: ARI and total runtime for five balanced GMM configurations. Each colored dot represents one run; boxes show the median and interquartile range across seeds. GEM achieves consistently high ARI, with occasional lower-scoring runs.

10 matched datasets, we evaluate GEM with iteration caps $T \in \{100, 300, 1000, 3000, 10000, 30000\}$, allowing runs to stop earlier upon convergence. The results shown in Figure 2. Across all three settings, FLASH-KMEANS rarely reaches the statistical floor: it succeeds in only 2–3 times and most runs instead finish with misclustering error between $0.07$ and $0.3$. Notably, doubling the separation to $\gamma = 2$ at $n = 10^6$ leaves this failure pattern largely intact, while GEM reaches the floor on all 10 matched datasets within a few seconds of iteration time. GEM also recovers on all evaluated seeds at $n = 10^7, \gamma = 1$, although there the required time rises to roughly $10^2$ seconds. These results favor GEM when reliable recovery matters, with a substantial runtime cost in the largest setting.

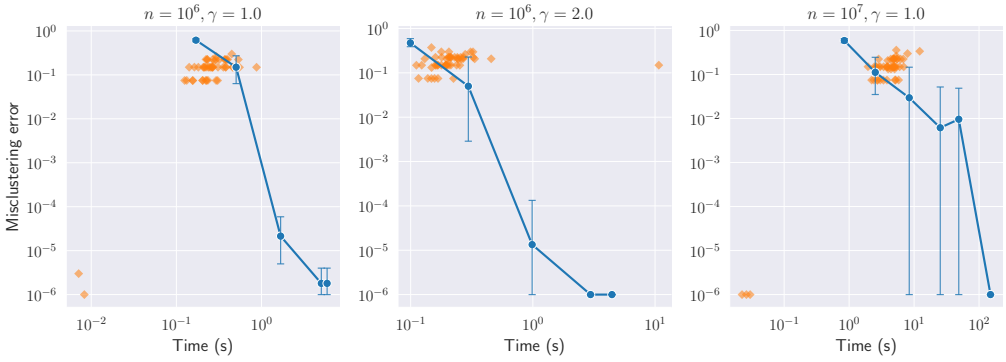


Figure 2: Misclustering error versus time for balanced Gaussian mixtures under three $(n, \gamma)$ settings. Diamonds show 50 individual FLASH-KMEANS runs. The line shows GEM's mean across 10 matched seeds at each iteration budget; bars span the range of error. Both axes are logarithmic. Times measure the GEM iteration loop and FLASH-KMEANS wall time, respectively.

## 5.2 REAL DATA APPLICATIONS

### 5.2.1 CYTOF

We evaluate the methods on a real-world mass cytometry (CyTOF) dataset (Levine et al., 2015; Weber, 2015) containing 104,184 labeled cell profiles measured across 32 protein markers. The labels identify 14 gated cell populations with markedly unequal sizes: the largest contains 26,366 cells, whereas the smallest contains 304. Figure 3 compares NMI and total runtime over 100 seeds. GEM achieves the highest median NMI, approximately 0.82, compared with approximately 0.79 for each baseline. FAST-PYTORCH-KMEANS has the shortest runtime due to its lower convergence criteria; GEM takes roughly 3 seconds per run.

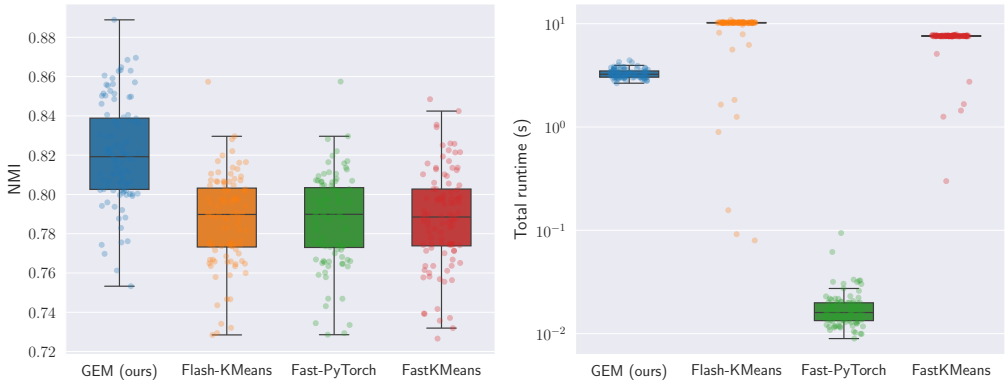


Figure 3: Clustering performance on the full CyTOF dataset across 100 random seeds. (Left) NMI against the gated cell labels; (Right) total runtime on a logarithmic scale. Boxes indicate the median and interquartile range, and dots represent individual runs. GEM achieves the highest median NMI.

### 5.2.2 GAIA DR3

Gaia Data Release 3 (DR3) (Gaia Collaboration et al., 2023) is the third data release of European Space Agency's Gaia space mission, and provides astrometry and photometry for about 1.8 billion sources. It also includes low-resolution BP/RP spectra for about 219 million sources, each represented by 110 Gauss–Hermite coefficients (De Angeli et al., 2023). We choose 118387161 sources with reliable astrometry and a confident spectral class from the ESP–HS pipeline (Creevey et al., 2023). Each band is normalized to a unit $L^2$ norm. The total data size is roughly 53 GB in single precision. The seven spectral classes (O, B, A, F, G, K, M) serve as reference labels for $K = 7$. Although the geometry of this dataset is challenging for Euclidean $K$-means clustering, its scale makes it a useful benchmark for computational scalability; our aim is to evaluate clustering algorithms at this scale rather than to develop a new method for stellar classification. Figure 4 compares clustering quality and total runtime over five random seeds. GEM achieves the highest mean and median NMI, with a mean of $0.315$, compared with $0.304$ for FLASH-KMEANS and $0.305$ for FAST-PYTORCH-KMEANS. These results demonstrate that GEM can improve agreement with the reference spectral classes even at the scale of 118 million sources. This improvement comes at a higher computational cost: GEM's median runtime is approximately 38 minutes, compared with 39 seconds for FLASH-KMEANS and 20 seconds for FAST-PYTORCH-KMEANS.




Figure 4: Clustering performance on 118 million Gaia DR3 sources across five random seeds. Left: NMI against the seven reference spectral classes. Right: total runtime on a logarithmic scale. Boxes show the interquartile range, horizontal lines indicate medians, and dots represent individual runs.

