GACHA: Adaptive Randomized Neighborhood Search for Clustering Noisy DNA Storage Reads

Puru Sharma*,†, Hugo Maximus Lim*, Zhenhao Gu, Care Greene Lee,
Quinn Chua, Gary Goh Yipeng, and Weng-Fai Wong

National University of Singapore, Singapore

Abstract

Recovering information stored in DNA requires clustering noisy sequencing reads by their originating strands before consensus reconstruction. At archive scale, hundreds of millions of reads must be grouped despite insertion, deletion, and substitution errors, making exhaustive pairwise comparisons infeasible and individual similarity representations susceptible to missed matches. We introduce GACHA, an adaptive randomized clustering algorithm that discovers complementary same-source relationships through repeated, independently randomized similarity orderings. In each pass, GACHA samples representatives from current clusters, constructs error-tolerant multi-strobe keys, and examines only nearby representatives after sorting. A cluster-size-dependent neighborhood budget prioritizes smaller, potentially incomplete clusters, while a cascade of blocked k-mer signatures and thresholded edit-distance verification limits expensive comparisons. Accepted matches are merged through union–find, progressively reducing the search space in subsequent passes. This combination of repeated randomized candidate discovery and size-adaptive agglomeration avoids exhaustive pairwise search and supports efficient GPU parallelization.

Across four real Illumina and Nanopore DNA storage datasets, GACHA achieves median Accuracy(0.95) of at least 0.978 and a 14-fold geometric-mean speedup over the fastest competing method for each dataset. On a synthetic 1 GB retrieval benchmark comprising approximately 370 million reads, it completes the clustering stage in 44 seconds, 762 times faster than the fastest completed baseline, while maintaining at least 99.99% clustering accuracy. These results demonstrate that adaptive randomized neighborhood search provides an effective algorithmic foundation for accurate, large-scale clustering of noisy DNA reads.

Keywords: Sequence clustering; randomized algorithms; DNA data storage; similarity search.
Code: `https://github.com/prongs1996/gacha/`

*Joint first author. †Correspondence to: `puru@u.nus.edu`.

1 Introduction

DNA-based data storage is a promising medium for long-term archival storage due to its high information density and stability [1]. Advances in synthesis and sequencing have enabled increasingly large storage demonstrations [2, 3], but efficient retrieval remains a challenge. High-throughput sequencing produces unordered copies of the stored DNA strands, corrupted by insertion, deletion, and substitution errors. Before consensus reconstruction and decoding, these reads must be clustered by their originating strands. Clustering directly affects reconstruction accuracy and can become a major computational bottleneck [4].

Clustering noisy DNA reads requires balancing accuracy against computational cost. Exhaustive pairwise comparison is quadratic in the number of reads and infeasible for archives of hundreds of millions of reads. Existing methods reduce comparisons using q-gram signature filters [5], locality-sensitive hashing [6], specialized similarity indexes [7, 8], and approximate sequence representations [5, 9]. However, restrictive representations can miss related reads when errors disrupt shared features, while more permissive ones generate large candidate sets that require expensive verification. Missed matches fragment reads from the same source into separate clusters, whereas false matches merge distinct strands and compromise reconstruction. The challenge is to identify enough reliable matches without comparing excessive numbers of read pairs.

A useful observation is that clustering does not require finding every pair of reads from the same strand. It is enough to discover a set of correct matches that connect the reads from each source, without linking distinct sources. Rather than relying on a single similarity representation, multiple inexpensive randomized views can expose different same-source relationships. As clusters merge, the search adapts to the remaining components instead of repeatedly examining individual reads. Independent randomization provides fresh opportunities to discover previously missed matches, enabling geometric convergence when unresolved fragments retain a positive probability of merging in every pass.

We introduce GACHA, an adaptive randomized neighborhood-search algorithm for clustering noisy DNA storage reads. GACHA alternates between randomized candidate discovery and agglomerative clustering. In each pass, it samples representatives from the current clusters, constructs randomized multi-strobe keys, and compares nearby representatives in the resulting sorted order. A cluster-size-dependent neighborhood budget gives smaller, potentially incomplete clusters a wider search range. Candidate pairs are screened using blocked k-mer signatures, with thresholded edit-distance verification reserved for ambiguous cases. Accepted matches are merged through union–find [10]. Subsequent passes resample representatives and construct new randomized orderings, exposing relationships missed in earlier passes while progressively reducing the search space. These operations can be efficiently parallelized on GPUs.

We evaluate GACHA on four real Illumina and Nanopore DNA storage datasets and synthetic workloads of increasing scale. On the real datasets, GACHA achieves median Accuracy(0.95) of at least 0.978 and a 14-fold geometric-mean speedup over the fastest competing method on each dataset. On a synthetic 1 GB retrieval benchmark containing approximately 370 million reads, it completes clustering in 44 seconds, a 762× speedup over the fastest completed baseline, while maintaining at least 99.99% clustering accuracy. Ablations confirm that fresh randomization in every pass is what recovers the clusters, as Accuracy(0.95) falls from 1.0000 to 0.017 with a fixed seed.

