A Batched Hartigan's k-Means Clustering

Edward Raff[1,2,3] **Michael Slawinski**[1]
[1]CrowdStrike
[2]University of Maryland, Baltimore County
[3]Syracuse University

Abstract

Hartigan's k-means algorithm has theoretical advantages over the standard Lloyd's variant, which has become synonymous with "k-means". Yet, Hartigan's algorithm is far slower in practice due to its inherently sequential calculations. We devise a Batched Hartigan's method that enables vectorized calculation by exploiting the triangle inequality to expose a set of parallel-safe updates and a smaller set of sequential updates. This makes Hartigan's method competitive with Lloyd's in execution speed for the first time.

1 Introduction

The k-means algorithm has been a workhorse of machine learning for over 50 years [21], with the simple procedure originally proposed by Lloyd still the dominant version of the algorithm fundamentally in use continuously since its proposal [19] and first presented in 1957. Yet, another old variant of k-means was proposed by Hartigan and Wong [14] has been consistently shown to be a theoretical Pareto improvement over the original Lloyd variant. As we will review soon, Hartigan and Wong's method (or Hartigan for short) converges to a strictly superior set of minima, in fewer epochs, with monotonically decreasing objective values in each iteration of the dataset. We posit that Hartigan's variant of k-means has been neglected due to computational cost. Despite having theoretically fewer calculations to perform, the original algorithm is entirely sequential and thus eliminates almost all opportunities for efficient vectorized operations afforded by Basic Linear Algebra Subprograms (BLAS), the standard compiled routines for vectorized linear algebra available in every scientific computing environment [7, 8, 15]. Our implementation relies only on BLAS, specifically GEneral Matrix Multiply (GEMM), the level-3 BLAS operation computing $C = \alpha AB + \beta C$, which achieves near-peak CPU throughput through cache blocking and single-instruction, multiple-data (SIMD) vectorization. We use GEMM for batch distance computation via the identity $\|\mathbf{x} - \mathbf{c}\|^2 = \|\mathbf{x}\|^2 - 2\mathbf{x} \cdot \mathbf{c} + \|\mathbf{c}\|^2$. Production BLAS uses the classical matrix-multiplication complexity: sub-cubic algorithms such as Strassen have large constants and weaker numerical stability and, to our knowledge, are not used in production BLAS. The advantage here is therefore constant-factor cache and SIMD efficiency, not a different asymptotic complexity.

By using the triangle inequality, we expose parallelism in the computation of Hartigan's algorithm that was not possible previously. Two key insights are required for our approach. First, in a modern system with large datasets and large k, the maximum change on a per-item basis to a given centroid is small, which enables safely determining most cluster assignments in bulk operations. Second, that the random order normally used for Hartigan's algorithm is not a necessary condition for any of its desirable properties, and so bulk operations can re-order points into a "parallel" set and a smaller "sequential" set, allowing us to vectorize the implementation and significantly improve runtime.

The rest of this article is organized as follows. First, we will review the related work to our own in section 2, covering the theory motivating the use of Hartigan's algorithm, along with the development of triangle inequality and faster k-means variants. Second, we will describe Hartigan-Wong's variation of k-means in detail in section 3, and its inherently sequential design. This motivates our improved

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

variant in section 4, where we use the triangle inequality to not just prune calculations, but expose batch calculations not previously available for efficient vectorization. Our results will show in section 5 that our Batched Hartigan algorithm maintains the same solution quality as the standard Hartigan algorithm, is usually $10\times$ faster than previous implementations of Hartigan's algorithm, and is competitive with modern fast implementations of Lloyd's k-means algorithm. Finally, we will conclude in section 6.

2 Related Work

Hartigan's algorithm was originally faster on older hardware [14], but is now misaligned with modern hardware efficiencies compared to variants of Lloyd's algorithm that updated the centroids once after viewing all data points. Hartigan's converges to a smaller and superior set as formally shown by [32], though the benefit is smaller when using good quality initial centers (e.g., via k-means++ [1, 29]). [30] further refined this by generalizing Hartigan's to use any Bregman divergence and showing how it is superior at avoiding degenerate cluster solutions. [22] However, a string of works have been steadily showing that Hartigan's method has theoretically justifiable superiority over Lloyds. [16] showed that Hartigan's is significantly superior in clustering in sparse and high dimensional problems. Others [3, 23] have proposed further improvements to Hartigan's method as well. In this article, we focused on the computational aspect of Hartigan's method and make it computationally viable for users to apply to their datasets, as we find all current implementations are unreasonably slow.

Using the triangle inequality to prune out redundant calculations has been popular since the 90s [27]. Elkan's [10] and Hammerly's [11] are the most popular today that we include as comparisons for runtime evaluation. Many works have further proposed different strategies for enabling pruning and balancing distance calculations vs efficient calculation [2, 5, 6, 12, 28], but an exhaustive comparison is beyond our scope. Our work shows how to use these methods not just to avoid redundant calculations, but a novel approach of enabling vectorized calculations via the triangle inequality rather than sacrificing bulk calculation to prune.

Other work accelerated Lloyd's brute-force k-means algorithm have been developed using GPUs to leverage their efficacy in large batch calculations and obtain $\approx 100\times$ speedups over the naive batch Lloyd on CPU [4, 17, 18, 20, 34]. However, very few have tried to use both a GPU and triangle-inequality based pruning, and results are far more modest with 3-8$\times$ speedup over the same CPU algorithm [31]. Better designing triangle-inequality pruning for effective GPU parallelization is still an open problem, and our results are not sufficient for GPU parallelism today — but it shows the potential path for future work in fast GPU clustering [24–26].

Table 1: Notation used throughout this article for the Batched Hartigan method.

Symbol	Domain	Meaning
B	$\mathbb{N}$	Batch size (points per GEMM)
$\mathcal{B}$	index set	Ordered set of B selected point indices; positions are relabeled $b = 1, \ldots, B$ within the batch
$\boldsymbol{\mu}_j^{(0)}$	$\mathbb{R}^D$	Centroid of cluster j at batch start
$\boldsymbol{\mu}_j^{(b)}$	$\mathbb{R}^D$	Centroid of cluster j after the b-th item is processed
$n_j^{(b)}$	$\mathbb{N}$	Size of cluster j after the b-th item
$\hat{d}_{bj}$	$\mathbb{R}_{\geq 0}$	Precomputed $\|\mathbf{x}_b - \boldsymbol{\mu}_j^{(0)}\|^2$ (from GEMM)
$\delta_j^{(b)}$	$\mathbb{R}_{\geq 0}$	Cumulative drift bound for cluster j after b items
$H_j^{\mathrm{lo}}, H_j^{\mathrm{hi}}$	$\mathbb{R}$	Lower/upper bounds on Hartigan cost (2) to cluster j
u_i, ℓ_i	$\mathbb{R}_{\geq 0}$	Per-point upper/lower distance bounds (cross-batch screening)
σ_j	$\mathbb{R}_{\geq 0}$	Cumulative centroid displacement since the last batch touching point i

3 Hartigan-Wong K-Means

Let $X \in \mathbb{R}^{N\times D}$ be the input data matrix with rows $\mathbf{x}_1, \ldots, \mathbf{x}_N$. We seek a partition of the index set $[N] = \{1, \ldots, N\}$ into k non-empty clusters $\mathcal{C} = \{C_1, \ldots, C_k\}$ that minimises the k-means

objective

$$\mathcal{L}(\mathcal{C}) = \sum_{j=1}^{k} \sum_{i \in C_j} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2 \qquad (1)$$

where $\boldsymbol{\mu}_j = \frac{1}{|C_j|} \sum_{i \in C_j} \mathbf{x}_i$, is the centroid of cluster C_j.

3.1 Hartigan Weighted Distance

The key insight of Hartigan's method is that, when considering whether to reassign $\mathbf{x}_i$ from its current cluster C_m to a candidate cluster C_j, one must account for the change in cluster sizes and the resulting shift of the centroids. This is captured by the *Hartigan weighted distance* [13]:

$$\Delta_H^2(\mathbf{x}_i, C_j) = \begin{cases} \frac{|C_j|}{|C_j|-1} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2, & \text{if } i \in C_j \\ \frac{|C_j|}{|C_j|+1} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2, & \text{if } i \notin C_j \end{cases} \qquad (2)$$

Algorithm 1 Hartigan's k-Means Clustering

Require: Data matrix $X \in \mathbb{R}^{N \times D}$ with rows $\mathbf{x}_1, \ldots, \mathbf{x}_N$; number of clusters k; initial partition $\mathcal{C} = \{C_1, \ldots, C_k\}$ (each $C_j \neq \varnothing$) and centroids $\{\boldsymbol{\mu}_j\}_{j=1}^k$.

repeat Repeat full data sweeps
  for $i = 1, \ldots, N$ **do**
    Let m be the cluster index s.t. $i \in C_m$
    Evaluate every destination
    for $j = 1, \ldots, k$ **do**
      $\Delta_j^2 \leftarrow \Delta_H^2(\mathbf{x}_i, C_j)$ ▷ Equation 2
    $j^\star \leftarrow \arg\min_j \Delta_j^2$
    Apply a beneficial move
    if $j^\star \neq m$ **and** $\Delta_{j^\star}^2 < \Delta_m^2$ **then** ▷ Move $\mathbf{x}_i$: $C_m \to C_{j^\star}$
      $C_m \leftarrow C_m \setminus \{i\}$; $C_{j^\star} \leftarrow C_{j^\star} \cup \{i\}$
      Update $\boldsymbol{\mu}_m$ and $\boldsymbol{\mu}_{j^\star}$
until No centroids change
return $\mathcal{C} = \{C_1, \ldots, C_k\}$, $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$

The weighting $|C_j|/(|C_j| - 1)$ inflates the squared distance to the current cluster (penalising self-influence of $\mathbf{x}_i$ on its own centroid), while $|C_j|/(|C_j| + 1)$ deflates the squared distance to a candidate cluster (anticipating the centroid shift upon absorption of $\mathbf{x}_i$). Algorithm 1 reassigns $\mathbf{x}_i$ to $j^\star = \arg\min_j \Delta_H^2(\mathbf{x}_i, C_j)$ whenever $\Delta_H^2(\mathbf{x}_i, C_{j^\star}) < \Delta_H^2(\mathbf{x}_i, C_m)$, the greedy move that maximally decreases the k-means objective (1) [33].

Despite its simplicity, three valuable properties have been proven using Hartigan's method: **(1) Monotone decrease:** Each accepted reassignment strictly decreases $\mathcal{L}(\mathcal{C})$, so the algorithm terminates in finitely many steps. **(2) No empty clusters:** When $N \geq k$ distinct points exist, the output partition has no empty clusters [33]. **(3) Strictly finer fixed points:** The set of local optima of Hartigan's method is a (possibly strict) subset of those of Lloyd's method [33]. In high-dimensional, high-noise regimes, Lloyd's algorithm can become trapped at essentially every balanced partition, whereas Hartigan's algorithm provably avoids these spurious fixed points [16].

4 Fast Batched Hartigan-Wong

This section introduces the batched method and its components. For $z \in \mathbb{R}$, write $[z]_+ := \max\{0, z\}$ for the positive part of z. In paired assignments, the $+$ branch supplies the first target and the $-$ branch supplies the second. Table 2 summarizes the state used by Algorithm 2 before the detailed definitions below.

Table 2: State used by Batched Hartigan.

Symbol	Meaning and detailed definition
$\hat{d}_{bj}, \delta_j^{(b)}, H_j^{\mathrm{lo}}, H_j^{\mathrm{hi}}$	Batch-start squared distance, drift bound, and Hartigan-cost bounds; §4.1
$\boldsymbol{\mu}_j^{(b)}, n_j^{(b)}$	Centroid and size of cluster j after batch position b; §4.1
$B, B_{\min}, B_{\max}$	Current and limiting batch sizes; §4.3
$\hat{p}, \lambda, c, p_{\mathcal{B}}, n_{\mathrm{moved}}$	Exponential moving average (EMA) state and batch-size inputs; §4.3
$\mathcal{B}, b$	Ordered selected point indices & their relabeled batch positions $b = 1, \ldots, B$; §4.4
$m(i)$	Cluster currently assigned to point i; §4.4
$u_i, \ell_i, \sigma_i^u, \Sigma_i^\ell, \sigma_j, \Sigma_{\max}$	Cross-batch distance bounds and lazy drift state; §4.4

Standard Hartigan (Algorithm 1) is inherently sequential: each reassignment updates two centroids, and the next point's distances are evaluated against the updated state. This prevents the use of level-3 BLAS routines for the distance computation. Empirically, for $k \geq 32$ and/or $D \geq 50$, the per-iteration cost of a Hartigan sweep exceeds that of a Lloyd E-step by a factor of 3–10×, even

though both perform $\Theta(NkD)$ distance evaluations, because Lloyd's distances are computed by a single highly-optimized matrix product (Table 5).

We propose *Batched Hartigan*, a framework that groups B consecutive points into a batch, precomputes all $B \times k$ distances in one GEMM, and then processes the batch—using the triangle inequality to certify that precomputed distances remain valid despite ongoing centroid drift. The subsections that follow develop each component in turn: safe sequential processing (§4.1), parallel batch execution (§4.2), adaptive batch sizing (§4.3), and cross-batch pruning of stable points (§4.4). Algorithm 2 provides the high-level structure.

Throughout this section, the notation in Table 1 distinguishes quantities fixed at batch start from quantities updated after each item.

To re-assign points to new centroids/clusters, we must bound the distance between a centroid and it's clusters potential locations $\mu^{(b)}$. Because Hartigan weights the update by cluster size, this value is meaningfully bounded. To demonstrate, we must start by defining the drift in the clusters:

Algorithm 2 Batched Hartigan k-Means

Require: Data matrix $X \in \mathbb{R}^{N\times D}$; number of clusters k; smoothing parameter $\lambda \in (0,1)$; safety factor $c \in (0,1)$; bounds $B_{\min}, B_{\max}$; initial partition $\mathcal{C}$ (each $C_j \neq \varnothing$).