### AI USE STATEMENT

In this work, we used generative AI tools to edit portions of the manuscript for clarity, readability, grammar, and style; to suggest possible titles; and to provide suggestions for improving the presentation and structure of some proof arguments. All AI-assisted suggestions concerning mathematical arguments were independently checked, verified, and revised by the authors. The tools were not used to independently establish the main mathematical results or determine the scientific conclusions of the paper. The authors take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

## REFERENCES


Amro Abbas, Kushal Tirumala, Dániel Simig, Surya Ganguli, and Ari S Morcos. SemDeDup: Data-efficient learning at web-scale through semantic deduplication. *arXiv preprint arXiv:2303.09540*, 2023. URL https://arxiv.org/abs/2303.09540.

Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, 2009. doi: 10.1007/s10994-009-5103-0. URL https://link.springer.com/article/10.1007/s10994-009-5103-0.

David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1027–1035, 2007. URL https://theory.stanford.edu/~sergei/papers/kMeansPP-soda.pdf.

Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable K-Means++. *Proceedings of the VLDB Endowment*, 5(7):622–633, 2012. doi: 10.14778/2180912.2180915. URL https://arxiv.org/abs/1203.6402.

Samuel Burer and Renato D. C. Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming*, 95(2):329–357, 2003. doi: 10.1007/s10107-002-0352-8. URL https://link.springer.com/article/10.1007/s10107-002-0352-8.

Timothy Carson, Dustin G. Mixon, Soledad Villar, and Rachel Ward. Manifold optimization for $k$-means clustering. In *2017 International Conference on Sampling Theory and Applications (SampTA)*, pp. 73–77, 2017. doi: 10.1109/SAMPTA.2017.8024388. URL https://ieeexplore.ieee.org/document/8024388.

Xiaohui Chen and Yun Yang. Cutoff for Exact Recovery of Gaussian Mixture Models. *IEEE Transactions on Information Theory*, 67(6):4223–4238, June 2021. ISSN 1557-9654. doi: 10.1109/TIT.2021.3063155. URL https://doi.org/10.1109/TIT.2021.3063155.

Benjamin Clavié and Benjamin Warner. fastkmeans: Accelerated KMeans clustering in PyTorch and Triton. https://github.com/AnswerDotAI/fastkmeans/, 2025.

O. L. Creevey, R. Sordo, F. Pailler, Y. Frémat, U. Heiter, et al. Gaia data release 3: Astrophysical parameters inference system (Apsis). I. methods and content overview. *Astronomy & Astrophysics*, 674:A26, 2023. doi: 10.1051/0004-6361/202243688. URL https://doi.org/10.1051/0004-6361/202243688.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088. doi: 10.52202/068431-1189. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html.

Sanjoy Dasgupta. The hardness of k-means clustering. Technical Report CS2008-0916, University of California, San Diego, 2008. URL https://escholarship.org/uc/item/2qm3k10c.

F. De Angeli, M. Weiler, P. Montegriffo, D. W. Evans, M. Riello, et al. Gaia data release 3: Processing and validation of BP/RP low-resolution spectral data. *Astronomy & Astrophysics*, 674:A2, 2023. doi: 10.1051/0004-6361/202243680. URL https://doi.org/10.1051/0004-6361/202243680.

Faiss developers. GPU k-means example, 2026. URL https://github.com/facebookresearch/faiss/wiki/GPU-k-means-example. Online software documentation; accessed September 25, 2026.

Yingjie Fei and Yudong Chen. Hidden Integrality and Semirandom Robustness of SDP Relaxation for Sub-Gaussian Mixture Model. *Mathematics of Operations Research*, 47(3):2464–2493, 2022. doi: 10.1287/moor.2021.1216. URL https://pubsonline.informs.org/doi/10.1287/moor.2021.1216.

Gaia Collaboration, A. Vallenari, A. G. A. Brown, T. Prusti, J. H. J. de Bruijne, et al. Gaia data release 3: Summary of the content and survey properties. *Astronomy & Astrophysics*, 674:A1, 2023. doi: 10.1051/0004-6361/202243940. URL https://doi.org/10.1051/0004-6361/202243940.

Christophe Giraud and Nicolas Verzelen. Partial recovery bounds for clustering with the relaxed $K$-means. *Mathematical Statistics and Learning*, 1(3/4):317–374, 2018. doi: 10.4171/MSL/8. URL https://ems.press/journals/msl/articles/16225.

N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011. doi: 10.1137/090771806.

Zhaoshui He, Shengli Xie, Rafal Zdunek, Guoxu Zhou, and Andrzej Cichocki. Symmetric nonnegative matrix factorization: Algorithms and applications to probabilistic clustering. *IEEE Transactions on Neural Networks*, 22(12):2117–2131, 2011. doi: 10.1109/TNN.2011.2172457. URL https://doi.org/10.1109/TNN.2011.2172457.

Emily L. Hunt and Sabine Reffert. Improving the open cluster census. II. an all-sky cluster catalogue with Gaia DR3. *Astronomy & Astrophysics*, 673:A114, 2023. doi: 10.1051/0004-6361/202346285. URL https://doi.org/10.1051/0004-6361/202346285.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2021. doi: 10.1109/TBDATA.2019.2921572. URL https://doi.org/10.1109/TBDATA.2019.2921572.

Da Kuang, Sangwoon Yun, and Haesun Park. SymNMF: nonnegative low-rank approximation of a similarity matrix for graph clustering. *Journal of Global Optimization*, 62(3):545–574, 2015. doi: 10.1007/s10898-014-0247-2. URL https://link.springer.com/article/10.1007/s10898-014-0247-2.

Brian Kulis, Arun C Surendran, and John C Platt. Fast low-rank semidefinite programming for embedding and clustering. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, volume 2 of *Proceedings of Machine Learning Research*, pp. 235–242. PMLR, 2007. URL https://proceedings.mlr.press/v2/kulis07a.html.

Jacob H. Levine, Erin F. Simonds, Sean C. Bendall, Kara L. Davis, El ad D. Amir, Michelle D. Tadmor, Oren Litvin, Harris G. Fienberg, Astraea Jager, Eli R. Zunder, Rachel Finck, Amanda L. Gedman, Ina Radtke, James R. Downing, Dana Pe'er, and Garry P. Nolan. Data-driven phenotypic dissection of AML reveals progenitor-like cells that correlate with prognosis. *Cell*, 162(1):184–197, 2015. ISSN 0092-8674. doi: 10.1016/j.cell.2015.05.047. URL https://www.sciencedirect.com/science/article/pii/S0092867415006376.

Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. ClusterKV: Manipulating LLM KV cache in semantic space for recallable compression. In *Proceedings of the 62nd Annual ACM/IEEE Design Automation Conference*, DAC '25, pp. 1–7. IEEE Press, 2025. ISBN 9798331503048. doi: 10.1109/DAC63849.2025.11132479. URL https://doi.org/10.1109/DAC63849.2025.11132479.

Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28 (2):129–137, 1982. doi: 10.1109/TIT.1982.1056489. URL https://doi.org/10.1109/TIT.1982.1056489.

Yu Lu and Harrison H. Zhou. Statistical and computational guarantees of Lloyd's algorithm and its variants. *arXiv preprint arXiv:1612.02099*, 2016. URL https://arxiv.org/abs/1612.02099.

J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pp. 281–297, Berkeley and Los Angeles, 1967. University of California Press. URL https://digicoll.lib.berkeley.edu/record/113015.

Dustin G Mixon, Soledad Villar, and Rachel Ward. Clustering subgaussian mixtures by semidefinite programming. *Information and Inference: A Journal of the IMA*, 6(4):389–415, 2017. doi: 10.1093/imaiai/iax001. URL https://academic.oup.com/imaiai/article/6/4/389/3074185.

Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, volume 14, pp. 849–856, 2001. URL https://proceedings.neurips.cc/paper/2001/hash/801272ee79cfde7fa5960571fee36b9b-Abstract.html.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, NY, 2nd edition, 2006. ISBN 978-0-387-40065-5. doi: 10.1007/978-0-387-40065-5. URL https://link.springer.com/book/10.1007/978-0-387-40065-5.

NVIDIA. cuBLAS library documentation. CUDA Toolkit documentation, 2026a. URL https://docs.nvidia.com/cuda/cublas/index.html. Accessed September 25, 2026.

NVIDIA. CUTLASS 3.0 GEMM API: Collective epilogue. CUTLASS documentation, 2026b. URL https://docs.nvidia.com/cutlass/4.5.2/media/docs/cpp/gemm_api_3x.html. Accessed September 25, 2026.

NVIDIA. cuML KMeans documentation, 2026c. URL https://docs.nvidia.com/cuml/latest/api/generated/cuml.cluster.KMeans/. Accessed September 25, 2026.

Sehban Omer. fast-pytorch-kmeans, 2020. URL https://github.com/DeMoriarty/fast_pytorch_kmeans. GitHub repository; accessed September 25, 2026.

Jiming Peng and Yu Wei. Approximating K-means-type clustering via semidefinite programming. *SIAM Journal on Optimization*, 18(1):186–205, 2007. doi: 10.1137/050641983. URL https://epubs.siam.org/doi/10.1137/050641983.

Wei Qian, Yuqian Zhang, and Yudong Chen. Structures of spurious local minima in $k$-means. *IEEE Transactions on Information Theory*, 68(1):395–422, 2022. doi: 10.1109/TIT.2021.3122465. URL https://doi.org/10.1109/TIT.2021.3122465.

Keshav Santhanam, Omar Khattab, Christopher Potts, and Matei Zaharia. PLAID: An efficient engine for late interaction retrieval. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, CIKM '22, pp. 1747–1756, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392365. doi: 10.1145/3511808.3557325. URL https://doi.org/10.1145/3511808.3557325.

D. Sculley. Web-scale K-Means clustering. In *Proceedings of the 19th International Conference on World Wide Web*, pp. 1177–1178, 2010. doi: 10.1145/1772690.1772862. URL https://doi.org/10.1145/1772690.1772862.

Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007. doi: 10.1007/s11222-007-9033-z. URL https://link.springer.com/article/10.1007/s11222-007-9033-z.

Yu-Xiong Wang and Yu-Jin Zhang. Nonnegative matrix factorization: A comprehensive review. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1336–1353, 2013. doi: 10.1109/TKDE.2012.51. URL https://doi.org/10.1109/TKDE.2012.51.

Lukas Weber. Clustering benchmark data: 32-dimensional data set from Levine et al. (2015), 2015. URL https://github.com/lmweber/benchmark-data-Levine-32-dim. GitHub repository; accessed September 25, 2026.

Peng Xu, Chun-Ying Hou, Xiaohui Chen, and Richard Y. Zhang. Scalable Second-order Riemannian Optimization for $K$-means Clustering. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/a7311af5d481dc9f6d3c993dbf8d2e21-Abstract-Conference.html.

Liuqin Yang, Defeng Sun, and Kim-Chuan Toh. SDPNAL+: a majorized semismooth Newton-CG augmented Lagrangian method for semidefinite programming with nonnegative constraints. *Mathematical Programming Computation*, 7(3):331–366, 2015. doi: 10.1007/s12532-015-0082-6. URL https://link.springer.com/article/10.1007/s12532-015-0082-6.

Shuo Yang, Haocheng Xi, Yilong Zhao, Muyang Li, Xiaoze Fan, Jintao Zhang, Han Cai, Yujun Lin, Xiuyu Li, Kurt Keutzer, et al. Flash-KMeans: Fast and Memory-Efficient Exact K-Means. *arXiv preprint arXiv:2603.09229*, 2026. URL https://arxiv.org/abs/2603.09229.

Kan Zhu, Tian Tang, Qinyu Xu, Zhan Jin, Yile Gu, Zhichen Zeng, Rohan Kadekodi, Liangyu Zhao, Ang Li, Arvind Krishnamurthy, and Baris Kasikci. Tactic: Adaptive Sparse Attention with Clustering and Distribution Fitting for Long-Context LLMs. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/33f94d79acf71051d6a27f4d8889e20e-Abstract-Conference.html.

Yubo Zhuang, Xiaohui Chen, and Yun Yang. Likelihood adjusted semidefinite programs for clustering heterogeneous data. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 43326–43346, 2023. URL https://proceedings.mlr.press/v202/zhuang23a.html.

Yubo Zhuang, Xiaohui Chen, Yun Yang, and Richard Zhang. Statistically optimal $K$-means clustering via nonnegative low-rank semidefinite programming. In *International Conference on Learning Representations (ICLR)*, volume 2024, pp. 18825–18857, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/5212d235f5d07a25170d4807d4d2d824-Abstract-Conference.html.