Our main contributions are: (i) an adaptive randomized clustering algorithm built on independently randomized multi-strobe orderings, (ii) a cluster-size-dependent neighborhood search with a cascaded verifier, (iii) an analysis showing that independent randomization enables repeated opportunities to recover missed matches, while incorrect merges remain irreversible, and (iv) a GPU implementation evaluated on real and synthetic DNA storage datasets at archive scale.

2 Problem Formulation and Related Work

2.1 Clustering Noisy DNA Storage Reads

In DNA-based data storage, digital information is encoded into short synthetic DNA strands, and sequencing produces multiple noisy copies of each, with substitutions, insertions, and deletions arising during synthesis, storage, and sequencing. The decoding pipeline clusters reads by their source strands, reconstructs a consensus sequence for each cluster, and recovers the data using strand indices and error-correcting codes. We focus on clustering, where neither the original sequences nor the read-to-strand assignments are known.

Let $S = \{s_1, \ldots, s_{K_0}\}$ denote the K_0 designed strands and $R = \{r_1, \ldots, r_n\}$ the observed reads, treating identical sequences as distinct read instances. Each read r_j originates from an unknown strand s_{z_j}, where $z_j \in \{1, \ldots, K_0\}$. The ground-truth partition is $\mathcal{C}^* = \{C_i^* \mid C_i^* \neq \emptyset,\ i \in \{1, \ldots, K_0\}\}$, where $C_i^* = \{r_j \in R \mid z_j = i\}$. We write $K = |\mathcal{C}^*|$ for the number of nonempty source clusters, which may be smaller than K_0. The goal is to recover a partition $\hat{\mathcal{C}}$ approximating $\mathcal{C}^*$, assigning every read to exactly one cluster. An expected cluster count $m^\star$ may be available.

Clustering errors can either split reads from the same source across multiple clusters or merge reads from different sources, compromising subsequent consensus reconstruction, and exhaustive pairwise comparison requires $O(n^2)$ edit-distance computations. Scalable clustering must therefore identify reliable candidate matches without examining every read pair.

2.2 Existing Clustering Approaches

General-purpose sequence clustering. General-purpose tools such as CD-HIT [11], UCLUST [12], DNACLUST [13], MeShClust [14], and Starcode [15] use greedy representative assignment, alignment-free similarity estimation, or edit-distance-based matching. However, these approaches can be costly at DNA storage scale, where millions of source strands generate hundreds of millions of noisy reads. Greedy centroid methods are also sensitive to read ordering and identity thresholds, potentially fragmenting clusters or merging distinct sources.

DNA-storage-specific clustering. Specialized methods reduce comparisons through approximate candidate discovery. Rashtchian et al. [5] combine representative sampling, anchor-based hashing, and blocked q-gram filtering, but may miss noisy matches. Clover [7] uses tree-based indexing, while locality-sensitive hashing (LSH) [6] trades candidate recall against search cost. GradHC [16] improves reliability through iterative hash-based refinement, whereas HSFC [9] uses compact hash sketches and fuzzy clustering. Learning-based methods estimate similarity through sequence embeddings [17, 18] or raw Nanopore signals [19], but have not demonstrated scalability to hundreds of millions of reads. Across these approaches, efficiently recovering missed matches without generating excessive candidates remains challenging.

3 The GACHA Algorithm

3.1 Algorithm Overview

GACHA performs multi-pass agglomerative clustering through randomized candidate discovery and cluster merging (Algorithm 1). Each pass samples cluster representatives, sorts them by randomized multi-strobe keys, and examines nearby pairs using size-adaptive neighborhoods. A cascaded verifier filters candidates before exact edit-distance verification, with accepted matches merged through union–find. Subsequent passes resample representatives and construct fresh orderings to recover previously missed relationships. Building on the agglomerative framework of Rashtchian et al. [5], GACHA replaces anchor-based hashing with randomized sorted-neighborhood search, introduces cluster-size-dependent budgets, and eliminates signature-only acceptance to prevent irreversible false merges.

3.2 Agglomerative Clustering and Representative Sampling

GACHA maintains the clustering with a union–find structure [10]. At the start of pass t the partition is $\mathcal{C}^{(t)} = \{C_1^{(t)}, \ldots, C_{m_t}^{(t)}\}$ with active roots $\mathcal{A}^{(t)}$, $|\mathcal{A}^{(t)}| = m_t$, and initially every read is a singleton. Accepted pairs merge their components, so the clusters are the connected components of the graph of accepted matches. Recovering a source therefore needs only enough correct matches to connect its reads, rather than comparisons between every pair.

In each pass, GACHA samples α representatives from every active component, with $\alpha = 1$ by default. Resampling lets different members of a component take part across passes, exposing matches missed by earlier representatives, and pass t processes only $M_t = \alpha m_t$ representatives rather than all n reads, a number that shrinks as components merge.