Initialise centroids $\boldsymbol{\mu}_j$, sizes n_j for $j = 1,\dots,k$ ▷ §4.1
Precompute $\|\mathbf{x}_i\|^2$ for all $i \in [N]$ ▷ reused across all batches
$\hat{p} \leftarrow 1$ ▷ exponential moving average; §4.3
$B \leftarrow B_{\max}$ ▷ §4.3
Initialise per-point bounds $u_i \leftarrow \infty,\ \ell_i \leftarrow 0,\ \sigma_i^u \leftarrow 0,\ \Sigma_i^\ell \leftarrow 0$ for all i ▷ §4.4
$\sigma_j \leftarrow 0$ for $j = 1,\dots,k$
$\Sigma_{\max} \leftarrow 0$ ▷ §4.4
repeat ▷ **Process epochs and batches**
  for each batch $\mathcal{B}$ **from** BUILDBATCH(B) **do** ▷ Algorithm 6
    $\boldsymbol{\mu}_j^{(0)} \leftarrow \boldsymbol{\mu}_j$ and $n_j^{(0)} \leftarrow n_j$ for all j ▷ batch-start state
    $\hat{d}_{bj} \leftarrow \|\mathbf{x}_b\|^2 - 2\,\mathbf{x}_b\cdot\boldsymbol{\mu}_j^{(0)} + \|\boldsymbol{\mu}_j^{(0)}\|^2$ **for all** $b \in \mathcal{B},\ j \in [k]$
    $(\{\delta_j^{(|\mathcal{B}|)}\},\ n_{\text{moved}}) \leftarrow$ PARTIALPARALLELBATCH($\mathcal{B}$, $\{\hat{d}_{bj}\}$, $\{\boldsymbol{\mu}_j^{(0)}\}$, $\{n_j^{(0)}\}$) ▷ Alg. 4
    Refresh cross-batch bounds
    $\sigma_j \leftarrow \sigma_j + \delta_j^{(|\mathcal{B}|)}$ **for each** j ▷ refresh bounds (§4.4)
    $\Sigma_{\max} \leftarrow \Sigma_{\max} + \max_j \delta_j^{(|\mathcal{B}|)}$
    for batch position b corresponding to dataset index i **do**
      $u_i \leftarrow \sqrt{\hat{d}_{b,\,m(i)}} + \delta_{m(i)}^{(|\mathcal{B}|)}$
      $\sigma_i^u \leftarrow \sigma_{m(i)}$
      $\ell_i \leftarrow \left[\min_{j\neq m(i)} \sqrt{\hat{d}_{bj}} - \max_j \delta_j^{(|\mathcal{B}|)}\right]_+$
      $\Sigma_i^\ell \leftarrow \Sigma_{\max}$
    Select the next batch size
    $\boldsymbol{\mu}_j \leftarrow \boldsymbol{\mu}_j^{(|\mathcal{B}|)}$ and $n_j \leftarrow n_j^{(|\mathcal{B}|)}$ for all j
    $p_{\mathcal{B}} \leftarrow n_{\text{moved}}/|\mathcal{B}|$ ▷ §4.3
    $B \leftarrow$ DETERMINEBATCHSIZE($\{n_j^{(|\mathcal{B}|)}\}$, $p_{\mathcal{B}}$, $\hat{p}$, λ, c) ▷ Algorithm 5
until No centroids change
return $\mathcal{C}$, $\{\boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_k\}$

4.1 Safe Reassignment

Definition 4.1 (Cumulative drift bound). *Let a batch begin with centroids $\boldsymbol{\mu}_j^{(0)}$, and let $\boldsymbol{\mu}_j^{(1)}, \boldsymbol{\mu}_j^{(2)}, \dots$ denote the centroid of cluster j after each successive reassignment within the batch. The cumulative drift bound is $\delta_j^{(b)} = \sum_{s=1}^{b} \|\boldsymbol{\mu}_j^{(s)} - \boldsymbol{\mu}_j^{(s-1)}\|$. By the triangle inequality, $\|\boldsymbol{\mu}_j^{(b)} - \boldsymbol{\mu}_j^{(0)}\| \le \delta_j^{(b)}$. If cluster j is not involved in any reassignment during steps $1,\dots,b$, then $\delta_j^{(b)} = 0$.*

This then allows us to define upper and lower-bounds on the Hartigan cost Equation 2. For brevity, paired signs use subtraction for the lower bound and addition for the upper bound. Since both $\sqrt{\hat{d}_{bj}}$ and $\delta_j^{(b-1)}$ are nonnegative, the positive-part operator is redundant in the upper bound.

Lemma 4.1 (Distance bound propagation)**.** *Let $\hat{d}_{bj} = \|\mathbf{x}_b - \boldsymbol{\mu}_j^{(0)}\|^2$ be the precomputed squared distance. After $b-1$ items in the batch have been processed (with cumulative drift $\delta_j^{(b-1)}$ for cluster j, Definition 4.1), where m denotes the index of $\mathbf{x}_b$'s currently assigned cluster, the true squared distance and Hartigan cost satisfy: **(i) Squared-distance bounds:** By the triangle inequality applied to $\|\mathbf{x}_b - \boldsymbol{\mu}_j^{(b-1)}\|$ and Definition 4.1, is bounded above/below by $\left[\sqrt{\hat{d}_{bj}} \pm \delta_j^{(b-1)}\right]_+^2$. **(ii) Hartigan cost bounds:** Since the cluster sizes $n_j^{(b-1)}$ are tracked exactly, the Hartigan cost (2) is bounded above/below (defining H^{hi} and H^{lo}): $\Delta_H^2\big(\mathbf{x}_b, C_m^{(b-1)}\big)$ by $\frac{n_m^{(b-1)}}{n_m^{(b-1)}-1}\left[\sqrt{\hat{d}_{bm}} \pm \delta_m^{(b-1)}\right]_+^2$ for the current cluster*

$b \in C_m$. *The candidate cluster case for* $j \neq m$ *is instead bounded above/below by* $\frac{n_j^{(b-1)}}{n_j^{(b-1)}+1}\big[\sqrt{\hat{d}_{bj}} \pm \delta_j^{(b-1)}\big]_+^2$.

Proof sketch. Apply the triangle and reverse-triangle inequalities to the cumulative centroid displacement, then multiply by the positive Hartigan weights; see Appendix H.

With this in place, we can state the conditions under which we can safely perform Hartigan's clustering while using potentially stale centroid distances. Let m denote the current cluster of $\mathbf{x}_b$. Using the Hartigan cost bounds from Lemma 4.1(ii), write

$$H_m^{\mathrm{lo}},\ H_m^{\mathrm{hi}} \quad \text{for the current-cluster bounds,} \qquad H_j^{\mathrm{lo}},\ H_j^{\mathrm{hi}} \quad \text{for each candidate } j \neq m. \tag{3}$$

Definition 4.2 (Safe reassignment decision)**.** *Let* $j^\star = \arg\min_{j \neq m} H_j^{\mathrm{lo}}$. *The reassignment decision for* $\mathbf{x}_b$ *is* safe *if one of:* ***(S1) Safe move:*** $H_{j^\star}^{\mathrm{hi}} < H_m^{\mathrm{lo}}$ *and* $H_{j^\star}^{\mathrm{hi}} < H_j^{\mathrm{lo}}$ *for all* $j \neq j^\star, m$ *(the upper-bound cost of* $j^\star$ *is below the lower-bound cost of staying and all other candidates).* ***(S2) Safe stay:*** $H_j^{\mathrm{lo}} \geq H_m^{\mathrm{hi}}$ *for all* $j \neq m$ *(every candidate's lower-bound cost exceeds the upper-bound cost of staying). Otherwise the decision is* ambiguous *and requires exact recomputation.*

In the Appendix Theorem H.1 we prove that this is exactly replicates Hartigan's algorithm under the same iteration order. However, we will later show a more useful result for reordering the items in a batch, and so deffer it's details.

Algorithm 3 SEQUENTIALBATCH: per-point processing with bounds

Require: Ordered set $\mathcal{A} \subseteq \mathcal{B}$; precomputed distances $\{\hat{d}_{bj}\}$; state $\{\boldsymbol{\mu}_j^{(b_0)}, n_j^{(b_0)}, \delta_j^{(b_0)}\}$ after b_0 decisions.

Initialize within-batch state

Use $b_0 = 0$ and $\delta_j^{(0)} = 0$ for a full sequential batch
$n_{\mathrm{moved}} \leftarrow 0$
for $r = 1, \ldots, |\mathcal{A}|$ **do** ▷ **Process points in order**
  $b \leftarrow b_0 + r$ and let $\mathbf{x}_b$ be item r of $\mathcal{A}$
  Set $(\boldsymbol{\mu}_j^{(b)}, n_j^{(b)}, \delta_j^{(b)}) \leftarrow (\boldsymbol{\mu}_j^{(b-1)}, n_j^{(b-1)}, \delta_j^{(b-1)})$ for all j
  $m \leftarrow$ cluster of $\mathbf{x}_b$
  if $n_m^{(b-1)} \leq 1$ **then continue**

Bound every Hartigan cost

  $(H_m^{\mathrm{hi}}, H_m^{\mathrm{lo}}) \leftarrow \frac{n_m^{(b-1)}}{n_m^{(b-1)}-1}\big[\sqrt{\hat{d}_{bm}} \pm \delta_m^{(b-1)}\big]_+^2$ ▷ Lemma 4.1
  for $j = 1, \ldots, k,\ j \neq m$ **do**
    $(H_j^{\mathrm{hi}}, H_j^{\mathrm{lo}}) \leftarrow \frac{n_j^{(b-1)}}{n_j^{(b-1)}+1}\big[\sqrt{\hat{d}_{bj}} \pm \delta_j^{(b-1)}\big]_+^2$
  $j^\star \leftarrow \arg\min_{j \neq m} H_j^{\mathrm{lo}}$

Apply S1, S2, or exact fallback

  if $H_{j^\star}^{\mathrm{hi}} < H_m^{\mathrm{lo}}$ **and** $H_{j^\star}^{\mathrm{hi}} < H_j^{\mathrm{lo}}$ for all $j \neq j^\star, m$ **then** ▷ safe move (S1)
    $\delta_m^{(b)} \leftarrow \delta_m^{(b-1)} + \|\mathbf{x}_b - \boldsymbol{\mu}_m^{(b-1)}\| \,/\, (n_m^{(b-1)} - 1)$
    $\delta_{j^\star}^{(b)} \leftarrow \delta_{j^\star}^{(b-1)} + \|\mathbf{x}_b - \boldsymbol{\mu}_{j^\star}^{(b-1)}\| \,/\, (n_{j^\star}^{(b-1)} + 1)$
    Update $\boldsymbol{\mu}_m^{(b)},\ \boldsymbol{\mu}_{j^\star}^{(b)}$ incrementally
    $n_m^{(b)} \leftarrow n_m^{(b-1)} - 1$
    $n_{j^\star}^{(b)} \leftarrow n_{j^\star}^{(b-1)} + 1$
    $n_{\mathrm{moved}} \leftarrow n_{\mathrm{moved}} + 1$
  else if $H_j^{\mathrm{lo}} \geq H_m^{\mathrm{hi}}$ for all $j \neq m$ **then** ▷ safe stay (S2)
    continue
  else ▷ ambiguous: recompute exact distances
    Compute exact $\|\mathbf{x}_b - \boldsymbol{\mu}_j^{(b-1)}\|^2$ for all j
    Evaluate exact Hartigan costs
    Reassign if beneficial; update $\delta_m^{(b)}, \delta_{j^\star}^{(b)}, \boldsymbol{\mu}_m^{(b)}, \boldsymbol{\mu}_{j^\star}^{(b)}, n_m^{(b)}, n_{j^\star}^{(b)}$, and increment n_{moved} if a move occurs
return $\{\boldsymbol{\mu}_j^{(b_0+|\mathcal{A}|)}\}$, $\{n_j^{(b_0+|\mathcal{A}|)}\}$, $\{\delta_j^{(b_0+|\mathcal{A}|)}\}$, and n_{moved}

Having defined the conditions of safe updates, we combine them all into Algorithm 3 for safe sequential evaluation of all points, following directly from the prior Lemmas and definitions. Bounds on the Hartigan cost Equation 2 are calculated from an initial bulk computation, and only violating points need to be recomputed. The sequential kernel performs at most $2\times$ the arithmetic of a plain Hartigan sweep: once for the batch GEMM distances, $O(BkD)$, and at most once more for exact recomputation of ambiguous points. In practice, fewer than 5% of points require recomputation in steady state, so this overhead is small.

4.2 Simultaneous Execution

The sequential inner loop processes B points in series, evaluating bounds one at a time. We now classify each point's tentative decision as *safe*—certifiable under worst-case drift bounds—or *ambiguous*. The safe majority is executed in a single parallel bulk update; the small ambiguous remainder is processed sequentially afterward, with drift initialized to the exact displacement produced by the safe execution.

Definition 4.3 (Tentative decisions and aggregate flow)**.** *Given precomputed distances* $\hat{d}_{bj} = \|\mathbf{x}_b - \boldsymbol{\mu}_j^{(0)}\|^2$ *and pre-batch sizes* n_j*, the* tentative decision $T(b)$ *for point* b *is the standard Hartigan*

comparison (Algorithm 1, lines 4–9) evaluated with exact distances $\hat{d}_{bj}$ and sizes n_j. The aggregate flow *for each cluster j is $L_j = |\{b : T(b)$ removes b from $C_j\}|$, $J_j = |\{b : b$ joins C_j under $T(b)\}|$.*

Once we have the tentative leave decisions L and join decisions J, we need to know that we can re-process the points in a safer order. Lemma 4.2 shows that these updates commute.

Lemma 4.2 (Commutativity of centroid updates)**.** *Let cluster C_j have centroid $\boldsymbol{\mu}_j$ and size n_j. Suppose a set of points $S^- \subsetneq C_j$ departs and a disjoint set $S^+ \subset [N] \setminus C_j$ arrives. Then the updated centroid $\boldsymbol{\mu}'_j = (n_j\, \boldsymbol{\mu}_j - \sum_{i \in S^-} \mathbf{x}_i + \sum_{i \in S^+} \mathbf{x}_i)/(n_j - |S^-| + |S^+|)$ is independent of the order in which the individual removals and insertions are applied.*

Proof sketch. The final centroid is the final member sum divided by the final member count, both independent of update order; see Appendix H.

Given commutativity (Lemma 4.2), the final cluster state after executing a fixed set of moves is unique regardless of processing order. It remains to verify that every point's decision is also order-independent: does every point make the same reassignment choice whether it is processed first, last, or simultaneously with all others? To answer this we must bound the centroid drift and size variation that any one point might experience at an arbitrary intermediate step via Lemma 4.3.

Lemma 4.3 (Aggregate batch bounds)**.** *Let $\mathcal{B}$ be a batch with tentative decisions $\{T(b)\}_{b=1}^{B}$ and aggregate flow L_j, J_j (Definition 4.3). Write $n_j^{\min} = n_j - L_j$, $M_j = L_j + J_j$, $R_j = \sum_{b:\, T(b) \text{ touches } C_j} \sqrt{\hat{d}_{bj}}$. Here, "$T(b)$ touches C_j" means that $T(b)$ either removes b from C_j (a leave counted by L_j) or adds b to C_j (a join counted by J_j). Assume $M_j < n_j^{\min}$ for all j. The* aggregate drift bound *is $\delta_j^{\text{agg}} = R_j/(n_j^{\min} - M_j)$. Then at every intermediate step s during sequential processing of $\mathcal{B}$ in any order: **(i)** $\|\boldsymbol{\mu}_j^{(s)} - \boldsymbol{\mu}_j^{(0)}\| \le \delta_j^{\text{agg}}$ for all j. **(ii)** The Hartigan cost of point b (in cluster m) is bounded above by $w_m^{\max}(\sqrt{\hat{d}_{bm}} + \delta_m^{\text{agg}})^2$ and below by $w_j^{\min}\big[\sqrt{\hat{d}_{bj}} - \delta_j^{\text{agg}}\big]_+^2$, where $w_m^{\max} = (n_m - L_m^{-b})/(n_m - L_m^{-b} - 1)$ with $L_m^{-b} = L_m - \mathbf{1}[T(b)$ leaves $m]$, and $w_j^{\min} = n_j^{\min}/(n_j^{\min} + 1)$. **(iii)** For safe-move (S1) verification, directions reverse: $w_m^{\min} = (n_m + J_m)/(n_m + J_m - 1)$ and $w_{j^\star}^{\max} = (n_{j^\star} + J_{j^\star})/(n_{j^\star} + J_{j^\star} + 1)$.*

Algorithm 4 PARTIALPARALLELBATCH: safe/ambiguous partition execution

Require: Batch $\mathcal{B}$; precomputed distances $\{\hat{d}_{bj}\}$; batch-start centroids $\{\boldsymbol{\mu}_j^{(0)}\}$; batch-start sizes $\{n_j^{(0)}\}$.