# A TECHINICAL RESULTS

## A.1 PROOF OF PROPOSITION 1

Fix $y$, and set

$$\hat{\mathbf{1}}_n \coloneqq \frac{\mathbf{1}_n}{\sqrt{n}}, \qquad \bar{y}(U) \coloneqq y + \beta h(U).$$

Then $\|\hat{\mathbf{1}}_n\|_2 = 1$ and $h(U) = UU^\top \hat{\mathbf{1}}_n - \hat{\mathbf{1}}_n$. Since $A$ is symmetric, differentiation gives

$$\nabla f(U) = 2AU + \hat{\mathbf{1}}_n \bar{y}(U)^\top U + \bar{y}(U)\hat{\mathbf{1}}_n^\top U.$$

Throughout, we use the mixed spectral/Frobenius inequality

$$\|AB\|_F \le \|A\|_2 \|B\|_F. \tag{21}$$

Take $U, V$ in the Frobenius ball $\{W : \|W\|_F \le \sqrt{K}\}$, and let $\Delta \coloneqq U - V$. Using

$$UU^\top - VV^\top = \Delta U^\top + V\Delta^\top$$

and (21), we obtain

$$\|UU^\top - VV^\top\|_F \le \big(\|U\|_2 + \|V\|_2\big)\|\Delta\|_F.$$

Consequently,

$$\|h(U) - h(V)\|_2 = \|(UU^\top - VV^\top)e\|_2 \le \big(\|U\|_2 + \|V\|_2\big)\|\Delta\|_F. \tag{22}$$

Moreover, throughout this ball,

$$\|U\|_2 \le \sqrt{K}, \qquad \|h(U)\|_2 \le \|UU^\top\|_2 + \|e\|_2 \le K + 1,$$

so

$$\|\bar{y}(U)\|_2 \le \|y\|_2 + \beta(K+1).$$

For the second term in the gradient, decompose

$$\bar{y}(U)^\top U - \bar{y}(V)^\top V = \bar{y}(U)^\top \Delta + [\bar{y}(U) - \bar{y}(V)]^\top V.$$

Since $\|\hat{\mathbf{1}}_n\|_2 = 1$, the mixed norm inequality and (22) give

$$\begin{aligned}\|\hat{\mathbf{1}}_n \bar{y}(U)^\top U - \hat{\mathbf{1}}_n \bar{y}(V)^\top V\|_F &\le \|\bar{y}(U)\|_2 \|\Delta\|_F + \beta \|h(U) - h(V)\|_2 \|V\|_2 \\ &\le \big(\|\bar{y}(U)\|_2 + \beta\big(\|U\|_2 + \|V\|_2\big)\|V\|_2\big)\,\|\Delta\|_F.\end{aligned}$$

For the third term, use the separate decomposition

$$\bar{y}(U)\hat{\mathbf{1}}_n^\top U - \bar{y}(V)\hat{\mathbf{1}}_n^\top V = \bar{y}(U)\hat{\mathbf{1}}_n^\top \Delta + [\bar{y}(U) - \bar{y}(V)]\hat{\mathbf{1}}_n^\top V.$$

The same estimates yield

$$\|\bar{y}(U)\hat{\mathbf{1}}_n^\top U - \bar{y}(V)\hat{\mathbf{1}}_n^\top V\|_F \le \big(\|\bar{y}(U)\|_2 + \beta\big(\|U\|_2 + \|V\|_2\big)\|V\|_2\big)\,\|\Delta\|_F.$$

Finally, using $\|A\|_2 = 1$ and $\|U\|_2, \|V\|_2 \le \sqrt{K}$, we have

$$\begin{aligned}\|\nabla f(U) - \nabla f(V)\|_F &\le (2 + 2\|\bar{y}(U)\|_2 + 4\beta K)\,\|\Delta\|_F \\ &\le (2 + 2\|y\|_2 + \beta(6K+2))\,\|\Delta\|_F.\end{aligned}$$

Thus the gradient is Lipschitz on the stated ball with constant $\mathsf{L}$ in (10).

## A.2 PROOF OF COROLLARY 1

Now let $U \in \Omega$ and $U^+ \in \Pi_\Omega(U - \alpha\nabla f(U))$ be an exact Euclidean projection, and set $D \coloneqq U^+ - U$. Since $U \in \Omega$, the projection property implies

$$\|U^+ - (U - \alpha\nabla f(U))\|_F^2 \le \|U - (U - \alpha\nabla f(U))\|_F^2.$$

Expanding both sides gives

$$\langle \nabla f(U), D\rangle \le -\frac{1}{2\alpha}\|D\|_F^2.$$

Both $U$ and $U^+$ lie in the ball of radius $\sqrt{K}$, and that ball is convex. The smoothness inequality therefore gives

$$f(U^+) \le f(U) + \langle \nabla f(U), D\rangle + \frac{\mathsf{L}}{2}\|D\|_F^2.$$

Combining the last two inequalities proves (11).

### A.3 Refining the Lipschitz constant

The global bound in Proposition 1 can be sharpened locally near full feasibility using the following result.

**Lemma 1.** *For any $U \in \Omega$ satisfying $h(U) = 0$, we have $\|U\|_2 = 1$.*

*Proof.* Let $M \coloneqq UU^\top$. Then $M$ is symmetric positive semidefinite and entrywise nonnegative, and $M\mathbf{1}_n = \mathbf{1}_n$. Hence $\|M\|_\infty = \|M\|_1 = 1$, so

$$\|M\|_2 \le \sqrt{\|M\|_1 \|M\|_\infty} = 1.$$

On the other hand, $M\mathbf{1}_n = \mathbf{1}_n$ implies that 1 is an eigenvalue of $M$. Therefore, $\|U\|_2^2 = \|M\|_2 = 1$. □

*Remark.* Lemma 1 applies to every fully feasible factor, regardless of optimality. In particular, if $U^*(U^*)^\top = Z^*$ and $Z^*$ is the cluster membership matrix, then, after reordering the samples, $Z^*$ is block-diagonal with blocks $\mathbf{1}\mathbf{1}^\top/n_k$. Each block is the orthogonal projection onto $\mathrm{span}(\mathbf{1})$ and has spectrum contained in $\{0, 1\}$.

To quantify the improvement near feasibility, note that the differential of $h$ satisfies

$$\mathrm{D}h(U)[D] = (DU^\top + UD^\top)\hat{\mathbf{1}}_n, \qquad \|\mathrm{D}h(U)[D]\|_2 \le 2\|U\|_2\|D\|_F.$$

Differentiating the gradient therefore gives the bound

$$\|\nabla^2 f(U)[D]\|_F \le \left(2 + 2\|y + \beta h(U)\|_2 + 4\beta\|U\|_2^2\right)\|D\|_F.$$

At any fully feasible $U$, Lemma 1 yields

$$\|\nabla^2 f(U)[D]\|_F \le \left(2 + 2\|y\|_2 + 4\beta\right)\|D\|_F. \tag{23}$$

Consequently, for fixed $y$ and $\beta$, and any $\varepsilon > 0$, the gradient is Lipschitz with constant $2+2\|y\|_2+4\beta+\varepsilon$ on a sufficiently small ball around any fully feasible point. This gives a local improvement over the global bound in Proposition 1.