Algorithm 1 GACHA: Adaptive Randomized Clustering

Require: Reads $\mathcal{R} = \{r_1, \ldots, r_n\}$; representatives per cluster α; neighborhood constant X; matched-prefix length P; thresholds T, S_{rej}; maximum passes $P_{\max}$; patience p; optional target cluster count $m^\star$

Ensure: Partition $\widehat{\mathcal{C}}$ of $\mathcal{R}$

if any threshold is automatic **then**
  $(T, S_{\text{rej}}) \leftarrow \textsc{Calibrate}(\mathcal{R}, \alpha, X)$
Initialize union–find with one component per read
$\mathcal{A}^{(0)} \leftarrow \{1, \ldots, n\}$; $z \leftarrow 0$
for $t = 0, \ldots, P_{\max} - 1$ **do**
  $\mathcal{V}_t \leftarrow \textsc{SampleRepresentatives}(\mathcal{A}^{(t)}, \alpha)$
  for all $r \in \mathcal{V}_t$ **in parallel do**
    $k_t(r) \leftarrow \textsc{MultiStrobeKey}(r, t)$
    $\kappa_t(r) \leftarrow \max\left\{1, \left\lfloor \frac{X}{|C_t(r)|} \right\rfloor\right\}$
  $\mathbf{v}_t \leftarrow \textsc{SortByKey}(\mathcal{V}_t, k_t)$
  $U_t \leftarrow 0$
  for all $(r_i, r_j) \in \textsc{CandidatePairs}(\mathbf{v}_t, k_t, \kappa_t, P)$ **in parallel do**
    if $\textsc{Find}(r_i) \neq \textsc{Find}(r_j)$ **then**
      if $\textsc{Verify}(r_i, r_j, T, S_{\text{rej}})$ **then**
        if $\textsc{AtomicUnion}(r_i, r_j)$ **then**
          $\textsc{AtomicAdd}(U_t, 1)$
  $\mathcal{A}^{(t+1)} \leftarrow \textsc{UpdateActiveRoots}(\mathcal{A}^{(t)})$
  $z \leftarrow z + 1$ if $U_t = 0$; otherwise $z \leftarrow 0$
  if $(p > 0 \wedge z \geq p)$ **or** ($m^\star$ specified $\wedge\ |\mathcal{A}^{(t+1)}| \leq m^\star$) **then**
    break
return the partition induced by union–find

3.3 Randomized Multi-Strobe Candidate Generation

GACHA uses randomized multi-strobe keys to place potentially related reads near one another in a sorted ordering. Inspired by strobemer-based sequence comparison [20], each key combines short subsequences from several regions of a read, and selecting local hash minima [21, 22] instead of fixed-position substrings makes these features less sensitive to the small shifts caused by insertions and deletions.

Let q be the strobe length, L the number of nominal strobe locations, H the number selected per key, and W the number of candidate q-mer positions examined around each selected location. A read r of length ℓ has $a_r = \ell - q + 1$ valid q-mer starts, and the nominal locations are $p_j(r) = \left\lfloor \frac{(j+1)a_r}{L+1} \right\rfloor$ for $j = 0, \ldots, L-1$. In pass t, a randomized ordered selection π_t chooses H distinct locations, each jittered and clamped to the valid range to give a window $\mathcal{W}_{t,j}(r)$ of at most W positions, and the minimum-hash q-mer in the window is selected: $p^*_{t,j}(r) = \arg\min_{p \in \mathcal{W}_{t,j}(r)} h_{t,j}(r[p : p+q])$, with $x_{t,j}(r) = r[p^*_{t,j}(r) : p^*_{t,j}(r) + q]$, where $h_{t,j}$ is a pass-specific hash function. A second hash gives the field $u_{t,j}(r) = \text{MSB}_b(g_{t,j}(x_{t,j}(r), j))$, the b most significant bits of the hash of the selected q-mer and its nominal location, and the complete key is $k_t(r) = u_{t,\pi_t(1)}(r) \| \cdots \| u_{t,\pi_t(H)}(r) \| \tau_t(r)$, concatenating the selected fields in order with a randomized tie-breaker $\tau_t(r)$.

By default, $q = 4$, $L = 14$, $H = 7$, $W = 8$, and $b = 8$, so the seven fields occupy 56 bits and the eight-bit tie-breaker completes a 64-bit key that can be efficiently radix-sorted. (See supplement for parameter justification.) Each pass refreshes the representative samples, selected locations and their order, positional jitter, hash functions, and tie-breakers, so reads that are distant in one ordering may become neighbors in another. Two windows agree on a field with a probability that depends only on how many q-mers they share, not on the sequences themselves (Lemma 1).

3.4 Size-Adaptive Candidate Neighborhoods

GACHA sorts the M_t representatives by their keys, yielding the ordered pairs $(k_{(1)}, v_{(1)}), \ldots, (k_{(M_t)}, v_{(M_t)})$. Like the sorted-neighborhood method for record linkage [23], GACHA examines nearby rows of the sorted

order, but rather than using a fixed neighborhood, the search budget of a representative r depends on the size of its component $C_t(r)$ at the start of pass t: $\kappa_t(r) = \max\{1, \lfloor X/|C_t(r)| \rfloor\}$, where X is a comparison-budget constant, with default $X = \text{round}(\lambda n/m^\star)$, with $\lambda = 10$, ten times the target-derived mean cluster size, when a target cluster count $m^\star$ is available. Smaller, potentially incomplete components thus receive wider neighborhoods, while larger components are limited and every component keeps a budget of at least one.