Ensure: Updates $\{\boldsymbol{\mu}_j^{(|\mathcal{B}|)}, n_j^{(|\mathcal{B}|)}\}$ and returns $\{\delta_j^{(|\mathcal{B}|)}\}$ and n_{moved}.

Form tentative decisions in parallel

for $b \in \mathcal{B}$ **in parallel do** ▷ tentative decisions (Definition 4.3)
  $m \leftarrow$ cluster of $\mathbf{x}_b$
  $T(b) \leftarrow$ standard Hartigan decision using $\hat{d}_{bj}$ and $n_j^{(0)}$

Compute aggregate drift bounds

Compute L_j, J_j, M_j, R_j for each j ▷ Def. 4.3, Lemma 4.3
if $M_j \ge n_j^{(0)} - L_j$ for any j **then**
  return SEQUENTIALBATCH($\mathcal{B}$, $\{\hat{d}_{bj}\}$, $\{\boldsymbol{\mu}_j^{(0)}, n_j^{(0)}, \delta_j^{(0)} = 0\}$, $b_0 = 0$) ▷ validity fails → full sequential
$\delta_j^{\text{agg}} \leftarrow R_j/(n_j^{(0)} - L_j - M_j)$ for each j ▷ Lemma 4.3

Partition safe and ambiguous points

for $b \in \mathcal{B}$ **in parallel do** ▷ per-point safety classification
  Compute worst-case Hartigan cost bounds using δ_j^{agg} ▷ Lemma 4.3
  safe$[b] \leftarrow T(b)$ satisfies (S1) or (S2)
$\mathcal{S} \leftarrow \{b :$ *safe*$[b]\}$, $\mathcal{A} \leftarrow \mathcal{B} \setminus \mathcal{S}$

Execute the resulting partition

if $\mathcal{A} = \varnothing$ **then** ▷ fast path: all safe
  Execute bulk centroid update (Lemma 4.2) for all movers in $\mathcal{S}$
  return drift from bulk update
else if $\mathcal{S} = \varnothing$ **then** ▷ fallback: all ambiguous
  return SEQUENTIALBATCH($\mathcal{B}$, $\{\hat{d}_{bj}\}$, $\{\boldsymbol{\mu}_j^{(0)}, n_j^{(0)}, \delta_j^{(0)} = 0\}$, $b_0 = 0$)
else ▷ mixed: safe first, then ambiguous
  Execute bulk centroid update for movers in $\mathcal{S}$ ▷ Lemma 4.2
  Let $\{\boldsymbol{\mu}_j^{(|\mathcal{S}|)}, n_j^{(|\mathcal{S}|)}\}$ be the resulting state
  $\delta_j^{(|\mathcal{S}|)} \leftarrow \|\boldsymbol{\mu}_j^{(|\mathcal{S}|)} - \boldsymbol{\mu}_j^{(0)}\|$ **for all** j ▷ actual drift, $O(kD)$
  SEQUENTIALBATCH($\mathcal{A}$, $\{\hat{d}_{bj}\}$, $\{\boldsymbol{\mu}_j^{(|\mathcal{S}|)}, n_j^{(|\mathcal{S}|)}, \delta_j^{(|\mathcal{S}|)}\}$, $b_0 = |\mathcal{S}|$) ▷ Alg. 3
  return total per-cluster drift

Proof sketch. Repeated triangle inequalities bound each centroid displacement, while the minimum possible cluster sizes bound the Hartigan weights; see Appendix H.

Given this, we can develop a stronger theoretical statement that the items within a batch do not need to be processed in the original order. It is possible to re-order the items within a batch, based on their

prunability. This allows us to re-order them for maximal compute efficacy, while still converging to a Hartigan fixed point solution:

Theorem 4.1 (Partial parallel batch correctness)**.** *Given a GEMM batch $\mathcal{B}$ with tentative decisions $\{T(b)\}_{b\in\mathcal{B}}$, aggregate flow L_j, J_j satisfying $M_j < n_j^{\min}$ for all j, and aggregate drift bound δ_j^{agg} (Lemma 4.3), partition the batch into $\mathcal{S} = \{b \in \mathcal{B} : T(b)$ satisfies (S1) or (S2) under $\delta_j^{\text{agg}}\}$ and $\mathcal{A} = \mathcal{B} \setminus \mathcal{S}$. Then: **(a)** The bulk centroid update restricted to movers in $\mathcal{S}$ is correct: every move decreases $\mathcal{L}(\mathcal{C})$. **(b)** Sequential processing of $\mathcal{A}$ with drift initialised to $\Delta_j = \|\boldsymbol{\mu}_j^{(\text{after } \mathcal{S})} - \boldsymbol{\mu}_j^{(\text{GEMM})}\|$ is correct: each decision uses valid Hartigan cost bounds. **(c)** Every accepted move strictly decreases $\mathcal{L}(\mathcal{C})$; no cluster is emptied. The algorithm terminates at a Hartigan fixed point—the same characterisation as standard Hartigan (Algorithm 1).*

Algorithm 2 calls PARTIALPARALLELBATCH (Algorithm 4) for each GEMM batch. The per-point safety classification is $O(Bk)$, and centroid updates cost $O(BD)$ per batch and $O(ND)$ across all batches in one epoch. Both are negligible compared to the $O(BkD)$ GEMM. In the common case the vast majority of points are safe, and only a small ambiguous remainder requires sequential processing. In the limiting case where all points are safe ($\mathcal{A} = \varnothing$), the algorithm reduces to a single bulk update per batch; when all points are ambiguous ($\mathcal{S} = \varnothing$), it degrades gracefully to the sequential fallback.

At convergence, the algorithm arrives at a Hartigan fixed point (Theorem 4.1(c)), which inherits the guarantees of [33]: Hartigan fixed points are a strict subset of Lloyd fixed points. The balanced-partition traps identified by [16] cannot arise. The $\mathcal{S}$-then-$\mathcal{A}$ reordering may yield different fixed points from the same initialisation, but $\mathcal{L}$ decreases monotonically and every accepted move is individually optimal at the time it is made.

4.3 Adaptive Batch Size

The batch size B controls the GEMM shape $(B, D) \times (D, k)$ and determines how much centroid drift accumulates before distances are refreshed. Larger B amortizes BLAS overhead better, but the aggregate drift bound $\delta_j^{\text{agg}} = R_j/(n_j^{\min} - M_j)$ (Lemma 4.3) grows with B, loosening the safety bounds and increasing fallback to exact computation. The validity condition $M_j < n_j^{\min}$ of Lemma 4.3 imposes a hard upper limit on the number of moves per batch. We thus use the constraints to inform the value of B based on the frequency of updates p, and then use a moving average to smoothly update our estimate of p over time.

Constraint on B. If the reassignment rate is p (fraction of points that change cluster), a batch of B points produces $\approx pB$ moves, each touching two clusters, so the expected per-cluster load is $\mathbb{E}[M_j] \approx 2pB/k$. The validity condition $M_j < n_j^{\min}$ then requires $B < k\, n_{\min}/(2p)$ where $n_{\min} = \min_j n_j$.

Batch-adaptive EMA. We maintain an exponential moving average (EMA) of the reassignment rate, updated after every batch at $O(1)$ cost. Let $p_{\mathcal{B}} = n_{\text{moved}}/|\mathcal{B}|$ be the observed rate; the EMA update is $\hat{p} \leftarrow (1-\lambda)\hat{p} + \lambda\, p_{\mathcal{B}}$ with $\lambda \in (0,1)$ and $\hat{p}_0 = 1$ (pessimistic). The next batch size is $B \leftarrow \text{clamp}(\lfloor c\, k\, n_{\min}/(2\hat{p})\rfloor,\ B_{\min},\ B_{\max})$ where $c \in (0,1)$ is a safety factor (default 0.5), $B_{\min}$ is the minimum useful GEMM size, and $B_{\max}$ is a cache-aware ceiling. Below $B \approx 64$,

Algorithm 5 DETERMINEBATCHSIZE: EMA-adaptive batch sizing

Require: Current sizes $\{n_j^{(|\mathcal{B}|)}\}$; observed reassignment rate $p_{\mathcal{B}}$; EMA state $\hat{p}$; smoothing parameter λ; safety factor c; bounds $B_{\min}, B_{\max}$.
Ensure: Updated $\hat{p}$ and new batch size B.
Update the reassignment-rate estimate
$\hat{p} \leftarrow (1-\lambda)\,\hat{p} + \lambda\, p_{\mathcal{B}}$ ▷ EMA update (§4.3)
Apply the drift-validity constraint
$n_{\min} \leftarrow \min_j n_j^{(|\mathcal{B}|)}$
return $\text{clamp}\big(\lfloor c\, k\, n_{\min}/(2\,\hat{p})\rfloor,\ B_{\min},\ B_{\max}\big)$ ▷ batch size (§4.3)

Algorithm 6 BUILDBATCH: screen stable points and collect unsettled indices

Require: Batch size B; per-point bounds $u_i, \ell_i, \sigma_i^u, \Sigma_i^\ell$; per-cluster drift σ_j; global drift $\Sigma_{\max}$; cluster sizes n_j.
Ensure: Yields batches $\mathcal{B}_1, \mathcal{B}_2, \ldots$ of unsettled point indices, each of size $\le B$.
$\mathcal{B} \leftarrow \varnothing$ ▷ **Initialize an empty batch**
Scan and screen every point
for $i = 1, \ldots, N$ **do**
  $u_i^{\text{eff}} \leftarrow u_i + (\sigma_{m(i)} - \sigma_i^u)$ ▷ bounds from (§4.4)
  $\ell_i^{\text{eff}} \leftarrow \ell_i - (\Sigma_{\max} - \Sigma_i^\ell)$
  if $\dfrac{n_j^{\min}}{n_j^{\min}+1}(\ell_i^{\text{eff}})^2 \ge \dfrac{n_{m(i)}}{n_{m(i)}-1}(u_i^{\text{eff}})^2$ **then**
    continue ▷ screened — safe stay (§4.4)
  $\mathcal{B} \leftarrow \mathcal{B} \cup \{i\}$
  if $|\mathcal{B}| = B$ **then**
    yield $\mathcal{B}$
    $\mathcal{B} \leftarrow \varnothing$
if $\mathcal{B} \neq \varnothing$ **then** ▷ **Emit the final partial batch**
  yield $\mathcal{B}$ ▷ remaining partial batch

fixed allocation, dispatch, and thread-synchronization costs can exceed the arithmetic time. Fixing $B = 64$ produces 1.4–9× slowdown in Table 12; the practical 64–256 range balances GEMM efficiency against drift accumulation. Initially $B = \text{clamp}(\lfloor kn_{\min}/4 \rfloor, B_{\min}, B_{\max})$; as $\hat{p}$ decreases, B grows until saturating at $B_{\max}$ in later epochs.

4.4 Pruning Stable Points

Our final insight is that some points may be so stable as to make any distance calculations at all irrelevant. In later epochs most points are stable. By maintaining cheap per-point distance bounds across batches, we *screen* stable points before batch formation and exclude them from the GEMM entirely (following the Hamerly design [11]). For each point i, $m(i)$ denotes its currently assigned cluster. We maintain $u_i \geq \|\mathbf{x}_i - \boldsymbol{\mu}_{m(i)}\|$ (upper bound to assigned centre) and $\ell_i \leq \min_{j \neq m(i)} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|$ (lower bound to nearest other centre), initialised to $u_i = \infty$, $\ell_i = 0$.

To avoid $O(N)$ bound updates per batch, we develop *lazy* drift tracking to maintain per-cluster cumulative drift $\sigma_j = \sum_t \delta_j^{(t)}$ and a global accumulator $\Sigma_{\max} = \sum_t \max_j \delta_j^{(t)}$. When point i's bounds are set after a GEMM, we record $\sigma_i^u = \sigma_{m(i)}$ and $\Sigma_i^\ell = \Sigma_{\max}$. The *effective bounds* at any later time are $u_i^{\text{eff}} = u_i + (\sigma_{m(i)} - \sigma_i^u)$ and $\ell_i^{\text{eff}} = \ell_i - (\Sigma_{\max} - \Sigma_i^\ell)$, each costing $O(1)$, which is proven by Lemma 4.4.

Lemma 4.4 (Cross-batch bound validity)**.** *After any sequence of batches,* $u_i^{\text{eff}} \geq \|\mathbf{x}_i - \boldsymbol{\mu}_{m(i)}\|$ *and* $\ell_i^{\text{eff}} \leq \min_{j \neq m(i)} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|$.

Proof sketch. Cumulative cluster drift preserves the upper bound, while the sum of per-batch maximum drifts preserves the lower bound; see Appendix H.

Point i (in cluster m) is *screened* if: $\frac{n_j^{\min}}{n_j^{\min}+1}(\ell_i^{\text{eff}})^2 \geq \frac{n_m}{n_m-1}(u_i^{\text{eff}})^2$, where $n_j^{\min} = \min_{j \neq m} n_j$. This certifies (S2), preserving the guarantee of Theorem H.1.

However, the lazy updates can make the bound excessively "stale", and must be refreshed. This refresh happens at two levels. After each batch with drifts δ_j: (1) *Global* ($O(k)$): $\sigma_j \mathrel{+}= \delta_j$, $\Sigma_{\max} \mathrel{+}= \max_j \delta_j$. (2) *Per-point* ($O(|\mathcal{B}|)$): for each $i \in \mathcal{B}$, $u_i \leftarrow \sqrt{\hat{d}_{i,m(i)}} + \delta_{m(i)}$, $\sigma_i^u \leftarrow \sigma_{m(i)}$, $\ell_i \leftarrow \left[\min_{j \neq m(i)} \sqrt{\hat{d}_{ij}} - \max_j \delta_j\right]_+$, $\Sigma_i^\ell \leftarrow \Sigma_{\max}$. All bounds carry over unchanged across epoch boundaries (warm-start).

5 Results

All experiments use k-means++ initialization, $k \in \{2, 4, 8, 16, 32, 64, 128\}$ (capped at $\lfloor\sqrt{N}\rfloor$), and report the median over 10 independent trials. Within each trial, all algorithms receive identical externally selected k-means++ initial centers. All runtimes are single-threaded wall-clock seconds on a Apple M4 Pro. Our results will show that: **(1)** Batched Hartigan maintains desirable empirical properties relative to the naive variant. **(2)** Our method is faster than the prior implementations of Hartigan's algorithm. (3) That our method is comparable to popular fast Lloyd's k-means variants, making it viable for practitioners to explore, and validates our strategy of creating vectorizable computations via the triangle inequality.

Five algorithms will be compared in total: sequential Hartigan (Python with Numba JIT; Algorithm 1) and our Batched Hartigan (Algorithm 2). The Lloyd-family baselines are standard Lloyd (Python/NumPy) [19], Elkan (Cython via scikit-learn) [10], and Hamerly (C++) [11]. These will be evaluated on 12 datasets spanning $N \in [11\,055,\ 5\,000\,000]$ and $D \in [3,\ 780]$. Dataset dimensions and preprocessing are reported in Appendix D.