The bound has no explicit dependence on $n$ or $K$ apart from any dependence through $\|y\|_2$ and $\beta$. We next control the optimal dual multiplier. Throughout this calculation, $A = -XX^\top$ denotes the objective matrix before normalization.

Following Zhuang et al. (2024, Assumption 2), consider the dual certificate constructed by Chen & Yang (2021):

$$(y^*)^\top = \left((y^*_{G_1^*})^\top \quad \dots \quad (y^*_{G_K^*})^\top\right),$$

where, writing $A_k \coloneqq A_{G_k^* G_k^*}$,

$$y^*_{G_k^*} = -\frac{2}{n_k} A_k \mathbf{1}_{n_k} - \frac{\lambda}{n_k}\mathbf{1}_{n_k} + \frac{1}{n_k^2}\mathbf{1}_{n_k}\left(\mathbf{1}_{n_k}^\top A_k \mathbf{1}_{n_k}\right).$$

Here $\lambda = d\sigma^2 + mC\Theta^2/4$ for some $C \in (0, 1)$, and $m = \min_{k\neq\ell} 2n_k n_\ell/(n_k + n_\ell)$ is the minimum pairwise harmonic mean of the cluster sizes. For $i \in G_k^*$,

$$[A_k\mathbf{1}_{n_k}]_i = -\sum_{j\in G_k^*} X_i^\top X_j = -n_k X_i^\top \bar{X}_k, \qquad \mathbf{1}_{n_k}^\top A_k \mathbf{1}_{n_k} = -n_k^2\|\bar{X}_k\|_2^2.$$

Thus the certificate formula is equivalently

$$y_i^* = 2X_i^\top \bar{X}_k - \|\bar{X}_k\|_2^2 - \frac{\lambda}{n_k}.$$

Under the normalization $s \coloneqq \|X\|_2^2 > 0$, the objective matrix and the trace multiplier are divided by $s$. Since the equality constraint is additionally divided by $\sqrt{n}$, the corresponding normalized multiplier is $\hat{y}^* \coloneqq (\sqrt{n}/s)y^*$. We bound this multiplier on the event that the constructed multipliers satisfy dual feasibility and complementary slackness at the true cluster matrix $Z^*$.

First, for $K \geq 2$, the certificate satisfies $0 \leq \lambda \leq s$. Let $B \in \mathbb{R}^{n\times n}$ be the symmetric, entrywise nonnegative dual multiplier associated with the constraint $Z \geq 0$. Complementary slackness gives $\langle B, Z^*\rangle = 0$, which implies $B_{G_k^* G_k^*} = 0$ for every $k$. Indeed, its dual slack matrix satisfies

$$Q \coloneqq \lambda I + \frac{1}{2}\big(\mathbf{1}_n (y^*)^\top + y^* \mathbf{1}_n^\top\big) + A - B \succeq 0, \qquad B \geq 0,$$

and equality of the primal and dual objectives gives

$$\mathbf{1}_n^\top y^* = -\langle A, Z^*\rangle - K\lambda.$$

We therefore obtain

$$0 \leq \hat{\mathbf{1}}_n^\top Q \hat{\mathbf{1}}_n = -\langle A, Z^* - \hat{\mathbf{1}}_n \hat{\mathbf{1}}_n^\top\rangle - (K-1)\lambda - \hat{\mathbf{1}}_n^\top B \hat{\mathbf{1}}_n.$$

Because $B$ is entrywise nonnegative and $Z^* - \hat{\mathbf{1}}_n \hat{\mathbf{1}}_n^\top$ is an orthogonal projection of rank $K-1$, while $0 \leq -A \leq sI$,

$$(K-1)\lambda \leq -\langle A, Z^* - \hat{\mathbf{1}}_n \hat{\mathbf{1}}_n^\top\rangle \leq (K-1)s.$$

Together with the chosen nonnegative value of $\lambda$, this proves the claim.

For each cluster, define

$$q_k \coloneqq \frac{\mathbf{1}_{n_k}}{\sqrt{n_k}}, \qquad a_k \coloneqq q_k^\top A_k q_k.$$

The block formula for the certificate becomes

$$y^*_{G_k^*} = -\frac{1}{\sqrt{n_k}}\big(2A_k + (\lambda - a_k)I\big)q_k.$$

Since $-sI \preceq A_k \preceq 0$, we have $-s \leq a_k \leq 0$. Together with $0 \leq \lambda \leq s$, this gives $0 \leq \lambda - a_k \leq 2s$, and hence

$$-2sI \preceq 2A_k + (\lambda - a_k)I \preceq 2sI.$$

Consequently,

$$\|2A_k + (\lambda - a_k)I\|_2 \leq 2s, \qquad \left\|y^*_{G_k^*}\right\|_2 \leq \frac{2s}{\sqrt{n_k}}.$$

Summing the squared bounds over the disjoint blocks yields

$$\|\hat{y}^*\|_2 = \frac{\sqrt{n}}{s}\|y^*\|_2 \leq 2\sqrt{n\sum_{k=1}^{K}\frac{1}{n_k}}.$$

Consequently, if the cluster sizes are balanced in the sense that $n_k \geq cn/K$ for a constant $c > 0$, then

$$\|\hat{y}^*\|_2 \leq \frac{2K}{\sqrt{c}}.$$

In particular, for equal cluster sizes, $\|\hat{y}^*\|_2 \leq 2K$. Thus the limiting local smoothness bound at the normalized dual certificate satisfies

$$2 + 2\|\hat{y}^*\|_2 + 4\beta \leq 2 + 4\sqrt{n\sum_{k=1}^{K}\frac{1}{n_k}} + 4\beta.$$

For balanced clusters this is $O(K + \beta)$, with an implied constant depending only on $c$ and no additional dependence on the sample size, dimension, noise level, or separation.

This conclusion concerns the optimal dual certificate. To extend it to the dual iterates, one additionally needs control of $\|y_t - \hat{y}^*\|_2$, since

$$\|y_t\|_2 \leq \|\hat{y}^*\|_2 + \|y_t - \hat{y}^*\|_2.$$

## B PSEUDOCODE

**Algorithm 2** GEM-KMEANS: detailed GPU implementation

**Input:** Row-major $\hat{X} \in \mathbb{R}^{n \times d}$, $U \in \Omega$ on the GPU;
$\alpha, \beta > 0, T, T_{\min}, \epsilon_d, \epsilon_c$.