Candidates must also share a prefix of their packed strobe fields. Writing $\text{pref}_P(k)$ for the first P fields of key k, excluding the tie-breaker, the candidate set for pass t is $\mathcal{P}^{(t)} = \big\{(v_{(i)}, v_{(j)}) : 1 \le i < j \le M_t,\ \text{pref}_P(k_{(i)}) = \text{pref}_P(k_{(j)}),\ j - i \le \max\{\kappa_t(v_{(i)}), \kappa_t(v_{(j)})\}\big\}$, with $P = 1$ by default. The symmetric rule permits a comparison whenever either endpoint's budget covers the pair, so small components can match nearby larger ones, and since $\kappa_t(r) \le X$, each pass examines at most $M_t X$ pairs before prefix filtering. Candidates are enumerated directly from the sorted order without materializing $\mathcal{P}^{(t)}$.

3.5 Cascaded Candidate Verification

Since computing exact edit-distance is expensive, GACHA verifies candidate pairs using blocked k-mer signatures first, and only invokes exact edit-distance computation for pairs the signatures do not reject; pairs already in the same component are skipped.

Following Rashtchian et al. [5], each read is divided into consecutive blocks of about 22 bases, and each block is represented by a 64-bit presence mask $\sigma_b(r) \in \{0,1\}^{64}$ with one bit per 3-mer, which retains coarse positional information and supports efficient bitwise comparison. The signature distance between reads r_i and r_j is $S(r_i, r_j) = \sum_{b=1}^{B} \text{popcount}(\sigma_b(r_i) \oplus \sigma_b(r_j))$, the Hamming distance between their concatenated signatures. A pair is rejected outright when its signature distance reaches a calibrated cutoff S_{rej}; every other pair is verified by an exact edit distance:

$$\text{Verify}(r_i, r_j) = \mathbb{I}\big[S(r_i, r_j) < S_{\text{rej}}\big]\ \mathbb{I}\big[d_E(r_i, r_j) \le T\big], \tag{1}$$

where d_E is Levenshtein distance and T is the edit-distance threshold.

Exact verification uses Myers' bit-vector algorithm [24], rejecting immediately pairs whose length difference exceeds T and terminating early once a lower bound on the distance exceeds T. The signature cutoff may reject valid same-source pairs, reducing the probability of discovering useful connections. Repeated randomization can compensate by exposing alternative pairs between unresolved components, although recovery is not guaranteed if all such connections fail verification (Proposition 1). Unlike the two-threshold signature test of Rashtchian et al. [5], we deliberately do not accept pairs from signatures alone, since such a merge is permanent and its errors accumulate across passes (Section 4.4). Accepted pairs trigger concurrent union–find updates using atomic compare-and-swap.

3.6 Automatic Calibration

GACHA supports user-specified thresholds or automatically estimates the edit-distance threshold T and signature rejection cutoff S_{rej} without ground-truth labels. Calibration uses a pseudo-pass with the same randomized candidate-generation procedure as clustering, collecting up to 400,000 candidate pairs supplemented by uniformly sampled random pairs. Both edit and signature distances are computed for each pair. The threshold T is selected from the valley separating the two modes of the candidate-pair edit-distance distribution, with random pairs providing a reference for between-source distances. The rejection cutoff S_{rej} is then derived from the signature distances of pairs on either side of T, filtering unlikely matches before exact verification.

3.7 Termination

Clustering stops when the optional target cluster count $m^\star$ is reached, the maximum number of passes $P_{\max}$ is exhausted, or p consecutive passes produce no successful unions; setting $p = 0$ disables the last condition. With $p = 100$, a pair whose per-pass merge probability exceeds 0.046 survives the stop with probability below 0.01 (Section 4.4), so the patience rule alone can end a run that is not stopped at the target count (Section 6.5). The target count is still needed to set the neighborhood width $X = \lambda n/m^\star$, unless the width is given directly. An optional tolerance θ stops the run once the cluster count is at most $(1 + \theta)m^\star$. We

use $\theta = 0$ except in the 1 GB experiment ($\theta = 10^{-4}$). As shown in Section 4.1, stopping at this tolerance guarantees Accuracy$(0.95) \geq 1 - \theta$ provided that no false merges occur. The final clusters are the connected components maintained by union–find.

4 Analysis

A read of source s_k passes through an insertion–deletion–substitution channel with per-base error rate ε, and $e_r = d_E(r, s_k)$. All randomness of pass t (representatives, strobe locations, jitter, hashes, tie bits) comes from a fresh seed. Recall that K is the number of nonempty sources. Write $\mathcal{F}_t$ for the history up to pass t.