5.1 Fast Batched Hartigan Efficacy

First we validate that our Batched variant maintains desirable properties of Hartigan's method. Solution quality is measured by the sum of squared errors (SSE), the k-means objective $\sum_i \|\mathbf{x}_i - \boldsymbol{\mu}_{a_i}\|^2$ from (1), where a_i denotes point i's assigned cluster. This starts with the convergence speed. Figure 1 shows the ratio of each algorithm's iteration count to Lloyd's k-means. Both Hartigan

variants converge in roughly half the epochs: the median Hartigan/Lloyd iteration ratio is 0.54, and the median Batched Hartigan/Lloyd ratio is 0.56.

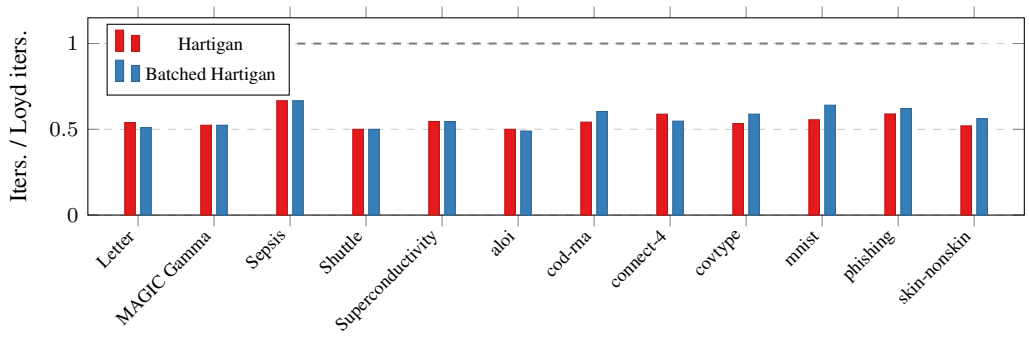

Figure 1: Median iteration ratio relative to Lloyd across k values. Both Hartigan variants converge in roughly half the epochs. This shows one advantage of Hartigan's method, and that our Batched variant maintains it in practice.

Hartigran's method converges to a superior solution set, though [32] noted the benefit was smaller when using k-means++ style seeding. Still, we confirm this in our work adn validate the same quality of solution that is generally equal-or-better using our Batched Hartigan. Table 3 reports the ratio of each Hartigan variant's converged SSE to Elkan's. Values below 1 indicate lower (better) SSE of Hartigan. Both Hartigan and Batched Hartigan consistently match or improve on Elkan's SSE, with median ratios of 0.9999 and 0.9999 respectively. This confirms that Hartigan fixed points yield equal or better clusterings than Lloyd fixed points (the latter being shared by Elkan and Lloyd), and that Batched Hartigan preserves this advantage.

Table 3: Median SSE ratio to Elkan across k values. Ratio < 1: Hartigan achieves lower SSE than Elkan.

Dataset	Hartigan / Elkan	Batched / Elkan
Letter	0.9991	0.9988
MAGIC Gamma	1.0000	1.0000
Sepsis	1.0000	1.0000
Shuttle	0.9883	0.9883
Superconductivity	0.9999	0.9999
aloi	1.0000	1.0000
cod-rna	1.0000	1.0000
connect-4	1.0000	1.0000
covtype	1.0000	1.0000
mnist	1.0000	1.0000
phishing	1.0000	1.0000
skin-nonskin	1.0000	1.0000

High-dimensional and sparse settings can produce larger differences between Lloyd and Hartigan fixed points [16]. Table 4 reports the best observed Batched Hartigan-to-Elkan inertia ratio over the tested k values and trials.

Every dataset had at least one lower-inertia configuration; ratios ranged from 0.8201 to 0.9933, or 0.7–18.0% lower inertia. Improvement is not expected at every k: very small or large k can yield simpler optima for which Lloyd and Hartigan solutions coincide. Density and runtime results are reported in Appendix E.

Table 4: Best observed sparse-data inertia ratio. Ratios are Batched Hartigan divided by Elkan; lower is better.

Dataset	D	Inertia ratio	At k
breast-cancer	10	0.9504	128
leukemia	7,129	0.9134	16
duke	7,129	0.8201	4
bibtex	1,836	0.9639	4
sector	55,197	0.9866	256
news20	62,061	0.9030	8
rcv1_multi	47,236	0.9933	256
E2006-tfidf	150,360	0.9891	32
scotus	126,405	0.9775	256
Wiki10-31K	104,374	0.9845	256
real-sim	20,958	0.9794	256

5.2 Implementation Speed

We compare our Batched Hartigan against two alternative Hartigan implementations: the sequential Hartigan from [30] (C++ with Cython bindings; `hartigan_cpp`) and the implementation of Lederman et al. [16] (Python with Numba JIT; `lederman`). All three produce Hartigan fixed points; the difference is purely in per-epoch throughput.

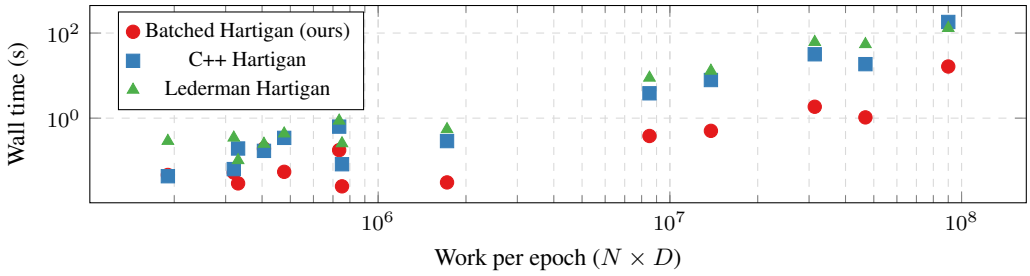

Figure 2: Wall-clock time vs. per-epoch work volume $N \times D$ (median across k, single-threaded). Batched Hartigan's GEMM acceleration and triangle-inequality pruning yield orders-of-magnitude speedups at large scale.

Table 5 gives per-dataset wall times. Times are displayed directly in seconds.

Figure 2 plots wall-clock time against the per-epoch work volume $N \times D$ (aggregated as the median across k values per dataset). Batched Hartigan is fastest on every dataset except the two smallest, where GEMM overhead dominates. The gap widens with scale: on COVTYPE ($N = 581$k, $D = 54$), Batched Hartigan is $17\times$ faster than `hartigan_cpp` and $33\times$ faster than `lederman`.

Table 5: Median wall time across k values for three Hartigan implementations, in seconds.

	Wall time (seconds)		
Dataset	Batched	C++	Lederman
phishing	0.025	0.083	0.255
MAGIC Gamma	0.046	0.043	0.290
Letter	0.053	0.063	0.347
Superconductivity	0.031	0.289	0.547
Shuttle	0.192	0.170	0.248
cod-rna	0.055	0.344	0.433
mnist	1.041	18.581	54.534
connect-4	0.381	3.837	8.879
aloi	0.503	7.871	12.934
Sepsis	0.029	0.193	0.102
skin-nonskin	0.177	0.632	0.868
covtype	1.853	31.704	61.330
SUSY	16.361	182.193	130.468

5.3 Runtime Comparison

We now compare Batched Hartigan to the accelerated Lloyd-family algorithms: Elkan [10] (scikit-learn's default), Hamerly [11] (C++ implementation of Drake and Hamerly [9]), and standard Lloyd [19] (our NumPy implementation). All run single-threaded.

Table 6 reports per-dataset median wall times. The corresponding per-k results are reported in Appendix C. At high D (e.g. MNIST, $D = 780$; ALOI, $D = 128$) Batched Hartigan is competitive with or faster than Elkan despite producing strictly better fixed points. At low D, Hamerly's cheaper $O(ND)$ bound update gives it an edge. Lloyd is consistently the slowest. This demonstates that our method is competitive with real-world k-means implementations in speed for the first time, enabling further study and experimentation.

Table 6: Median wall time across k values, in seconds.

	Wall time (seconds)			
Dataset	Lloyd	Elkan	Hamerly	Batched H.
phishing	0.136	0.016	0.062	0.025
MAGIC Gamma	0.154	0.049	0.046	0.046
Letter	0.211	0.051	0.052	0.053
Superconductivity	0.215	0.018	0.088	0.031
Shuttle	0.135	0.029	0.051	0.192
cod-rna	0.301	0.036	0.029	0.055
mnist	18.392	1.293	14.842	1.041
connect-4	4.188	0.323	2.082	0.381
aloi	9.740	0.580	2.710	0.503
Sepsis	0.061	0.016	0.011	0.029
skin-nonskin	0.586	0.086	0.077	0.177
covtype	30.495	2.503	4.269	1.853
SUSY	—	26.660	20.912	16.361

6 Conclusion

Slonim et al. [30] concluded "*Hartigan's algorithm is scarcely used in practice. Perhaps it is time for a change.*", and we elucidate the simple reason it did not happen: Hartigan's method was $100\times$ slower than alternatives. Rather than using the triangle inequality to jut prune calculations, we use it to enable vectorized calculations where none existed before.

References

[1] David Arthur and Sergei Vassilvitskii. k-means++: The Advantages of Careful Seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, volume 8, pages 1027–1035, 2007. URL http://dl.acm.org/citation.cfm?id=1283494.

[2] Thomas Bottesch, K Markus, Markus Kaechele, and U N I Ulm. Speeding up k -means by approximating Euclidean distances via block vectors. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2578–2586, 2016.

[3] François Clément and Stefan Steinerberger. An effective variant of the Hartigan k-means algorithm, April 2026. URL http://arxiv.org/abs/2604.21798. arXiv:2604.21798 [cs] version: 1.

[4] Michel B. Cordeiro and Wagner M. Nunan Zola. Parallel K-means on GPU using Warp-Centric Strategies. In *2024 IEEE 30th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 720–727, October 2024. doi: 10.1109/ICPADS63350.2024.00098. URL https://ieeexplore.ieee.org/document/10763758. ISSN: 2690-5965.

[5] Ryan R Curtin. A Dual-Tree Algorithm for Fast k-means Clustering With Large k. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 300–308. Society for Industrial and Applied Mathematics, Philadelphia, PA, June 2017. doi: 10.1137/1.9781611974973.34. URL http://epubs.siam.org/doi/10.1137/1.9781611974973.34.

[6] Yufei Ding, Yue Zhao, Xipeng Shen, Madanlal Musuvathi, and Todd Mytkowicz. Yinyang K-means: A Drop-in Replacement of the Classic K-means with Consistent Speedup. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, pages 579–587. JMLR.org, 2015. URL http://dl.acm.org/citation.cfm?id=3045118.3045181. Series Title: ICML'15.

[7] J. J. Dongarra, Jeremy Du Croz, Sven Hammarling, and I. S. Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 16(1):1–17, March 1990. ISSN 0098-3500. doi: 10.1145/77626.79170. URL https://dl.acm.org/doi/10.1145/77626.79170.

[8] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. Algorithm 656: an extended set of basic linear algebra subprograms: model implementation and test programs. *ACM Trans. Math. Softw.*, 14(1):18–32, March 1988. ISSN 0098-3500. doi: 10.1145/42288.42292. URL https://dl.acm.org/doi/10.1145/42288.42292.

[9] Jonathan Drake and Greg Hamerly. Accelerated k-means with adaptive distance bounds. In *OPT: Optimization for Machine Learning (NIPS Workshop)*, 2012.

[10] Charles Elkan. Using the triangle inequality to accelerate k-means. In *Proceedings of the 20th International Conference on Machine Learning (ICML)*, pages 147–153, 2003.

[11] Greg Hamerly. Making k-means even faster. In *Proceedings of the 2010 SIAM International Conference on Data Mining (SDM)*, pages 130–140, 2010.

[12] Greg Hamerly and Jonathan Drake. Accelerating Lloyd's Algorithm for k-Means Clustering. In *Partitional Clustering Algorithms*, pages 41–78. Springer International Publishing, 2015.

[13] J. A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, 1975.

[14] J. A. Hartigan and M. A. Wong. Algorithm AS 136: A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979. ISSN 0035-9254. doi: 10.2307/2346830. URL https://www.jstor.org/stable/2346830.

[15] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for Fortran Usage. *ACM Trans. Math. Softw.*, 5(3):308–323, September 1979. ISSN 0098-3500. doi: 10.1145/355841.355847. URL https://dl.acm.org/doi/10.1145/355841.355847.

[16] Roy R. Lederman, Daniel Silva-Sánchez, Zhizhen Chen, Gilles Mordant, Ariel Balanov, and Tamir Bendory. The catastrophic failure of the k-means algorithm in high dimensions, and how Hartigan's algorithm avoids it. *arXiv preprint arXiv:2602.09936*, 2026.

[17] Mi Li, Eibe Frank, and Bernhard Pfahringer. Large scale K-means clustering using GPUs. *Data Mining and Knowledge Discovery*, 37(1):67–109, January 2023. ISSN 1573-756X. doi: 10.1007/s10618-022-00869-6. URL https://doi.org/10.1007/s10618-022-00869-6.

[18] You Li, Kaiyong Zhao, Xiaowen Chu, and Jiming Liu. Speeding up k-Means algorithm by GPUs. *Journal of Computer and System Sciences*, 79(2):216–229, March 2013. ISSN 0022-0000. doi: 10.1016/j.jcss.2012.05.004. URL https://www.sciencedirect.com/science/article/pii/S0022000012000992.

[19] Stuart P. Lloyd. Least squares quantization in PCM. volume 28, pages 129–137, 1982.

[20] Clemens Lutz, Sebastian Breß, Tilmann Rabl, Steffen Zeuch, and Volker Markl. Efficient k-means on GPUs. In *Proceedings of the 14th International Workshop on Data Management on New Hardware*, DAMON '18, pages 1–3, New York, NY, USA, June 2018. Association for Computing Machinery. ISBN 978-1-4503-5853-8. doi: 10.1145/3211922.3211925. URL https://dl.acm.org/doi/10.1145/3211922.3211925.

[21] 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: Statistics*, volume 5.1, pages 281–298. University of California Press, January 1967. URL https://projecteuclid.org/ebooks/berkeley-symposium-on-mathematical-statistics-and-probability/Proceedings-of-the-Fifth-Berkeley-Symposium-on-Mathematical-Statistics-and/chapter/Some-methods-for-classification-and-analysis-of-multivariate-observations/bsmsp/1200512992.

[22] Bodo Manthey and Jesse van Rhijn. Worst-Case and Smoothed Analysis of the Hartigan-Wong Method for k-Means Clustering. In Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov, editors, *41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024)*, volume 289 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 52:1–52:16, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-311-9. doi: 10.4230/LIPIcs.STACS.2024.52. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.STACS.2024.52.

[23] Frank Nielsen and Richard Nock. Further heuristics for k-means: The merge-and-split heuristic and the (k,l)-means, June 2014. URL http://arxiv.org/abs/1406.6314. arXiv:1406.6314 [cs].

[24] Corey J. Nolet, Victor Lafargue, Edward Raff, Thejaswi Nanditale, Tim Oates, John Zedlewski, and Joshua Patterson. Bringing UMAP Closer to the Speed of Light with GPU Acceleration. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence*, 2021. URL http://arxiv.org/abs/2008.00325. arXiv: 2008.00325.