$\tilde{y} \leftarrow 0$; $\gamma \leftarrow \beta/n$; ▷ *Stored dual $\tilde{y} = y/\sqrt{n}$;*
$B \leftarrow 256$; $G \leftarrow \lceil n/B \rceil$; ▷ *CUDA block and grid sizes*
$N_{\text{tile}} \leftarrow \lceil n/128 \rceil \lceil r/64 \rceil$; ▷ *CUTLASS output tiles*
▷ *Initial constraint residual* ◁
$\texttt{Sgemv}(\texttt{N}, r, n, 1, U, r, \mathbf{1}, 1, 0, p, 1)$; ▷ *cuBLAS: $p = U^\top \mathbf{1}$*
$\texttt{Sgemv}(\texttt{T}, r, n, 1, U, r, p, 1, 0, q_r, 1)$; ▷ *cuBLAS: $q_r = Up$*
$\texttt{Saxpy}(n, -1, \mathbf{1}, 1, q_r, 1)$; ▷ *cuBLAS: $q_r \leftarrow q_r - \mathbf{1}$*
**for** $t = 0, \ldots, T-1$ **do**
  ▷ *(a) Objective-gradient intermediate* ◁
  $\texttt{Sgemm}(\texttt{N}, \texttt{T}, r, d, n, 1, U, r, \hat{X}, d, 0, \Theta, r)$; ▷ *cuBLAS: $\Theta = \hat{X}^\top U$*
  ▷ *(b) Constraint-gradient vectors* ◁
  $\texttt{compute_ybar}\ \texttt{<<<}\ \texttt{G}, \texttt{B}\ \texttt{>>>}(\bar{y}, \tilde{y}, q_r, \gamma, n)$; ▷ *CUDA: $\bar{y} = \tilde{y} + \gamma q_r$*
  $\texttt{Sgemv}(\texttt{N}, r, n, 1, U, r, \bar{y}, 1, 0, s, 1)$; ▷ *cuBLAS: $s = U^\top \bar{y}$*
  ▷ *(c) CUTLASS fused update and statistics* ◁
  $\texttt{FusedGemm}(\hat{X}, \Theta; U, \bar{y}, p, s, \alpha, a_{\text{tile}}, b_{\text{tile}}, h_{\text{tile}})$; ▷ *CUTLASS; schematic arguments*
  ▷ *Epilogue: $u \leftarrow U_{ij}$, $w \leftarrow \max\{u + 2\alpha(\hat{X}\Theta)_{ij} - \alpha \bar{y}_i p_j - \alpha s_j, 0\}$* ◁
  ▷ *FP64 accumulation: $w^2$, $u^2$, $wu$ over valid entries* ◁
  ▷ *In-place store: $U_{ij} \leftarrow w$; shared-memory reduction to tile partials* ◁
  ▷ *(d) Global reduction and projection* ◁
  $\texttt{reduce_vplus_partials}\ \texttt{<<<}\ 1, \texttt{B}\ \texttt{>>>}(a_{\text{tile}}, b_{\text{tile}}, h_{\text{tile}}, a_{\text{dev}}, b_{\text{dev}}, h_{\text{dev}}, N_{\text{tile}})$;
  ▷ *Per-thread strided FP64 sums; shared-memory block reduction* ◁
  ▷ *$a = a_{dev} = \sum_\tau a_\tau$, $b = b_{dev} = \sum_\tau b_\tau$, $h = h_{dev} = \sum_\tau h_\tau$* ◁
  $\nu \leftarrow \sqrt{a}$; ▷ *Host: pre-scaling norm*
  **if** $(\nu < 10^{-20})$ **then**
    **break**; ▷ *Normalization breakdown*
  $c_f \leftarrow \mathrm{FP32}(\sqrt{K}/\nu)$; ▷ *Host: FP64 ratio*
  $\texttt{cublasSscal_64}(nr, c_f, U, 1)$; ▷ *cuBLAS: in-place normalization*
  ▷ *(e) Movement estimate on the host* ◁
  $c \leftarrow \mathrm{FP64}(c_f)$; ▷ *Rounded scaling multiplier*
  $\Delta \leftarrow \max\{c^2 a + b - 2ch, 0\}$; ▷ *Squared movement estimate*
  $\hat{\delta} \leftarrow \sqrt{\Delta}/\sqrt{K}$; ▷ *Relative movement*
  $\texttt{dual} \leftarrow (\Delta < \epsilon_d^2 K)$; ▷ *Dual-update gate*
  ▷ *(f) Updated constraint residual* ◁
  $\texttt{Sgemv}(\texttt{N}, r, n, 1, U, r, \mathbf{1}, 1, 0, p, 1)$; ▷ *cuBLAS: $p = U^\top \mathbf{1}$*
  $\texttt{Sgemv}(\texttt{T}, r, n, 1, U, r, p, 1, 0, q_r, 1)$; ▷ *cuBLAS: $q_r = Up$*
  $\texttt{Saxpy}(n, -1, \mathbf{1}, 1, q_r, 1)$; ▷ *cuBLAS: $q_r \leftarrow q_r - \mathbf{1}$*
  ▷ *(g) Conditional dual ascent and stopping* ◁
  **if** $\texttt{dual}$ **then**
    $\texttt{Saxpy}(n, \gamma, q_r, 1, \tilde{y}, 1)$; ▷ *cuBLAS: $\tilde{y} \leftarrow \tilde{y} + \gamma q_r$*
  $\texttt{Snrm2}(n, q_r, 1, \texttt{constraint_norm})$; ▷ *cuBLAS: host result $\|q_r\|_2$*
  $\rho \leftarrow \texttt{constraint_norm}/\sqrt{n}$; ▷ *Host: normalized feasibility*
  **if** $\texttt{dual}$ **and** $\hat{\delta} < \epsilon_c$ **and** $\rho < \epsilon_c$ **and** $t + 1 \geq T_{\min}$ **then**
    **break**.

Algorithm 3 shows the original CPU implementation of the NLR algorithm (Zhuang et al., 2024). We collect it here for reference.

**Algorithm 3** Hardware-agnostic NLR $K$-means algorithm

**Input:** Dissimilarity matrix $A = -X^\top X$;
cluster parameter $K > 0$, rank parameter $r \geq K$;
step size $\alpha > 0$, augmentation parameter $\beta > 0$
primal tolerance $\varepsilon_p$, tolerance $\varepsilon_d$.