4.1 False merges are costly

Recall that representatives are the $M_t = \alpha m_t$ reads sampled in pass t (one per component for $\alpha = 1$), and they are sorted by their packed key. The *rank gap* between two representatives is the number of positions that separate them in this sorted order. A pair is only examined if its rank gap is at most the neighborhood width $\kappa_t(C) = \max\{1, \lfloor X/|C| \rfloor\}$ of either component, where $X = \lambda n/m^\star$ is the base width (equal to $\lambda n/K$ when the target is exact) (Section 4.4). A pair is *merged directly* in a pass if it is examined, passes the signature test, and has $d_E \leq T$.

After pass t the union–find partition is the set of connected components of the accepted pairs, whatever the order in which GPU threads apply unions. If every accepted pair has $d_E \leq T$ and no two reads of different sources are within T, every output cluster lies inside one true cluster, and every error is a *split*: a source whose accepted pairs do not connect all its reads.

Proposition 1 (Fresh randomization and irreversible merges)**.** *Conditional on the current partition and fixed calibration parameters, each pass draws fresh randomness for representative sampling and candidate generation. Consequently, previously missed relationships may be discovered in subsequent passes, whereas accepted merges are irreversible.*

Proof: Each pass independently samples representatives and constructs randomized similarity orderings based on the current partition. Thus, candidate-discovery opportunities are renewed, although their probabilities depend on the current components. However, union–find only merges components and never splits them, so an incorrect merge cannot be reversed.

A decision that can only *lose* an edge may be aggressive, because it can be re-offered on the next pass, but on the other hand, a decision that can *add* an edge must be conservative.

When no false merges occur, every predicted cluster is contained within a true source cluster. If the output contains at most $(1 + \theta)m^\star$ clusters with $m^\star = K$, there are at most θK excess fragments and consequently at most θK fragmented sources. Therefore, Accuracy$(0.95) \geq 1 - \theta$ under the no-false-merge assumption. This conditional bound motivates early stopping near the target cluster count, avoiding the potentially expensive recovery of the final fragments. Since the long tail of difficult-to-cluster reads dominates clustering time, this stopping rule can be used to greatly reduce clustering time while still achieving nearly the same accuracy. (e.g., in the 1 GB experiment, we set $\theta = 10^{-4}$)

4.2 Probability of Pair Being Missed

Consider two representatives x, y from distinct fragments A, B of the same source among M_t representatives. Let S_x, S_y be the sets of distinct q-mers in their corresponding strobe windows, with Jaccard similarity $J_s = |S_x \cap S_y|/|S_x \cup S_y|$. Each key field f_s is a b-bit hash of the minimum-hash q-mer in its window.

Lemma 1 (Field agreement, cf. min-wise hashing [22])**.** *Under independent uniform hashing,* $\Pr[f_s(x) = f_s(y) \mid S_x, S_y] = J_s + (1 - J_s)2^{-b}$.

Proof: The minimum-hash q-mer belongs to $S_x \cap S_y$ with probability J_s; otherwise, the independently hashed fields agree with probability 2^{-b}.

Thus, field agreement depends on the overlap between the windows' q-mer sets. Let $D(x, y)$ denote the number of matching leading key fields, with $a_j = \Pr[D \geq j]$ for same-source pairs and $\rho_j = \Pr[D \geq j]$ for different-source pairs. We estimate these probabilities empirically to account for overlapping strobe windows.

Proposition 2 (Single-pass exposure)**.** *Consider representatives x, y of two unresolved same-source fragments among M_t representatives. Let $D(x, y)$ denote their number of matching leading key fields, and let N_j count the other representatives sharing the first j fields with x.*

Define $\mu_j = \mathbb{E}[N_j \mid D(x, y) = j]$. Let $\kappa = \max\{\kappa_t(A), \kappa_t(B)\}$, and let π_T lower-bound the conditional probability of acceptance whenever (x, y) is examined. Then

$$\sigma_t \geq \pi_T \sum_{j=1}^{H} \Pr[D = j] \left(1 - \frac{\mu_j}{\kappa}\right)^+.$$

The bound depends on the number of representatives sharing sufficiently long key prefixes. In the approximation where unrelated representatives dominate this count and $\rho_j \approx \rho^j$, the prefix length required to control interference grows approximately as $\ln(M/\kappa)/\ln(1/\rho)$. This motivates the heuristic key-capacity estimate $M_t \lesssim \kappa/(2\rho_H)$. For $q = 3, W = 20$, the estimated capacity is 2×10^7 representatives for random references but only 5×10^5 for Microsoft Illumina. Increasing to $q = 4, W = 8$ raises these estimates to 2×10^{13} and 1.3×10^{11}, respectively.