[25] Corey J. Nolet, Divye Gala, Edward Raff, Joe Eaton, Brad Rees, John Zedlewski, and Tim Oates. Semiring Primitives for Sparse Neighborhood Methods on the GPU. In *MLSys Conference*, 2022. URL http://arxiv.org/abs/2104.06357. arXiv: 2104.06357.

[26] Corey J. Nolet, Divye Gala, Alex Fender, Mahesh Doijade, Joe Eaton, Edward Raff, John Zedlewski, Brad Rees, and Tim Oates. cuSLINK: Single-Linkage Agglomerative Clustering on the GPU. In Danai Koutra, Claudia Plant, Manuel Gomez Rodriguez, Elena Baralis, and Francesco Bonchi, editors, *Machine Learning and Knowledge Discovery in Databases: Research Track*, volume 14169 of *Lecture Notes in Computer Science*, pages 711–726. Springer Nature Switzerland, Cham, 2023. ISBN 978-3-031-43411-2 978-3-031-43412-9. doi: 10.1007/978-3-031-43412-9_42. URL https://link.springer.com/10.1007/978-3-031-43412-9_42.

[27] Dan Pelleg and Andrew Moore. Accelerating Exact K-means Algorithms with Geometric Reasoning. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 277–281, New York, NY, USA, 1999. ACM. ISBN 1-58113-143-7. doi: 10.1145/312129.312248. URL http://doi.acm.org/10.1145/312129.312248. Series Title: KDD '99.

[28] Steven J. Phillips. Acceleration of K-Means and Related Clustering Algorithms. In *Proc. 4th Int. Workshop on Algorithm Engineering and Experiments (ALENEX 2002)*, pages 166–177, 2002. doi: 10.1007/3-540-45643-0_13. URL http://link.springer.com/10.1007/3-540-45643-0_13.

[29] Edward Raff. Exact Acceleration of K-Means ++ and K-Means. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2928–2935, 2021. doi: 10.24963/ijcai.2021/403. URL https://arxiv.org/abs/2105.02936.

[30] Noam Slonim, Ehud Aharoni, and Koby Crammer. Hartigan's K-means Versus Lloyd's K-means: Is It Time for a Change? In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, pages 1677–1684. AAAI Press, 2013. ISBN 978-1-57735-633-2. URL http://dl.acm.org/citation.cfm?id=2540128.2540369. Series Title: IJCAI '13.

[31] Colin Taylor and Michael Gowanlock. Accelerating the Yinyang K-Means Algorithm Using the GPU. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1835–1840, April 2021. doi: 10.1109/ICDE51399.2021.00163. URL https://ieeexplore.ieee.org/abstract/document/9458604. ISSN: 2375-026X.

[32] Matus Telgarsky and Andrea Vattani. Hartigan's Method: k-means Clustering without Voronoi. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9, pages 820–827, Chia Laguna Resort, Sardinia, Italy, 2010. JMLR Workshop and Conference Proceedings. URL http://proceedings.mlr.press/v9/telgarsky10a.html. Series Title: Proceedings of Machine Learning Research.

[33] Matus Telgarsky and Andrea Vattani. Hartigan's method: k-means clustering without Voronoi. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 820–827, 2010.

[34] Shuo Yang, Haocheng Xi, Yilong Zhao, Muyang Li, Xiaoze Fan, Jintao Zhang, Han Cai, Yujun Lin, Xiuyu Li, Kurt Keutzer, Song Han, Chenfeng Xu, and Ion Stoica. Flash-KMeans: Fast and Memory-Efficient Exact K-Means, April 2026. URL http://arxiv.org/abs/2603.09229. arXiv:2603.09229 [cs].

A Worked Example

This section follows one batch from screening through the exact fallback. The example uses $N = 54$ two-dimensional observations divided among $k = 3$ clusters. At the start of the batch, the cluster sizes and centroids are

$$(n_1, n_2, n_3) = (20, 4, 30), \qquad \boldsymbol{\mu}_1 = (2, 1), \quad \boldsymbol{\mu}_2 = (5, 4), \quad \boldsymbol{\mu}_3 = (8, 2).$$

The symbols $\mathbf{x}_1, \ldots, \mathbf{x}_8$ denote the eight candidate observations considered for this batch of size $B = 8$; they do not denote all N observations. The other 46 observations determine the cluster sizes and centroids above but are omitted from the plots.

A.1 Step 1: Screen Stable Candidates

The screening step decides whether an observation needs any new distance calculation. Three candidates are already stable:

candidate	coordinates	current cluster	(u_i, ℓ_i)
$\mathbf{x}_1$	$(1.70, 1.10)$	C_1	$(0.4, 3.5)$
$\mathbf{x}_2$	$(5.20, 4.10)$	C_2	$(0.3, 3.1)$
$\mathbf{x}_3$	$(8.10, 1.80)$	C_3	$(0.3, 3.0)$

Each satisfies $u_i < \ell_i$, so the assigned centroid remains closer than any competing centroid under the stored cross-batch bounds. These three observations are skipped. The five remaining candidates $\mathbf{x}_4, \ldots, \mathbf{x}_8$ enter the batch.

A.2 Step 2: Compute the Batch Distances

The active observations are stacked as the five rows of $X_{\mathcal{B}}$. One GEMM evaluates all $5 \times 3 = 15$ squared distances:

$$\widehat{D} = \|X_{\mathcal{B}}\|^2 - 2X_{\mathcal{B}}\boldsymbol{\mu}^{\mathsf{T}} + \|\boldsymbol{\mu}\|^2.$$

Table 7 gives both the coordinates and the resulting rows of $\widehat{D}$. The "current cluster" column records the assignment before this batch; it is not inferred from marker color or shape.

Table 7: The five active observations and the squared distances computed by one GEMM.

Point	Coordinates	Current cluster	$\hat{d}_{i1}$	$\hat{d}_{i2}$	$\hat{d}_{i3}$
$\mathbf{x}_4$	$(2.40, 0.80)$	C_1	0.20	17.00	32.80
$\mathbf{x}_5$	$(4.80, 4.40)$	C_2	19.40	0.20	16.00
$\mathbf{x}_6$	$(7.60, 2.30)$	C_3	33.05	9.65	0.25
$\mathbf{x}_7$	$(4.00, 4.00)$	C_1	13.00	1.00	20.00
$\mathbf{x}_8$	$(3.25, 2.32)$	C_1	3.30	5.88	22.66

A.3 Step 3: Certify Safe Decisions

The S1/S2 tests combine the precomputed distances with the current cluster sizes and drift bounds. They certify four of the five active observations:

- $\mathbf{x}_4$, $\mathbf{x}_5$, and $\mathbf{x}_6$ satisfy the safe-stay condition (S2): for each point, every candidate cluster's lower-bound Hartigan cost is at least the current cluster's upper-bound cost. Therefore they remain in C_1, C_2, and C_3, respectively.
- $\mathbf{x}_7$ satisfies the safe-move condition (S1): the upper-bound cost of moving to C_2 is below both the lower-bound cost of remaining in C_1 and the lower-bound cost of moving to any other candidate cluster. Therefore it moves from C_1 to C_2.
- $\mathbf{x}_8$ is not certified by either test and is deferred to the sequential fallback.

The ambiguity of $\mathbf{x}_8$ can be seen numerically. The move of $\mathbf{x}_7$ permits centroid drifts $\delta_1 = 0.190$ and $\delta_2 = 0.200$. Applying the Hartigan weights and the distance bounds from Lemma 4.1 gives

$$[H_1^{\mathrm{lo}}, H_1^{\mathrm{hi}}] = [2.80, 4.25], \qquad [H_2^{\mathrm{lo}}, H_2^{\mathrm{hi}}] = [4.13, 5.75].$$

The intervals overlap on $[4.13, 4.25]$. Therefore the upper bound for staying is not below the lower bound for moving, so S2 cannot certify a stay; S1 also cannot certify a move.

Figure 3 visualizes Steps 1 and 3. In both panels, horizontal and vertical position are simply the observation's first and second feature values. Color and shape encode only the computational status stated in each legend, not cluster membership.

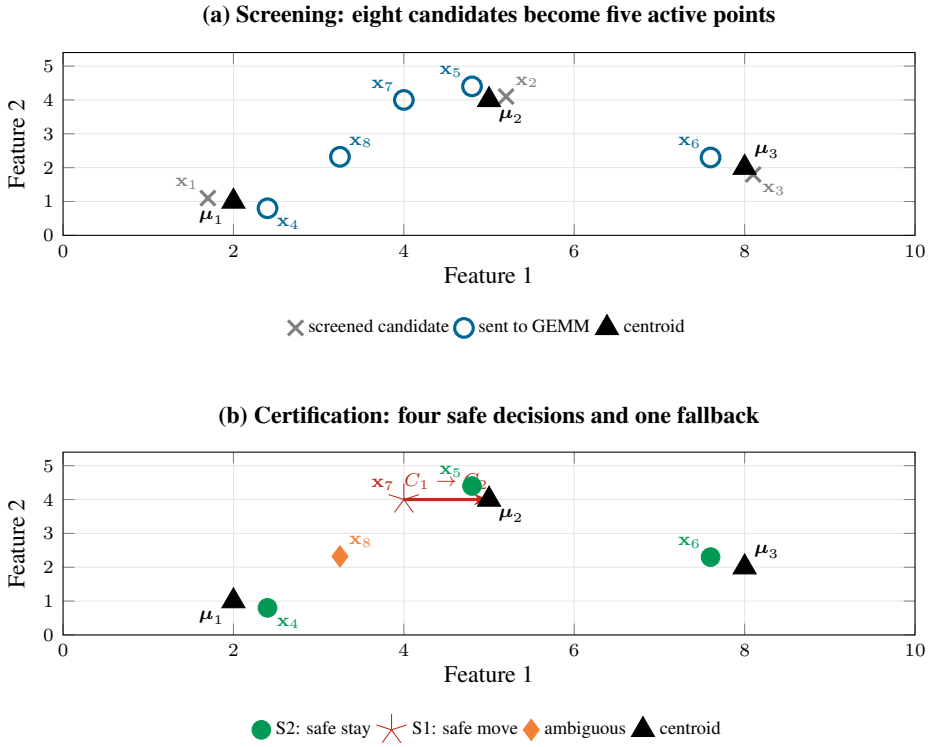

Figure 3: The eight candidate observations in the worked example. **(a)** Gray crosses are removed by cross-batch screening, while blue open circles proceed to the GEMM. **(b)** Green circles are certified S2 stays, the red star is the S1 move from C_1 to C_2, and the orange diamond requires exact fallback. Black triangles are centroids in both panels. The axes give the two feature coordinates.

A.4 Step 4: Apply the Certified Move

The three S2 decisions leave their assignments unchanged. Applying the single S1 decision moves $\mathbf{x}_7 = (4, 4)$ from C_1 to C_2, so the sizes change from $(20, 4, 30)$ to $(19, 5, 30)$. Both affected centroids must be updated:

$$\boldsymbol{\mu}_1' = \frac{20(2,1) - (4,4)}{19} = \left(\frac{36}{19}, \frac{16}{19}\right) \approx (1.895, 0.842), \quad \|\boldsymbol{\mu}_1' - \boldsymbol{\mu}_1\| \approx 0.190,$$

$$\boldsymbol{\mu}_2' = \frac{4(5,4) + (4,4)}{5} = (4.8, 4), \qquad \|\boldsymbol{\mu}_2' - \boldsymbol{\mu}_2\| = 0.200.$$

This is the bulk-update result: the three certified stays require no change, and the one certified move updates only its source and destination clusters.

A.5 Step 5: Resolve the Ambiguous Point Exactly

The fallback now processes $\mathbf{x}_8 = (3.25, 2.32)$ against the updated centroids and sizes. The exact squared distances are

$$\|\mathbf{x}_8 - \boldsymbol{\mu}_1'\|^2 = 4.021,$$
$$\|\mathbf{x}_8 - \boldsymbol{\mu}_2'\|^2 = 5.225,$$
$$\|\mathbf{x}_8 - \boldsymbol{\mu}_3\|^2 = 22.665.$$

Because $\mathbf{x}_8$ currently belongs to C_1, staying uses the removal weight $19/(19-1)$. Moving to C_2 or C_3 uses the corresponding insertion weight. The exact Hartigan costs are therefore

$$\Delta_H^2(\mathbf{x}_8, C_1) = \frac{19}{18}(4.021) = 4.244,$$
$$\Delta_H^2(\mathbf{x}_8, C_2) = \frac{5}{6}(5.225) = 4.354,$$
$$\Delta_H^2(\mathbf{x}_8, C_3) = \frac{30}{31}(22.665) = 21.934.$$

The current-cluster cost 4.244 is the smallest, so the exact fallback keeps $\mathbf{x}_8$ in C_1. The batch is complete: three observations were screened before GEMM, four decisions were certified from the precomputed distances and bounds, and only one observation required exact sequential work.

B Visual Depiction of Batched Hartigan Bounds

Figure 4 illustrates the geometric interpretation of the Hartigan cost bounds H^{lo} and H^{hi} used for safe reassignment decisions. Each point $\mathbf{x}_b$ has a precomputed distance to each centroid $\boldsymbol{\mu}_j^{(0)}$, but subsequent reassignments of *other* points cause centroids to drift by at most δ_j. The triangle inequality then bounds the true (unknown) distance within an interval, and the Hartigan weight $\frac{n_j}{n_j \pm 1}$ converts these distance bounds into cost bounds. Geometrically, each cluster's possible Hartigan cost occupies an annular band centered on $\mathbf{x}_b$: the inner arc corresponds to $\sqrt{H^{\text{lo}}}$ and the outer arc to $\sqrt{H^{\text{hi}}}$.

When the bands are well-separated, a decision can be made without recomputing exact distances: **(S1)** if the worst-case candidate cost is still below the best-case current cost, the point is safely moved; **(S2)** if the best-case cost of every candidate exceeds the worst-case current cost, the point safely stays. Large clusters produce narrower bands (the Hartigan weight approaches 1), making safe decisions more likely.

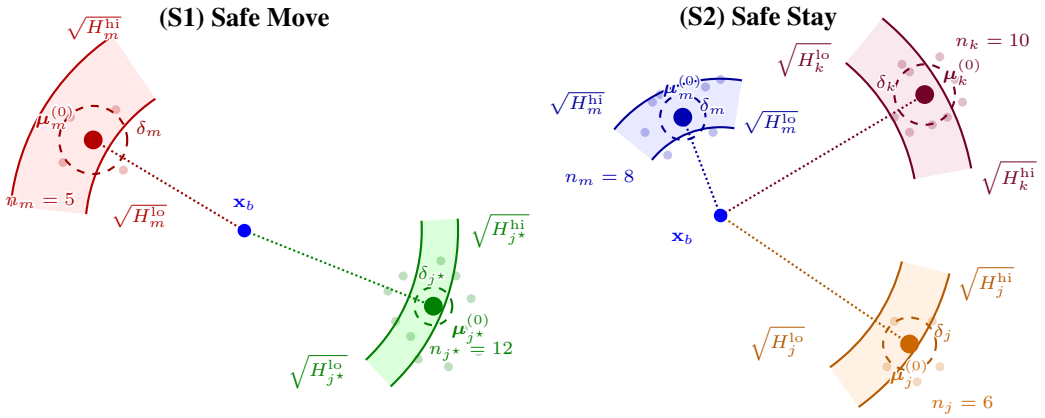

Figure 4: Geometric interpretation of Hartigan cost bounds. Dashed circles show centroid drift uncertainty δ_j (the true centroid lies somewhere inside). Colored bands show the range of possible $\sqrt{H}$ values (Hartigan-weighted distance from $\mathbf{x}_b$); inner arcs are $\sqrt{H^{\text{lo}}}$, outer arcs are $\sqrt{H^{\text{hi}}}$. The Hartigan weight $\frac{n_j}{n_j \pm 1}$ shrinks bands for large clusters, making safe decisions more frequent. **Left:** $H_{j^\star}^{\text{hi}} < H_m^{\text{lo}}$ so the point safely moves (worst-case candidate still better than best-case current). **Right:** $H_j^{\text{lo}} \geq H_m^{\text{hi}}$ for all $j \neq m$, so the point safely stays (best-case candidate still worse than worst-case current).