**for** $t = 1, \ldots, T$ **do**
  $U_{t,0} \leftarrow U_t$;
  ▷ *Primal descent* ◁
  **for** $s = 1, \ldots, S$ **do**
    $\bar{y} \leftarrow y + \beta(U_{t,s-1}U_{t,s-1}^\top \mathbf{1} - \mathbf{1})$;
    $\nabla\mathcal{L} \leftarrow (2A + 2\ell I + \mathbf{1}_n\bar{y}^\top + \bar{y}\mathbf{1}^\top)U_{t,s-1}$;
    $U_{t,s} = \Pi(U_{t,s-1} - \alpha\nabla\mathcal{L})$, where $\Pi(V) = \sqrt{K}V_+/\|V_+\|_F$; ▷ *Projected gradient descent*
  $U_{t+1} \leftarrow U_{t,S}$;
  ▷ *Dual ascent* ◁
  **if** $\|U_{t+1} - U_t\|_F < \varepsilon_p$ **then**
    $\boldsymbol{s} \leftarrow U_{t+1}U_{t+1}^\top \mathbf{1} - \mathbf{1}$;
    ▷ *Stopping criterion* ◁
    **if** $\max\{\|U_{t+1} - U_t\|_F, \|\boldsymbol{s}\|_2\} < \varepsilon_d$ **then**
      **break**;
    $y \leftarrow y + \beta\boldsymbol{s}$.

# C GPU IMPLEMENTATION DETAILS

This appendix maps the GPU calls in Algorithm 2 to the normalized iteration. The CUTLASS epilogue, movement estimate, and storage conventions are described in the main text.

## C.1 STORAGE AND PARAMETER CONVENTIONS

The matrices $\hat{X}$, $U$, and $\Theta$ are stored in row-major order. cuBLAS interprets these buffers as column-major matrices with transposed dimensions:

| Row-major matrix | cuBLAS interpretation | Leading dimension |
|---|---|---|
| $\hat{X} \in \mathbb{R}^{n\times d}$ | $\hat{X}^\top \in \mathbb{R}^{d\times n}$ | $d$ |
| $U \in \mathbb{R}^{n\times r}$ | $U^\top \in \mathbb{R}^{r\times n}$ | $r$ |
| $\Theta \in \mathbb{R}^{d\times r}$ | $\Theta^\top \in \mathbb{R}^{r\times d}$ | $r$ |

(24)

No explicit transpose is needed. CUTLASS reads the same buffers directly in row major order.

The mathematical dual variable $y$ multiplies the normalized residual $g(U)/\sqrt{n}$. The device buffer named `y` stores $y/\sqrt{n}$, and the scalar `penalty_beta` equals $\beta/n$. Consequently, `compute_ybar` evaluates

$$\bar{y} = \texttt{y} + \texttt{penalty_beta}\, q_r = \frac{y}{\sqrt{n}} + \frac{\beta}{n} q_r.$$

The conditional dual `Saxpy` adds $(\beta/n)q_r$ to this buffer, implementing $y \leftarrow y + (\beta/\sqrt{n})q_r$ in mathematical notation.

**CUDA launch conventions.** In Algorithm 2, $B = 256$ is the thread-block size for the custom CUDA kernels and $G = \lceil n/B \rceil$ is the grid size for `compute_ybar`. The kernel `reduce_vplus_partials` uses one block of $B$ threads. CUTLASS configuration is described in Appendix C.3.

## C.2 CUBLAS ROUTINE REFERENCE

Table 3 summarizes the cuBLAS routines used in the iteration. The prefix `S` denotes single-precision real arithmetic. In this table, $\lambda$ and $\mu$ are generic BLAS coefficients, independent of the algorithmic parameters $\alpha$ and $\beta$.

Table 3: cuBLAS routines used in the iteration. Matrices and vectors reside on the device; the norm result is returned to host memory in this implementation.

| Routine | BLAS level | Operation |
|---|---|---|
| `Sgemm` | 3 | $C \leftarrow \lambda\,\mathrm{op}(A)\,\mathrm{op}(B) + \mu C$ |
| `Sgemv` | 2 | $v \leftarrow \lambda\,\mathrm{op}(A)u + \mu v$ |
| `Saxpy` | 1 | $v \leftarrow \lambda u + v$ |
| `Snrm2` | 1 | $\rho \leftarrow \|u\|_2$ |
| `Sscal` | 1 | $u \leftarrow \lambda u$ |

BLAS levels 1, 2, and 3 denote vector operations, matrix–vector operations, and matrix–matrix operations, respectively. The transpose flags specify

$$\mathrm{op}(A) = \begin{cases} A, & \texttt{N}\ (\texttt{CUBLAS_OP_N}), \\ A^\top, & \texttt{T}\ (\texttt{CUBLAS_OP_T}). \end{cases}$$

Setting $\mu = 0$ overwrites the output without using its previous contents.

In `Sgemm`, $(m, n, k)$ specify the product dimensions: $\mathrm{op}(A)$ is $m \times k$, $\mathrm{op}(B)$ is $k \times n$, and $C$ is $m \times n$. In `Sgemv`, $(m, n)$ specify the stored matrix $A$ before applying the transpose flag. Leading dimensions refer to the stored arrays.

**GPU calls and execution order.** Table 4 lists the computational calls in execution order. Before the loop, calls 7–9 are also used to initialize $p$ and $q_r$.

Table 4: GPU kernel call sites for one normalized GEM iteration. The first column gives kernel order, not pseudocode line numbers. For cuBLAS matrix calls, flags and dimensions are listed as passed to the library.

| | Routine | Configuration | Mathematical interpretation |
|---|---|---|---|
| 1 | `Sgemm` | N, T; $r, d, n$ | $\Theta^\top \leftarrow U^\top \hat{X}$; the output buffer stores $\Theta = \hat{X}^\top U$. |
| 2 | `compute_ybar` | Elementwise CUDA kernel | $\bar{y} \leftarrow y/\sqrt{n} + (\beta/n)q_r$. |
| 3 | `Sgemv` | N; $r, n$ | $s \leftarrow U^\top \bar{y}$. |
| 4 | CUTLASS FusedGEMM | Problem shape $(n, r, d)$; row-major operands | Compute $W$ from (14), accumulate tile partials of $a, b, h$, and overwrite $U$ with $W$. |
| 5 | `reduce_vplus_partials` | One block of $B = 256$ threads | Reduce the three arrays of tile partials to $a, b, h$ in (17). |
| 6 | `Sscal` | Length $nr$; multiplier $c$ | Scale the overwritten factor buffer: $U \leftarrow cW$. |
| 7 | `Sgemv` | N; $r, n$ | $p \leftarrow U^\top \mathbf{1}$. |
| 8 | `Sgemv` | T; $r, n$ | $q_r \leftarrow Up$. |
| 9 | `Saxpy` | Length $n$; multiplier $-1$ | $q_r \leftarrow q_r - \mathbf{1}$. |
| 10 | `Saxpy` (conditional) | Length $n$; multiplier $\beta/n$ | Update the stored dual buffer by $(\beta/n)q_r$, equivalent to $y \leftarrow y + (\beta/\sqrt{n})q_r$. |
| 11 | `Snrm2` | Length $n$ | Return $\|q_r\|_2$ for the feasibility diagnostic $\|q_r\|_2/\sqrt{n}$. |