4.3 Justification of pass re-randomization

Theorem 1 (Geometric contraction)**.** *Assume that no false merges occur. Fix a source with c reads, and let X_t denote its number of fragments after pass t, minus one.*

Suppose that, conditional on the clustering history, each unresolved fragment is incident to an accepted same-source merge in the next pass with probability at least $\sigma > 0$. Then $\mathbb{E}[X_{t+1} \mid \mathcal{F}_t] \leq (1 - \sigma/2)X_t$, and $\Pr[$source remains split after t passes$] \leq (c - 1)(1 - \sigma/2)^t$. If the same assumptions hold for all K sources, all are recovered after $t \geq \frac{2}{\sigma} \ln \frac{n-K}{\delta}$ passes with probability at least $1 - \delta$.

Proof: We bound the expected number of merges in one pass through the exposure graph of the fragments, and then apply the tower rule and a union bound over sources. (See supplementary for full proof.)

Theorem 1 requires a positive lower bound on the conditional merge probability in every pass. Fresh randomization renews candidate-discovery opportunities, but does not by itself establish this bound: exposure depends on the evolving partition, representative sampling, and verification thresholds. Under the stated assumption, these repeated opportunities yield geometric contraction. In contrast, reusing the same random choices can cause clustering to stagnate:

Proposition 3 (Fixed-seed stagnation)**.** *Suppose that every pass reuses the same random choices and that the pass update is deterministic given the current partition. If a pass produces no successful union, all subsequent passes leave the partition unchanged.*

Proof: With fixed random choices, representative selection, key construction, neighborhood search, and verification depend only on the current partition. A pass without a successful union leaves that partition unchanged, so every subsequent pass repeats the same computation.

Repeated randomization provides complementary opportunities for candidate discovery without requiring a proportionally wider neighborhood in every pass. The ablation in Section 6 confirms that fixed randomization can cause clustering to stagnate.

4.4 Budget, Stopping, and Verification

With $X = \lambda n/m^\star$, the number of candidate pairs in pass t satisfies $Q_t \leq 2\alpha \sum_{C \in \mathcal{C}^{(t)}} \kappa_t(C) \leq 2\alpha X m_t$. Thus, the comparison budget scales with the target-derived mean coverage, while size-adaptive neighborhoods prioritize smaller, unresolved clusters. By Proposition 1, a pair with conditional per-pass merge probability at least σ remains unmerged after p passes with probability at most $e^{-\sigma p}$. For $p = 100$, this probability is below 0.01 when $\sigma \geq 0.046$, motivating the patience-based stopping rule.

Verification must distinguish between missed and incorrect matches. Because blocked k-mer signatures discard the order within each block, their distances cannot reliably bound edit distance d_E. Rejecting a valid pair may delay recovery, as alternative connections can be discovered in later passes. In contrast, an incorrectly accepted pair permanently merges distinct sources, and such errors accumulate across passes.

Two-threshold signature tests [5] permit direct acceptance based on signature similarity, but this shortcut saves only 11% of runtime while reducing accuracy on Stanford Nanopore. GACHA therefore uses signatures only for rejection and verifies every accepted merge with exact edit distance. The rejection filter alone provides a 23× speedup over performing edit-distance verification on every candidate.

The edit-distance threshold T must balance within-source connectivity against false merges. Let $e_{\max}$ be the maximum read error and $D_{\min}$ the minimum distance between distinct source strands. By the triangle inequality, within-source distances are at most $2e_{\max}$, while cross-source distances are at least $D_{\min} - 2e_{\max}$. Thus, when $D_{\min} > 4e_{\max}$, any $T \in (2e_{\max}, D_{\min} - 2e_{\max})$ separates the two classes. GACHA estimates this threshold without labels by locating the valley between the modes of the candidate-pair edit-distance histogram. Unlike Otsu's mass-based criterion [25], which fails on one of our 15 datasets, this approach targets the separation between likely within-source and cross-source pairs.

5 Computational Complexity and GPU Implementation

5.1 Candidate Complexity and Computational Cost

Let m_t denote the number of clusters in pass t, with $M_t = \alpha m_t$ sampled representatives and Q_t candidate pairs. Since each representative examines at most X positions in the sorted order, $Q_t \leq M_t X$. As clusters merge, m_t decreases, progressively reducing the comparison workload. Over τ passes, the total number of candidate pairs is at most $\alpha X n \tau$, with at most $n - 1$ successful unions overall. Each pass requires key construction and radix sorting for M_t representatives, signature comparisons for Q_t candidates, and Myers' edit-distance verification [24] for the $F_t \leq Q_t$ pairs that pass signature filtering. The number of passes depends on the conditional convergence guarantee of Theorem 1.

5.2 Parallelization and Memory Efficiency

GACHA stores DNA at two bits per nucleotide and keeps union–find state on the GPU; sampling, key generation, signatures, neighborhood enumeration and verification run in parallel, keys are sorted with CUB radix sort [26], and accepted matches trigger atomic compare-and-swap unions.

To limit GPU memory use, GACHA supports full, lean (default), bucketed, and host-sequence modes, which trade memory consumption against additional processing and data movement while using the same clustering rules. The default lean mode processes globally sorted representatives in bounded groups, constructing signatures only for each group and its overlapping neighborhoods.

Complete proofs, additional probabilistic bounds, calibration details, and implementation-level complexity and memory analyses are provided in the Supplementary Material.