C Per-k Runtime Results

This section reports the k-dependence underlying the aggregate runtime comparison in subsection 5.3. We use the eight datasets with $N \geq 50{,}000$ and complete Lloyd timings.

Figure 5 shows Batched Hartigan speedup relative to Lloyd for each dataset and k.

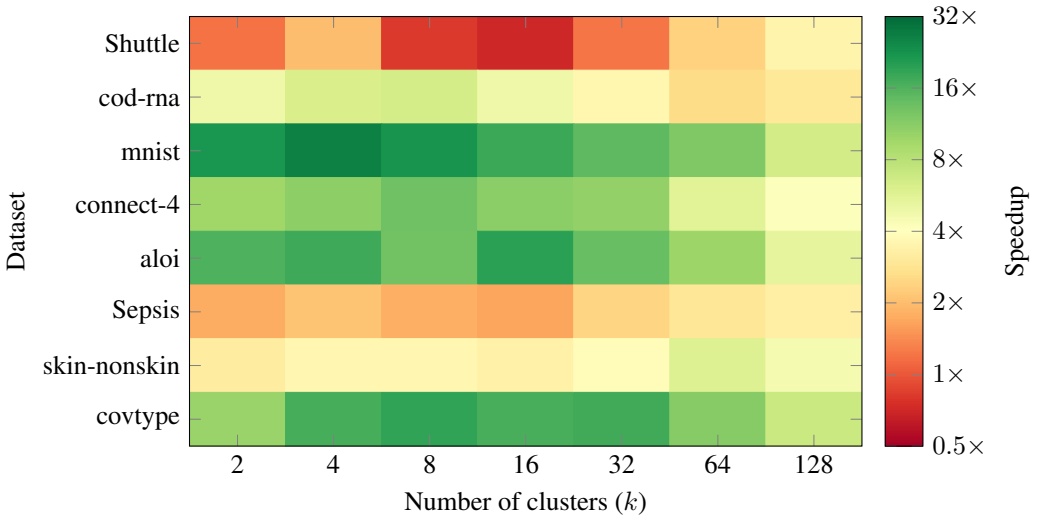

Figure 5: Batched Hartigan speedup relative to Lloyd by dataset and k. Color encodes log-base-two speedup; $1\times$ is parity and higher is faster.

Table 8 summarizes all three comparisons over the same datasets.

Table 8: Per-k speedup relative to Lloyd. Entries are mean (minimum–maximum); higher is faster.

k	Prior H. (C++)	Elkan	Batched H. (ours)
2	2.6 (0.3–4.6)	12.8 (2.6–23.7)	8.5 (1.2–21.3)
4	2.3 (0.3–5.9)	13.2 (4.0–23.4)	10.6 (2.0–26.0)
8	1.6 (0.5–3.1)	12.0 (3.8–19.6)	9.9 (0.8–21.9)
16	0.9 (0.2–1.2)	9.6 (2.8–16.8)	9.4 (0.7–19.4)
32	0.7 (0.3–1.2)	8.6 (3.7–13.0)	8.4 (1.2–17.2)
64	0.6 (0.2–0.9)	7.3 (4.5–11.4)	6.5 (2.3–11.8)
128	0.4 (0.1–1.2)	6.0 (4.4–8.4)	4.6 (3.0–6.9)

Elkan and Batched Hartigan retain the same order of speedup at every k. Mean prior C++ Hartigan speedup falls below Lloyd from $k = 16$ onward.

D Dataset Characteristics

This section reports the input scales and preprocessing used in section 5. Table 9 lists all datasets in increasing order of N.

Features were used at source scale without additional normalization. For UCI data, nonnumeric or entirely missing columns and rows containing missing feature values were removed. LIBSVM inputs were densified without rescaling.

E Sparse Results

This section reports the density and runtime context for the sparse-data quality results in subsection 5.1. We evaluated 11 LIBSVM datasets in CSR format using shared k-means++ initializations, three trials, $k \in \{4, 8, 16, 32, 64, 128, 256\}$ when $k < N$, and at most 300 iterations.

Table 10 reports dataset scale, the best observed inertia ratio, and runtime at the corresponding k.

Across all 213 configurations, runtime speedup ranged from $0.08\times$ to $3.20\times$, with median $0.54\times$; Batched Hartigan was faster in 72 configurations. Quality and runtime gains therefore occur in different regimes.

Table 9: Dataset dimensions and total dense data elements.

Dataset	N	D	$N \times D$
phishing	11,055	68	752K
MAGIC Gamma	19,020	10	190K
Letter	20,000	16	320K
Superconductivity	21,263	81	1.7M
Shuttle	58,000	7	406K
cod-rna	59,535	8	476K
mnist	60,000	780	46.8M
connect-4	67,557	126	8.5M
aloi	108,000	128	13.8M
Sepsis	110,341	3	331K
skin-nonskin	245,057	3	735K
covtype	581,012	54	31.4M
SUSY	5,000,000	18	90M

Table 10: Sparse benchmark results. The inertia ratio is Batched Hartigan divided by Elkan for the best observed trial. Runtime speedup is median Elkan time divided by median Batched Hartigan time over three trials at the selected k; values above one favor Batched Hartigan.

Dataset	N	D	Density	k	Inertia ratio	Runtime speedup
breast-cancer	683	10	1.0000	128	0.9504	1.51×
leukemia	38	7,129	1.0000	16	0.9134	0.25×
duke	44	7,129	1.0000	4	0.8201	0.29×
bibtex	7,395	1,836	0.0374	4	0.9639	1.73×
sector	6,412	55,197	0.0030	256	0.9866	1.57×
news20	15,935	62,061	0.0013	8	0.9030	0.10×
rcv1_multi	15,564	47,236	0.0014	256	0.9933	1.83×
E2006-tfidf	16,087	150,360	0.0083	32	0.9891	0.34×
scotus	7,800	126,405	0.0107	256	0.9775	0.23×
Wiki10-31K	20,762	104,374	0.0065	256	0.9845	0.28×
real-sim	72,309	20,958	0.0024	256	0.9794	1.25×

F Lloyd Refinement

This section evaluates Batched Hartigan initialized from a converged Lloyd solution. This refinement is valid because Hartigan fixed points form a subset of Lloyd fixed points [33].

For each dataset and selected k, we ran five shared k-means++ initializations. Elkan was run to a Lloyd fixed point; Batched Hartigan was run both from the original initialization and from Elkan's converged centers. Table 11 reports quality and refinement cost.

Table 11: Batched Hartigan as a Lloyd refinement. Direct is the best observed Batched-Hartigan-to-Elkan inertia ratio from the original three-trial sweep; Refine is the median ratio over five refinement trials. Overhead is the median of $100\, t_{\mathrm{refine}}/t_{\mathrm{Elkan}}$.

Dataset	N	D	Density	k	Direct	Refine	Overhead
breast-cancer	683	10	1.0000	128	0.9504	0.9754	44.1%
leukemia	38	7,129	1.0000	16	0.9134	0.9407	305.4%
duke	44	7,129	1.0000	4	0.8201	0.9551	298.5%
bibtex	7,395	1,836	0.0374	4	0.9639	0.9864	75.7%
sector	6,412	55,197	0.0030	256	0.9866	0.9869	77.2%
news20	15,935	62,061	0.0013	8	0.9030	0.9305	493.8%
rcv1_multi	15,564	47,236	0.0014	256	0.9933	0.9934	63.5%
E2006-tfidf	16,087	150,360	0.0083	32	0.9891	1.0073	130.1%
scotus	7,800	126,405	0.0107	256	0.9775	0.9801	392.9%
Wiki10-31K	20,762	104,374	0.0065	256	0.9845	0.9917	362.7%
real-sim	72,309	20,958	0.0024	256	0.9794	0.9843	81.6%

Refinement lowered median inertia on 10 of 11 datasets, but its ratio was higher than the selected direct-start result in every row. Refinement added 44.1–493.8% of one Elkan run. We did not evaluate matched-budget independent Lloyd restarts, so these results do not establish a comparison with that strategy.

G Ablation Study

We conduct two ablation studies to validate the design of Batched Hartigan: (1) a component-removal study showing that each algorithmic component contributes meaningfully, and (2) a hyperparameter sensitivity study confirming that defaults are robust across diverse datasets. All experiments use single-threaded NumPy (no multi-core BLAS), tolerance $\tau = 0$, maximum 1000 iterations, and are averaged over 2 trials with shared k-means++ initialization. Seven datasets are tested spanning N=11k to N=5M.

G.1 Component Removal

We test five ablated variants against the full Batched Hartigan algorithm:

- **No BuildBatch**: Screening is disabled; all N points are treated as unsettled every epoch, eliminating the $O(N)$ savings from skipping converged points.
- **No Parallel**: The bulk GEMM-based reassignment path is disabled; all unsettled points are processed by the sequential kernel with per-point $O(k)$ cost.
- **No Sequential**: The drift-aware sequential bounds (S1/S2) are disabled; ambiguous points are resolved by brute-force exact-distance Hartigan reassignment.
- **No Dynamic** $B_{\max}$: Batch size is fixed at $B_{\max} = 4096$ (no EMA-based adaptation).
- **No Dynamic** $B_{\min}$: Batch size is fixed at $B_{\min} = 64$.

Table 12 reports the wall-time ratio of each variant relative to the full algorithm (values >1 indicate slowdown). All variants converge to equivalent inertia ($<0.01\%$ difference), confirming that the components affect *speed*, not solution quality.

Table 12: Component ablation: wall-time ratio relative to full Batched Hartigan. Values >1 indicate slowdown. Bold marks the most impactful removal per row.

Dataset	k	No Build	No Par.	No Seq.	No Dyn.$_{\max}$	No Dyn.$_{\min}$
phishing	8	1.48	0.71	0.97	0.95	**5.10**
	32	1.26	0.80	1.05	1.00	**3.63**
Letter	8	1.86	0.69	0.95	0.98	**6.26**
	32	1.58	0.80	0.87	0.89	**5.94**
	128	2.00	0.92	1.00	1.03	**3.47**
mnist	8	1.55	0.93	1.32	1.00	**2.73**
	32	1.11	0.93	**1.51**	1.00	1.90
	128	1.34	1.05	**1.53**	1.01	1.42
connect-4	8	1.62	0.72	1.02	0.95	**4.85**
	32	1.46	0.96	1.15	1.00	**3.50**
	128	1.43	0.84	1.21	1.01	**1.97**
aloi	8	2.23	0.69	0.99	0.96	**5.59**
	32	2.10	0.87	1.05	1.00	**4.49**
	128	1.20	0.68	1.07	1.00	**2.29**
covtype	8	4.03	0.69	1.00	1.00	**5.81**
	32	**18.91**	1.23	0.83	0.83	5.68
	128	**10.90**	0.67	0.76	1.02	3.11
SUSY	8	3.12	0.87	1.02	1.00	**8.13**
	32	**9.55**	**1.27**	0.99	1.00	8.93
	128	**4.73**	0.94	0.98	0.99	3.82

Screening (BuildBatch) is consistently important. Removing screening forces reprocessing of all N points every epoch, producing $2-19\times$ slowdowns that grow with dataset size. On covtype (N=581k, k=32): $18.9\times$ slower; on SUSY (N=5M, k=32): $9.6\times$ slower. On large datasets most points are stable after early epochs, and screening correctly identifies them.

The parallel path is critical for large k and large N. Table 13 shows the fraction of non-screened point operations resolved by the parallel BLAS path versus those requiring sequential $O(k)$ processing.

Table 13: Impact of removing the parallel path. Without it, all non-screened operations become sequential $O(k)$-per-point work, growing dramatically with k and N — up to $1{,}671\times$ on SUSY (N=5M).

Dataset	k	% Parallel (full)	Seq. ops ratio	Iter. ratio
phishing	8	69.7%	3.5×	1.04×
	32	71.5%	3.2×	0.82×
Letter	8	80.4%	5.2×	1.00×
	32	86.4%	7.4×	0.95×
	128	90.5%	11.3×	1.13×
mnist	8	87.7%	8.3×	1.00×
	32	89.9%	10.3×	0.98×
	128	94.4%	23.1×	1.23×
connect-4	8	81.0%	5.3×	0.96×
	32	85.1%	6.9×	0.99×
	128	87.1%	6.6×	0.79×
aloi	8	94.5%	19.5×	1.02×
	32	95.5%	23.7×	1.04×
	128	96.8%	24.5×	0.61×
covtype	8	98.8%	90.1×	1.07×
	32	99.1%	121.5×	5.83×
	128	99.2%	125.6×	0.16×
SUSY	8	99.8%	604.2×	0.92×
	32	99.9%	1,039.9×	0.92×
	128	99.9%	1,671.3×	0.87×

Without the parallel path, the volume of sequential operations increases by orders of magnitude on large datasets: $90-130\times$ on covtype (N=581k) and $604-1{,}671\times$ on SUSY (N=5M). The growth with k is clear: at k=128 the full algorithm resolves $\geq 99\%$ of non-screened points via bulk BLAS, but removing this forces all of them into per-point $O(k)$ sequential processing. On SUSY k=32 this translates directly to a $1.27\times$ wall-time slowdown even in single-threaded execution.

Observed pruning rates. After epoch 2, cross-batch screening removes 50–95% of points. Of the remainder, the parallel safety path certifies 70–99.9% (Table 13); the sequential fallback handles fewer than 5% of points in steady state.

Fixed small batch size is catastrophic. Setting $B{=}B_{\min}{=}64$ produces $1.4-9\times$ slowdowns across all configurations due to excessive per-batch overhead. The largest impact is on SUSY ($8.9\times$ at k=32). The adaptive EMA-based sizing avoids this failure mode automatically.

The sequential drift-aware bounds provide modest but consistent benefit. Removing the sequential kernel's S1/S2 bounds (brute-forcing ambiguous points) incurs $1.3-1.5\times$ overhead on high-dimensional data (mnist D=780), where recomputing exact distances is expensive. On lower-dimensional data the brute-force cost is negligible.

G.2 Hyperparameter Sensitivity

We vary each hyperparameter independently by $10\times$ above and below its default, holding all others constant. Table 14 reports the wall-time ratio relative to the default configuration.

Table 14: Hyperparameter sensitivity: wall-time ratio vs. default. Values near 1.0 indicate insensitivity. All configurations produce equivalent inertia ($<0.01\%$ variation).