### C.3 CUTLASS CONFIGURATION AND EPILOGUE CONSTRUCTION

The second matrix product, $\hat{X}\Theta$, uses a CUTLASS GEMM with row-major operands and problem shape $(n, r, d)$. The kernel is instantiated through `DefaultGemmWithVisitor` and exposed through `GemmUniversalAdapter`. It uses the `Sm80` architecture tag with `OpClassTensorOp`, threadblock tile $(128, 64, 16)$, warp tile $(64, 32, 16)$, instruction shape $(16, 8, 8)$, and three mainloop stages. Storage and accumulation are FP32; the selected Tensor Core multiplication uses TF32 internally. The operand alignment is one element, and the epilogue uses one stage.

**Epilogue visitor tree.** The epilogue is constructed recursively using `Sm80EVT<Node, Children...>`. Each parent receives fragments from its children and applies its operation. Figure 5 shows the resulting composition. Let $a_{ij} = (\hat{X}\Theta)_{ij}$ denote the GEMM accumulator.

**Loads and arithmetic.** The accumulator leaf supplies $a_{ij}$, and the auxiliary loads supply the old factor entry $U_{ij}$. Row broadcasts provide $p_j$ and $s_j$, while the column broadcast provides $\bar{y}_i$. Together with the scalar coefficients, these inputs evaluate the entrywise update in (15). All nodes operate within the same GEMM kernel.

**Statistics and storage.** The custom statistics node receives the unnormalized update $w$ and the old entry $u$ before $U_{ij}$ is overwritten. It accumulates $w^2$, $u^2$, and $wu$ in FP64 over valid output entries and passes $w$ to the store. The two `ULoad` leaves represent two uses of the old factor, not separate factor buffers. Thread local sums are reduced to one partial per statistic per output tile. Subseqeunt global reductions and scaling are performed in separate calls outside the tree.

## D ADDITIONAL COMPUTATION DETAILS

**Initialization.** The original NLR algorithm initializes $U_0 = \Pi_\Omega(Z)$ with i.i.d. $N(0, 1)$ entries in $Z$. To incorporate data structure, we instead use

$$V = XX^\top Z, \qquad V = QR, \qquad U_0 = \Pi_\Omega(Q).$$

The first two steps apply the randomized range finder of Halko et al. (2011) to the Gram matrix $XX^\top$. Multiplication by $XX^\top$ emphasizes its leading eigen-directions, while QR decomposition produces an orthonormal basis for the sampled range. Although the subsequent projection need not preserve

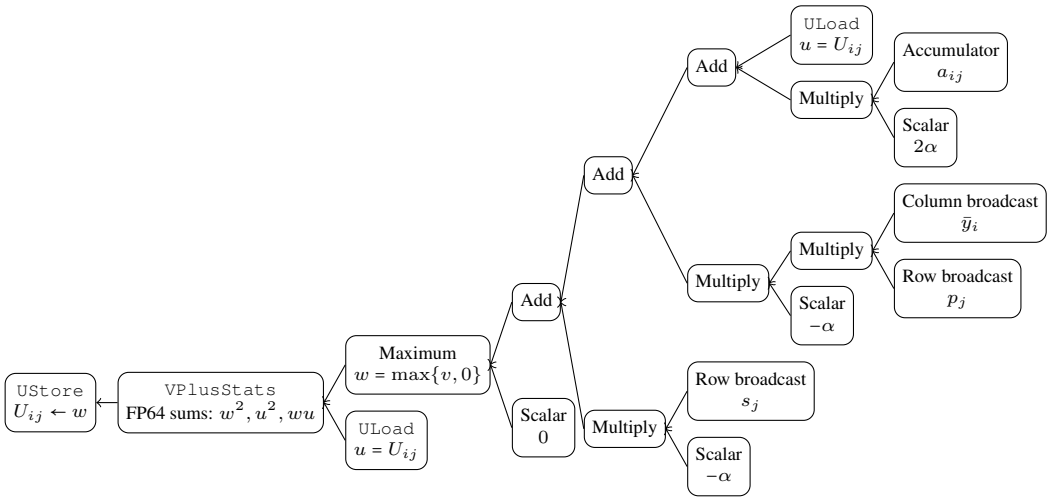


Figure 5: CUTLASS epilogue visitor tree. Arrows indicate fragment flow from children to parents, from the leaves on the right toward the store on the left. Arithmetic nodes form the nonnegative update, the custom visitor accumulates statistics, and the root stores the updated factor entry.

this range, the construction provides a data-informed initialization. Empirically, orthonormalizing before projection substantially reduces the initial augmented-Lagrangian value.

For efficient GPU computation, we implement the orthonormalization using Cholesky QR in the smaller feature space. Writing $M = X^\top Z$ and $C = X^\top X$, we have

$$V = XM, \qquad V^\top V = M^\top C M = R^\top R, \qquad Q = X(MR^{-1}),$$

where $R$ is the upper-triangular Cholesky factor and $MR^{-1}$ is computed by a triangular solve. This formulation replaces QR of an $n \times r$ matrix with an $r \times r$ Cholesky factorization and a small triangular solve, while the large operations remain GPU-efficient matrix multiplications. Moreover, $C$ can be reused from data normalization. That said, Cholesky QR requires a sufficiently well-conditioned, full-rank sample matrix; otherwise, a more stable orthogonalization is needed.

**Recover cluster labels from $U$ matrix.** Under exact recovery, the optimal clustering matrix $U^*(U^*)^\top$ has the block structure of the true partition, and the rows of any nonnegative factor $U^*$ are constant within each cluster, see Zhuang et al. (2024, Theorem 2) and Xu et al. (2026, Lemma 1). For an approximate solution $U$, we apply SVD and take its leading $K$ left singular vectors as a spectral embedding, and then apply $K$-means to their rows to obtain cluster labels. Although this SVD becomes costly for large $U$, its runtime is small relative to the main optimization loop in our experiments.

**Hardware information.** All experiments were conducted on NVIDIA H200 GPU nodes equipped with Intel Xeon Platinum 8558 processors and 32 GB of RAM. Additionally, some of the code were also tested on NVIDIA A100 and GeForce RTX 5090 GPUs.