6 Experimental Evaluation

We evaluate GACHA on real Illumina and Nanopore DNA storage datasets and synthetic workloads containing up to 370 million reads. We assess clustering accuracy, scalability, and runtime, and use ablation studies to isolate the contributions of the main algorithmic components.

6.1 Datasets, Baselines, and Evaluation Metrics

Datasets. The four real datasets are Microsoft Illumina [3], Stanford Nanopore [27], Trellis-BMA [28] and BlockSem Illumina [29, 4]. The robustness and validation studies use 15 datasets: these four, BlockSem Nanopore [29], four DNAformer Illumina and Nanopore datasets [30], and six synthetic sets of up to 100 MB. The real datasets cover Illumina and Nanopore sequencing, with mean normalized edit-distance error rates ranging from 0.36% to 9.43%, and vary in the number of source strands and sequencing coverage.

For scalability experiments, we generate synthetic workloads of 1 MB to 1 GB with 120-nucleotide source strands, independent substitution, insertion, and deletion probabilities of 0.013 each, and shifted negative-binomial coverage with mean 10 and minimum one, motivated by the overdispersion observed in real sequencing data [31]. The largest dataset contains approximately 37 million source strands and 370 million reads.

Baselines and experimental setup. We compare GACHA against five DNA-storage-specific clustering methods: q-grams (the algorithm of Rashtchian et al. [5]), Clover [7], locality-sensitive hashing (LSH) [6], GradHC [16], and HSFC [9], and against the general-purpose methods UCLUST [12], MeShClust [14], and Starcode [15]. We use the released baseline implementations with their documented presets, except where a different published configuration is explicitly specified. Ground-truth labels are not used to select settings for the main comparisons. GACHA uses one fixed configuration on every dataset. All methods receive the same preprocessed reads, ground-truth assignments are not used for tuning, and the known source-strand count is provided when supported.

The baselines ran on an AMD EPYC 9355 server with 1 TB RAM, and GACHA on one NVIDIA H200 NVL GPU (141 GB, driver 580.178) in a separate node with two AMD EPYC 9355 CPUs and 1 TB RAM, compiled with CUDA 12.9 at `-O3`. GACHA results are medians over three seeds, with the minimum over seeds given where it differs. Over ten seeds the medians on the real datasets are unchanged except on Trellis-BMA, where the median is 0.9703 (0.827 to 0.996). GACHA's time starts once the reads are packed and copied to the GPU and includes calibration; it excludes CUDA initialization, file loading, output serialization and evaluation, as do the baselines' wrapper times, and runs exceeding resource limits are reported as incomplete.

Evaluation metrics. Our primary metric is Accuracy(0.95) [5], the fraction of true source clusters for which a pure predicted cluster contains at least 95% of the source reads.

Additional dataset statistics, baseline configurations, evaluation metrics, memory measurements, and detailed ablation, calibration, and robustness results are provided in the Supplementary Material.

6.2 Clustering Accuracy and Runtime on Real Datasets

Table 1 compares GACHA with existing methods on four real DNA-storage datasets. GACHA achieves median Accuracy(0.95) $\geq$ 0.9786 on all four, reaching 1.0000 on three. On Trellis-BMA, accuracy falls to 0.827 in the worst seed because every source has another within 26–30 edits, preventing a single threshold from separating all sources (Section 4.4). A label-derived threshold, set halfway between the 99.9th percentile of same-source and the 0.01st percentile of different-source read distances using the ground truth, raises accuracy to 0.9977. GACHA is also the fastest method on all four datasets, from 5.7× faster than UCLUST on BlockSem Illumina (2.14 s against 12.25 s) to 108× faster than q-grams on Microsoft Illumina (1.24 s against 134.26 s). On Trellis-BMA, q-grams is more accurate (0.9939 against 0.9786) but ten times slower (2.00 s against 0.20 s). Across the four datasets, this is a 14-fold geometric-mean speedup over the fastest completed competitor on each dataset, although GACHA runs on a GPU and the baselines on CPUs.

Several baselines score close to zero on the Nanopore datasets. In almost every case the cause is over-fragmentation under a strict metric rather than a completely failed run: Accuracy(0.95) gives no credit to a source split into several pure clusters, whereas the purity-style metrics used in some of the baseline papers do not penalize splitting. We summarize the mechanism for each method, discuss their performance, and provide the supporting measurements in the Supplementary Material.

6.3 Scalability to Archive-Scale Workloads

We measure scalability on synthetic workloads from 1 MB to 1 GB with fixed strand length, coverage, and error parameters, and compare GACHA with q-grams, the fastest completed DNA-storage-specific baseline at 1 GB, and UCLUST (Figure 1a). At 1 GB, GACHA clusters approximately 370 million reads in 43.9 s (median of three runs), compared with 33,467.53 s for q-grams, a 762× speedup, while UCLUST does not finish within 48 hours. This runtime uses a stopping tolerance of 0.01% relative to the target cluster count. The measured Accuracy(0.95) is 0.99990, consistent with the conditional bound of Section 4.1 when no false merges occur. A full run, stopped only by the patience rule, takes 218 s and leaves 3 of the 37 million sources split. Timings exclude input loading and output serialization.