Parameter	Default	Swept	Median time ratio (range across datasets)		
		values	$0.1\times$	$1\times$	$10\times$
$B_{\min}$	64	6, 64, 640	1.00 (0.99–1.02)	1.00	1.00 (0.99–1.01)
$B_{\max}$	4096	410, 4096, 40960	1.81 (0.65–2.75)	1.00	1.08 (0.72–3.29)
λ	0.1	0.01, 0.1, 1.0	1.00 (0.91–1.25)	1.00	1.00 (0.96–1.02)
c	0.5	0.05, 0.5, 5.0	1.00 (0.82–1.44)	1.00	1.00 (0.97–1.02)

Only $B_{\max}$ has a meaningful impact: reducing it $10\times$ to 410 causes median $1.8\times$ slowdown (more batches per epoch), while increasing it $10\times$ to 40,960 produces mixed results—faster on large-N/low-D datasets but up to $3.3\times$ slower on high-D data (mnist k=128) due to oversized GEMM operations. The remaining parameters ($B_{\min}$, λ, c) are essentially inert: $10\times$ perturbations in either direction produce $<5\%$ median runtime variation. Combined with zero effect on solution quality, this confirms that users need not tune these parameters.

H Supporting Lemmas and Proofs

The following lemma is used throughout the proofs below.

Lemma H.1 (Center drift)**.** *Suppose point $\mathbf{x}_i$ is reassigned from cluster C_m (with $n_m \geq 2$) to cluster $C_{j^\star}$. Let $\boldsymbol{\mu}_m, \boldsymbol{\mu}_{j^\star}$ denote the centroids* before *the move. Then the centroids after the move satisfy $\boldsymbol{\mu}'_m = \frac{n_m \boldsymbol{\mu}_m - \mathbf{x}_i}{n_m - 1}$ and $\boldsymbol{\mu}'_{j^\star} = \frac{n_{j^\star} \boldsymbol{\mu}_{j^\star} + \mathbf{x}_i}{n_{j^\star} + 1}$ with changes $\|\boldsymbol{\mu}'_m - \boldsymbol{\mu}_m\| = \frac{\|\mathbf{x}_i - \boldsymbol{\mu}_m\|}{n_m - 1}$ and $\|\boldsymbol{\mu}'_{j^\star} - \boldsymbol{\mu}_{j^\star}\| = \frac{\|\mathbf{x}_i - \boldsymbol{\mu}_{j^\star}\|}{n_{j^\star} + 1}$ respectively.*

Proof of Lemma H.1 (Center drift). For removal:

$$\boldsymbol{\mu}'_m - \boldsymbol{\mu}_m = \frac{n_m \boldsymbol{\mu}_m - \mathbf{x}_i}{n_m - 1} - \boldsymbol{\mu}_m = \frac{\boldsymbol{\mu}_m - \mathbf{x}_i}{n_m - 1}.$$

Taking norms gives the removal result. For insertion:

$$\boldsymbol{\mu}'_{j^\star} - \boldsymbol{\mu}_{j^\star} = \frac{n_{j^\star} \boldsymbol{\mu}_{j^\star} + \mathbf{x}_i}{n_{j^\star} + 1} - \boldsymbol{\mu}_{j^\star} = \frac{\mathbf{x}_i - \boldsymbol{\mu}_{j^\star}}{n_{j^\star} + 1},$$

giving the result of the Lemma. □

Proof of Lemma 4.2 (Commutativity of centroid updates). Each incremental update $\boldsymbol{\mu} \leftarrow (n\boldsymbol{\mu} \pm \mathbf{x}_i)/(n \pm 1)$ preserves the invariant $\boldsymbol{\mu} = (\sum_{i \in C_j} \mathbf{x}_i)/|C_j|$. After any permutation of the $|S^-| + |S^+|$ operations, the membership set is $C'_j = (C_j \setminus S^-) \cup S^+$, whose member sum and cardinality are permutation-invariant. □

Proof of Lemma 4.1 (Distance bound propagation). (i) By the triangle inequality, $\|\mathbf{x}_b - \boldsymbol{\mu}_j^{(b-1)}\| \leq \|\mathbf{x}_b - \boldsymbol{\mu}_j^{(0)}\| + \|\boldsymbol{\mu}_j^{(b-1)} - \boldsymbol{\mu}_j^{(0)}\| \leq \sqrt{\hat{d}_{bj}} + \delta_j^{(b-1)}$, and the reverse triangle inequality gives the lower bound. Squaring preserves order on $[0, \infty)$. (ii) The Hartigan weights $n_j^{(b-1)}/(n_j^{(b-1)} \pm 1)$ are positive, so multiplying by the appropriate weight preserves the inequalities. □

Theorem H.1 (Correctness of Batched Hartigan)**.** *Consider standard Hartigan (Algorithm 1) processing points in a fixed order $1, 2, \ldots, N$. Let Batched Hartigan process the same points in the same order, partitioned into contiguous batches of size at most B. If Batched Hartigan makes a reassignment only under condition* (S1) *and skips a reassignment only under condition* (S2)*, falling back to exact distance computation in all ambiguous cases, then the sequence of moves produced by Batched Hartigan is identical to that of standard Hartigan.*

Proof of Theorem H.1 (Correctness of Batched Hartigan). By induction on the position in the processing order.

Base case. The first point in the first batch has $\delta_j^{(0)} = 0$ for all j, so all bounds are exact and conditions (S1), (S2) reduce to the standard Hartigan comparison.

Inductive step. Suppose that through points $1, \ldots, b-1$, Batched Hartigan has produced the same moves as standard Hartigan. Then the cluster state (memberships, sizes, centroids) is identical at step b. For point b:

- The precomputed distances $\hat{d}_{bj}$ were computed with respect to $\boldsymbol{\mu}_j^{(0)}$.
- The cumulative drifts $\delta_j^{(b-1)}$ correctly bound $\|\boldsymbol{\mu}_j^{(b-1)} - \boldsymbol{\mu}_j^{(0)}\|$ by Definition 4.1.
- The sizes $n_j^{(b-1)}$ are tracked exactly.

If (S1) holds, the true Hartigan cost to $j^\star$ is at most $H_{j^\star}^{\mathrm{hi}}$, which is strictly less than the lower bound on every other option. Exact Hartigan would also choose $j^\star$. If (S2) holds, the true cost to every candidate is at least $H_j^{\mathrm{lo}} \geq H_m^{\mathrm{hi}}$, which exceeds the true cost of staying. Exact Hartigan would also keep the point. If neither holds, exact distances are recomputed, so the decision is trivially identical. In every case the move matches, the induction carries, and the cluster state remains synchronised. □

Proof of Lemma 4.3 (Aggregate batch bounds). (i) Let $D_j^{(s)} := \|\boldsymbol{\mu}_j^{(s)} - \boldsymbol{\mu}_j^{(0)}\|$ denote the total drift of cluster j after s moves within the batch have been processed (in any order), with $D_j^{(0)} = 0$. We show $D_j^{(s)} \leq \delta_j^{\mathrm{agg}}$ for all s.

Per-move drift. Suppose that at step s a point $\mathbf{x}_{b_s}$ is added to or removed from cluster j. By Lemma H.1, the centroid shift is

$$\|\boldsymbol{\mu}_j^{(s)} - \boldsymbol{\mu}_j^{(s-1)}\| \;=\; \frac{\|\mathbf{x}_{b_s} - \boldsymbol{\mu}_j^{(s-1)}\|}{n_j^{(s-1)} \pm 1}\,, \tag{4}$$

where $n_j^{(s-1)} \pm 1$ denotes $n_j^{(s-1)} - 1$ for removal or $n_j^{(s-1)} + 1$ for insertion (both $\geq n_j^{\min}$ since at most L_j points leave cluster j).

Bounding the intermediate distance. The numerator involves the intermediate centroid $\boldsymbol{\mu}_j^{(s-1)}$, not the precomputed $\boldsymbol{\mu}_j^{(0)}$. By the triangle inequality:

$$\|\mathbf{x}_{b_s} - \boldsymbol{\mu}_j^{(s-1)}\| \;\leq\; \|\mathbf{x}_{b_s} - \boldsymbol{\mu}_j^{(0)}\| \;+\; \|\boldsymbol{\mu}_j^{(s-1)} - \boldsymbol{\mu}_j^{(0)}\| \;=\; \sqrt{\hat{d}_{b_s,j}} + D_j^{(s-1)}. \tag{5}$$

Recurrence. Combining (4) and (5), and bounding the denominator below by $n_j^{\min}$:

$$D_j^{(s)} \;\leq\; D_j^{(s-1)} + \frac{\sqrt{\hat{d}_{b_s,j}} + D_j^{(s-1)}}{n_j^{\min}} \;=\; D_j^{(s-1)}\Big(1 + \frac{1}{n_j^{\min}}\Big) + \frac{\sqrt{\hat{d}_{b_s,j}}}{n_j^{\min}}\,. \tag{6}$$

Write $\alpha = 1/n_j^{\min}$. With $D_j^{(0)} = 0$ the recurrence $D^{(s)} \leq D^{(s-1)}(1+\alpha) + \alpha\sqrt{\hat{d}_{b_s,j}}$ unrolls by induction to

$$D_j^{(M_j)} \;\leq\; \alpha \sum_{t=1}^{M_j} \sqrt{\hat{d}_{b_t,j}}\,(1+\alpha)^{M_j - t} \;\leq\; \alpha\,(1+\alpha)^{M_j - 1} \cdot R_j\,, \tag{7}$$

where the second inequality holds because $(1+\alpha)^{M_j - t}$ is largest for $t = 1$ and $R_j = \sum_t \sqrt{\hat{d}_{b_t,j}}$ (Lemma 4.3).

It remains to bound $\alpha(1+\alpha)^{M_j - 1}$ by $1/(n_j^{\min} - M_j)$. We use the elementary inequality

$$\big(1 + \tfrac{1}{n}\big)^M \;\leq\; e^{M/n} \;\leq\; \frac{n}{n - M} \qquad (0 \leq M < n), \tag{8}$$

where the first step is $(1+1/n)^n \le e$ and the second follows from $e^x(1-x) \le 1$ for $x \ge 0$ (since $\frac{d}{dx}[e^x(1-x)] = -x\,e^x \le 0$ and the value at $x=0$ is 1). Applying (8) to (7) with $n = n_j^{\min}$ and $M = M_j$:

$$D_j^{(M_j)} \;\le\; \frac{1}{n_j^{\min}} \cdot \frac{n_j^{\min}}{n_j^{\min} - M_j} \cdot R_j \;=\; \frac{R_j}{n_j^{\min} - M_j} \;=\; \delta_j^{\mathrm{agg}}.$$

Since only $s \le M_j$ of the batch moves touch cluster j and the recurrence is monotone in s, the bound $D_j^{(s)} \le \delta_j^{\mathrm{agg}}$ holds for every intermediate step.

(ii) During sequential processing, the size of cluster j lies in $[n_j - L_j,\ n_j + J_j]$. The Hartigan weight for the current cluster is $n_m^{(s)}/(n_m^{(s)} - 1)$, which is maximised when $n_m^{(s)}$ is smallest. The smallest possible size when processing point b is $n_m - L_m^{-b}$, where L_m^{-b} excludes b's own tentative departure (which has not yet occurred when b is evaluated), giving $w_m^{\max}$ (Lemma 4.3(ii)). The candidate weight $n_j^{(s)}/(n_j^{(s)} + 1)$ is minimised when $n_j^{(s)}$ is smallest, giving $w_j^{\min}$. Combining these weights with the squared-distance bounds $\big[\sqrt{\hat{d}_{bj}} \pm \delta_j^{\mathrm{agg}}\big]_+^2$ from (i) yields the Hartigan cost bounds.

(iii) For safe-move verification, the worst-case directions reverse: we need the *smallest* current-cluster weight (maximise $n_m^{(s)}$, giving $n_m + J_m$) and the *largest* candidate weight (maximise $n_j^{(s)}$, giving $n_j + J_j$). □

Proof of Lemma H.2 (Aggregate bounds with pre-existing drift). The argument follows the drift recurrence (6) from Lemma 4.3, but with a nonzero initial condition. Write $\alpha = 1/n_j^{\min}$ and let $D_j^{(s)}$ denote the total drift from the GEMM centroids after s moves within $\mathcal{A}$ have been processed. Now $D_j^{(0)} = \Delta_j^{\mathrm{pre}}$ (not zero).

The per-move drift bound (4) and intermediate distance bound (5) are unchanged, since they depend only on the triangle inequality and the current cluster size (which is at least $n_j^{\min}$). The recurrence (6) therefore reads

$$D_j^{(s)} \;\le\; D_j^{(s-1)}(1+\alpha) + \alpha\,\sqrt{\hat{d}_{b_s,j}}\,,$$

with $D_j^{(0)} = \Delta_j^{\mathrm{pre}}$. Unrolling:

$$D_j^{(M_j)} \;\le\; \Delta_j^{\mathrm{pre}}\,(1+\alpha)^{M_j} \;+\; \alpha\,(1+\alpha)^{M_j-1} \cdot R_j^{\mathcal{A}}. \tag{9}$$

Applying the elementary bound $(1+1/n)^M \le n/(n-M)$ from (8) with $n = n_j^{\min}$ and $M = M_j$:

$$\begin{aligned} D_j^{(M_j)} \;&\le\; \Delta_j^{\mathrm{pre}} \cdot \frac{n_j^{\min}}{n_j^{\min} - M_j} \;+\; \frac{R_j^{\mathcal{A}}}{n_j^{\min} - M_j} \\ &=\; \frac{\Delta_j^{\mathrm{pre}} \cdot n_j^{\min} + R_j^{\mathcal{A}}}{n_j^{\min} - M_j} \;=\; \delta_j^{\mathrm{eff}}. \end{aligned}$$

Since $D_j^{(s)}$ is monotonically non-decreasing in s and only M_j moves touch cluster j, the bound holds at every intermediate step. When $\Delta_j^{\mathrm{pre}} = 0$, the expression reduces to $\delta_j^{\mathrm{agg}} = R_j/(n_j^{\min} - M_j)$ from Lemma 4.3. □

Lemma H.2 (Aggregate bounds with pre-existing drift)**.** *Let $\mathcal{A}$ be a set of points with tentative decisions having aggregate flow $L_j^{\mathcal{A}}, J_j^{\mathcal{A}}$ and batch quantities as in Lemma 4.3, computed with respect to sizes n_j. Suppose $\|\boldsymbol{\mu}_j^{(\mathrm{current})} - \boldsymbol{\mu}_j^{(\mathrm{GEMM})}\| \le \Delta_j^{\mathrm{pre}}$. Write $n_j^{\min} = n_j - L_j^{\mathcal{A}}$, $M_j = L_j^{\mathcal{A}} + J_j^{\mathcal{A}}$, and assume $M_j < n_j^{\min}$. Then at every intermediate step, $\|\boldsymbol{\mu}_j^{(\mathrm{step})} - \boldsymbol{\mu}_j^{(\mathrm{GEMM})}\| \le \delta_j^{\mathrm{eff}} := (\Delta_j^{\mathrm{pre}} \cdot n_j^{\min} + R_j^{\mathcal{A}})/(n_j^{\min} - M_j)$, where $R_j^{\mathcal{A}} = \sum_{b \in \mathcal{A}:\, T(b) \text{ touches } j} \sqrt{\hat{d}_{bj}}$.*

Proof of Theorem 4.1 (Partial parallel batch correctness). (a) Restricting the batch to $\mathcal{S}$ gives reduced flow $L_j^{\mathcal{S}} \le L_j$, $J_j^{\mathcal{S}} \le J_j$, $R_j^{\mathcal{S}} \le R_j$, and $n_j^{\min,\mathcal{S}} = n_j - L_j^{\mathcal{S}} \ge n_j^{\min}$. The restricted drift bound therefore satisfies

$$\delta_j^{\mathcal{S}} \;=\; \frac{R_j^{\mathcal{S}}}{n_j^{\min,\mathcal{S}} - M_j^{\mathcal{S}}} \;\le\; \delta_j^{\mathrm{agg}}.$$

Points in $\mathcal{S}$ satisfy (S1) or (S2) under the larger bound δ_j^{agg}, hence they also satisfy the conditions under $\delta_j^{\mathcal{S}}$. By Lemma 4.3, the drift at every intermediate step of a hypothetical sequential processing of $\mathcal{S}$ remains within $\delta_j^{\mathcal{S}} \le \delta_j^{\text{agg}}$, so each decision is correct: the Hartigan cost strictly decreases. By Lemma 4.2, the bulk centroid update yields the same final state as any sequential ordering of $\mathcal{S}$'s movers.

(b) After executing $\mathcal{S}$'s bulk update, the actual per-cluster displacement from the GEMM reference is $\Delta_j = \|\boldsymbol{\mu}_j^{(\text{after } \mathcal{S})} - \boldsymbol{\mu}_j^{(\text{GEMM})}\|$. This is an *exact* measurement (not a bound). Sequential processing of $\mathcal{A}$ starting from this centroid state uses the SequentialBatch algorithm (Algorithm 3) with per-cluster drift initialised to Δ_j. The drift at every subsequent step is bounded by Lemma H.2 (Lemma H.2). Each point in $\mathcal{A}$ therefore evaluates Hartigan cost bounds that are valid—tighter than under δ_j^{agg}—and makes the optimal reassignment decision or falls through to the exact-distance computation.

(c) Every accepted move (whether from $\mathcal{S}$ or $\mathcal{A}$) strictly decreases $\mathcal{L}(\mathcal{C})$: moves from $\mathcal{S}$ by part (a), and moves from $\mathcal{A}$ because each sequential decision is a valid Hartigan move with respect to the current centroid state. Since $\mathcal{L}$ is bounded below and strictly decreases with each accepted move, the algorithm terminates in a finite number of epochs.

At convergence, no points move, so $L_j = J_j = M_j = 0$ and $\delta_j^{\text{agg}} = 0$ for all j. Every point is in $\mathcal{S}$ (all are safe stayers) and the partition satisfies: for every point $\mathbf{x}_i$ in cluster m, no single-point move to any cluster $j \neq m$ would decrease $\mathcal{L}(\mathcal{C})$. This is exactly the Hartigan fixed-point condition. □

Proof of Lemma 4.4 (Cross-batch bound validity). Let the bounds u_i, ℓ_i be set exactly after batch t_0 from a GEMM, with snapshots $\sigma_i^u = \sigma_{m(i)}^{(t_0)}$ and $\Sigma_i^\ell = \Sigma_{\max}^{(t_0)}$.

Upper bound. By the triangle inequality, at any later batch $t > t_0$:

$$\|\mathbf{x}_i - \boldsymbol{\mu}_{m(i)}^{(t)}\| \;\le\; \|\mathbf{x}_i - \boldsymbol{\mu}_{m(i)}^{(t_0)}\| + \|\boldsymbol{\mu}_{m(i)}^{(t)} - \boldsymbol{\mu}_{m(i)}^{(t_0)}\| \;\le\; u_i + \big(\sigma_{m(i)}^{(t)} - \sigma_{m(i)}^{(t_0)}\big) = u_i^{\text{eff}},$$

where the second inequality uses the triangle inequality on the cumulative drift: $\|\boldsymbol{\mu}_{m(i)}^{(t)} - \boldsymbol{\mu}_{m(i)}^{(t_0)}\| \le \sum_{s=t_0+1}^{t} \delta_{m(i)}^{(s)} = \sigma_{m(i)}^{(t)} - \sigma_{m(i)}^{(t_0)}$.

Lower bound. For any cluster $j \neq m(i)$, the reverse triangle inequality gives $\|\mathbf{x}_i - \boldsymbol{\mu}_j^{(t)}\| \ge \|\mathbf{x}_i - \boldsymbol{\mu}_j^{(t_0)}\| - (\sigma_j^{(t)} - \sigma_j^{(t_0)})$. Taking the minimum over $j \neq m(i)$:

$$\min_{j \neq m(i)} \|\mathbf{x}_i - \boldsymbol{\mu}_j^{(t)}\| \;\ge\; \ell_i - \max_{j \neq m(i)} \big(\sigma_j^{(t)} - \sigma_j^{(t_0)}\big).$$

It remains to show $\Sigma_{\max}^{(t)} - \Sigma_{\max}^{(t_0)} \ge \max_j(\sigma_j^{(t)} - \sigma_j^{(t_0)})$. Expanding the left side:

$$\Sigma_{\max}^{(t)} - \Sigma_{\max}^{(t_0)} \;=\; \sum_{s=t_0+1}^{t} \max_j \delta_j^{(s)} \;\ge\; \max_j \sum_{s=t_0+1}^{t} \delta_j^{(s)} \;=\; \max_j \big(\sigma_j^{(t)} - \sigma_j^{(t_0)}\big),$$

where the inequality holds because $\sum_s \max_j f_j(s) \ge \max_j \sum_s f_j(s)$ for any non-negative $f_j(s)$ (the sum of per-step maxima dominates the maximum of per-cluster sums). Therefore $\ell_i^{\text{eff}} = \ell_i - (\Sigma_{\max}^{(t)} - \Sigma_i^\ell)$ is a valid lower bound. □

I GPU Speedup

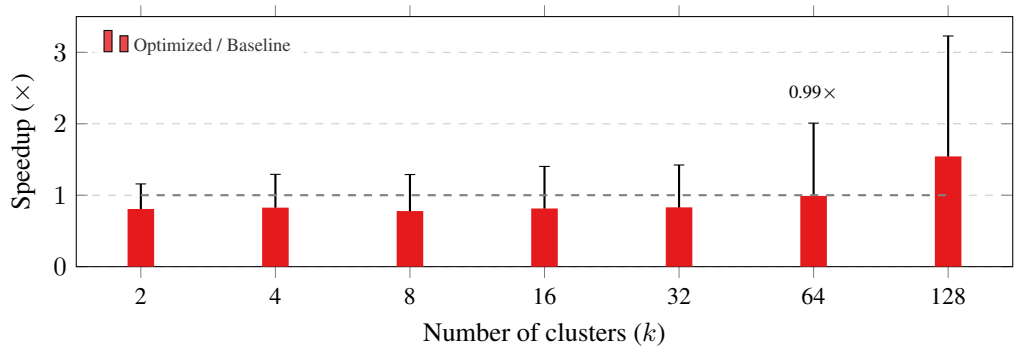

Figure 6: **GPU speedup after optimization (NVIDIA Tesla T4).** Bars show mean speedup across 14 datasets (Letter Recognition, MAGIC Gamma Telescope, SUSY, Sepsis Survival, Skin Segmentation, Statlog Shuttle, Superconductivity, aloi, cod-rna, connect-4, covtype, mnist, phishing, skin-nonskin). Error bars indicate min-max range across datasets. **Optimizations:** (1) Persistent GPU memory: centers and cluster sizes remain on GPU across batches within each epoch, updated in-place; (2) GPU-resident distance computation: GEMM output ($B \times k$) stays on GPU for downstream operations; (3) Vectorized tentative decisions: broadcast Hartigan weights to ($B \times k$), compute element-wise costs, parallel row-wise argmin on GPU instead of nested CPU loops; (4) GPU-accelerated S1/S2 safety checks: compute distance bounds on GPU, transfer only boolean mask (B bits) instead of intermediate matrices ($2Bk$ floats). **Performance trends:** For $k \leq 32$, fixed GPU overheads (kernel launch, synchronization, memory allocation) dominate, resulting in 0.81× average speedup (19% slower). For $k = 64$, performance approaches parity (0.99×). For $k = 128$, the $O(Bk)$ operations achieve high GPU occupancy and amortize fixed costs, yielding 1.54× average speedup with best case 3.2× on SUSY ($N = 5$M, $D = 18$). See Figure 7 for per-dataset breakdown.

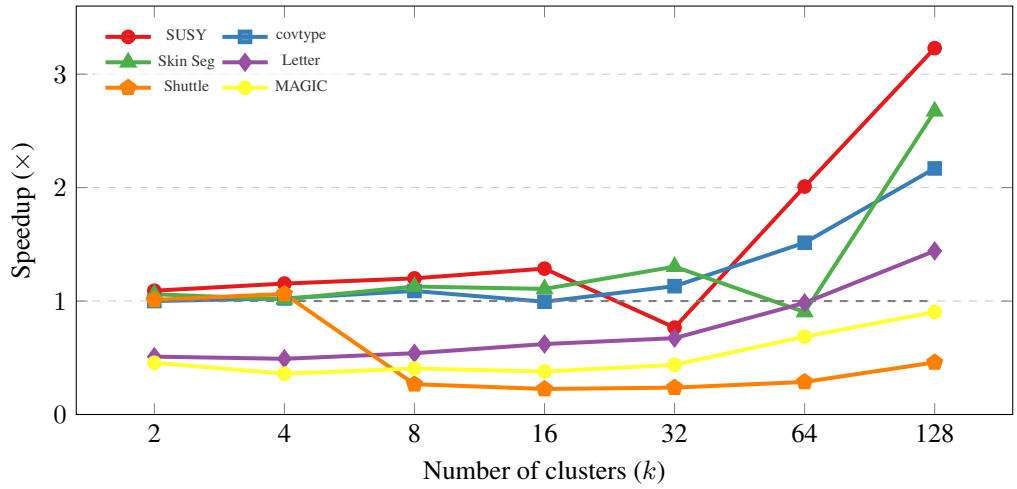

Figure 7: **Per-dataset GPU speedup breakdown (line plot).** Six representative datasets shown across all k values. **Key observations:** (1) Large datasets with high dimensionality benefit most: SUSY ($N = 5$M, $D = 18$) achieves 3.2× at $k = 128$; covtype ($N = 581$K, $D = 54$) achieves 2.2×; Skin Segmentation ($N = 245$K, $D = 3$) achieves 2.7×. (2) Small, low-dimensional datasets suffer: MAGIC Gamma ($N = 19$K, $D = 10$) and Letter Recognition ($N = 20$K, $D = 16$) show 50–60% slowdown for $k \leq 32$. (3) Pathological case: Statlog Shuttle ($N = 58$K, $D = 7$) shows severe degradation (0.23× at $k = 16$), likely due to small batch sizes and poor GPU occupancy. (4) Crossover point: Most datasets achieve parity around $k = 64$ and benefit at $k = 128$. This heterogeneity suggests **adaptive backend selection**: use CPU for small k or small datasets, GPU for large k with sufficient data.

NeurIPS Paper Checklist

1. **Claims**

Algorithm 7 Baseline GPU Batch Processing

Input: Batch $X_b \in \mathbb{R}^{B \times D}$, centers $C \in \mathbb{R}^{k \times D}$
GPU: Transfer X_b, C to GPU
GPU: Compute $\hat{D} = \text{GEMM}(X_b, C)$
Transfer: $\hat{D} \to$ CPU
CPU: Compute tentative assignments
for $i = 1$ to B **do**
  for $j = 1$ to k **do**
    Compute weighted distance
  $j^* \leftarrow \arg\min$ weighted distance
CPU: S1/S2 safety checks
CPU: Update centers, cluster sizes
Output: Updated centers, labels

Algorithm 8 Optimized GPU Batch Processing

Input: Batch $X_b \in \mathbb{R}^{B \times D}$, persistent $C_{gpu} \in \mathbb{R}^{k \times D}$
GPU: Transfer X_b to GPU only
GPU: Compute $\hat{D}_{gpu} = \text{GEMM}(X_b, C_{gpu})$ [keep on GPU]
GPU: Vectorized tentative assignments
$W \leftarrow$ broadcast weights $(B \times k)$
costs $\leftarrow \hat{D}_{gpu} \odot W$
$j^* \leftarrow \arg\min(\text{costs}, \text{axis} = 1)$
GPU: S1/S2 safety checks (vectorized)
Transfer: Safety mask $\in \{0, 1\}^B \to$ CPU
CPU: Update centers, cluster sizes
Output: Updated centers, labels

Figure 8: Comparison of baseline and optimized GPU batch processing. **Key changes (blue):** (1) Keep distance matrix $\hat{D}$ on GPU throughout processing; (2) Vectorize tentative decisions as parallel matrix operations instead of nested CPU loops; (3) Compute safety checks on GPU, transfer only boolean mask; (4) Persist centers on GPU across batches (not shown). Data transfer reduced from $O(Bk)$ floats to $O(B)$ bools. Speedup scales with k because GPU parallelism increases while fixed overheads remain constant.

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [Yes]

Justification: The abstract and introduction state the computational motivation, the Batched Hartigan contribution, and the main empirical claims. These claims are supported by the algorithmic development in Sections 4–5 and the appendix experiments.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses important limitations in Sections 4.3, 5.3, and 6, including the dependence on batch size, dimensionality, and the limited current benefit for multi-core/GPU-style parallelism.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The theoretical statements define their assumptions in Sections 4.1–4.4. Complete proofs for the stated lemmas and theorems are provided in Appendix H.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper gives the algorithms, datasets, tested k values, number of trials, initialization, and hardware in Section 5 and Appendix G.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

Justification: The code will be released soon; the bureaucracy is expanding to meet the needs of the expanding bureaucracy. Sorry about that.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5 specifies the initialization, tested k range, trial count, aggregation statistic, compared algorithms, dataset scale, and single-threaded hardware setting. Appendix G gives additional algorithm hyperparameters and ablation settings.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The main results report medians over independent trials but do not provide error bars, confidence intervals, or significance tests.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 5 reports single-threaded wall-clock times on an Apple M4 Pro. No exotic compute is needed.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work studies a clustering algorithm and evaluates it on benchmark datasets without human-subject experiments, high-risk model release, or privacy-sensitive data collection described in the paper. We are not aware of any aspect of the research that deviates from the NeurIPS Code of Ethics.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: We don't think anyone is concerned that clustering could lead to evil AGI.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release high-risk pretrained models, image generators, scraped datasets, or other assets with a high risk of misuse. Therefore special release safeguards are not applicable.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [No]

Justification: The paper cites the algorithms and implementations used for comparison, but it does not explicitly state dataset/code package versions, licenses, or terms of use for all existing assets.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: Released code will be licensed at its Github repo.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments or research with human subjects. No participant instructions, screenshots, or compensation details are applicable.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects. IRB or equivalent approval details are therefore not applicable.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development does not use LLMs as an important, original, or non-standard research component. Any LLM use limited to writing, editing, or formatting would not require declaration under this checklist item.