6.4 Ablation Studies

We evaluate the contributions of randomized candidate discovery, adaptive agglomeration, and cascaded verification on the 100 MB negative-binomial benchmark and six additional real and synthetic datasets.

	Microsoft		Trellis-BMA		BlockSem		Stanford	
Method	Acc.	Time (s)	Acc.	Time (s)	Acc.	Time (s)	Acc.	Time (s)
GACHA	**1.0000**	**1.24**	0.9786	**0.20**	**1.0000**	**2.14**	**1.0000**	**0.77**
Q-grams	0.9856	134.26	**0.9939**	2.00	1.0000	46.45	0.9836	4.92
GradHC	0.0199	16517.07	0.0001	126.17	1.0000	3159.42	0.9929	390.00
Clover	0.0244	237.14	0.1524	4.29	0.1506	27.65	0.0024	12.33
LSH	0.1570	21916.40	0.0264	163.77	0.9565	2580.36	0.0034	494.61
UCLUST	0.9999	16533.57	0.0007	5.03	1.0000	12.25	0.0359	30.58
MeShClust	0.0000	1924.69	0.0018	271.92	0.0872	3322.73	0.0009	499.43
HSFC	0.0000	40015.48	0.5675	119.91	>12 hrs		0.0123	236.55
Starcode	>12 hrs		0.0199	71.57	1.0000	83.23	0.0005	659.09

Table 1: Clustering quality and runtime on four real DNA storage datasets. Acc. denotes Accuracy(0.95) and time is clustering-only seconds. Higher accuracy and lower runtime are better. "12+ hrs" denotes a run exceeding a twelve-hour limit, with unavailable accuracy. CD-HIT [11] and DNACLUST [13] were also run but are omitted here for space. Baselines were run on a separate 16-thread CPU host with each method's recommended or best-found preset; UCLUST and GradHC were run single-threaded on Microsoft. For GACHA, we take the median over three seeds; the minimum accuracy for Trellis-BMA over seeds is 0.827.

Randomized candidate discovery: Fresh randomization across passes is essential for convergence. On the 100 MB benchmark, fixing the random choices reduces Accuracy(0.95) from 1.0000 to 0.0174, while a single pass achieves only 0.0162. The fixed-seed run does not perform any new unions beyond the 8th pass (hence ends automatically at 107 passes) (Figure 1b), consistent with Proposition 3. Across six datasets, fixed randomization similarly stalls after 104–108 passes, with Accuracy(0.95) between 0.003 and 0.21. Resampling representatives also improves efficiency: using only component roots preserves accuracy within 0.001 but requires 1.6–6 times more passes and increases runtime by 1.5× in geometric mean.

Adaptive agglomeration: Replacing the size-adaptive neighborhood with a fixed width matched to its mean comparison budget preserves accuracy under fresh randomization but increases runtime by 1.13× in geometric mean. The effect varies by dataset: fixed widths are faster on high-coverage Nanopore data (0.62–0.73× GACHA's runtime) but 1.3–2.3× slower on Microsoft Illumina and synthetic workloads. Under matched verification budgets, conventional alternatives are less effective. Sorting all reads with fresh keys achieves Accuracy(0.95) of 0.9995–0.9997 at width 8 but runs 1.2–3.0× slower; widening the neighborhood to 32 spends the same budget in about four times fewer passes (13–32 instead of 54–145), which shortens runtime but reduces accuracy to 0.980–0.9993, as fewer passes leave fewer fresh draws to resolve the remaining splits. Uniformly random candidate selection achieves only 0.0005–0.006 accuracy. These results show that representative sampling and size-adaptive neighborhoods concentrate comparisons on unresolved clusters.

Cascaded verification: Allowing direct acceptance based on signature similarity saves 11% of runtime but risks irreversible false merges. On Stanford Nanopore with the label-derived threshold, this shortcut reduces Accuracy(0.95) from 0.9997–1.0000 to 0.983–0.9996, and to 0.973–0.998 when the run is not stopped at the target count, as false accepts keep accumulating. GACHA therefore verifies every accepted merge using exact edit distance. Conversely, removing the signature rejection filter increases runtime on the 100 MB benchmark from 4.90 s to 114.4 s (23×), demonstrating its importance for efficient verification.

6.5 Robustness and Parameter Sensitivity

Target cluster count. Although GACHA can run without a target cluster count, the target determines the neighborhood width $X = \lambda n/m^\star$. We disabled the stop at the target count on all 15 datasets, and on eight of them also misestimated the target by factors of 0.25–4. Accuracy(0.95) stays within 0.001 of the true-target result except on Trellis-BMA, at an overall geometric-mean slowdown of 1.44× without the stop and 1.7–3.5× with a misestimated target. Trellis-BMA is notably sensitive: accuracy falls from 0.979 to 0.478 without the stop, because its calibrated threshold lies above the 28-edit spacing of its sources, so

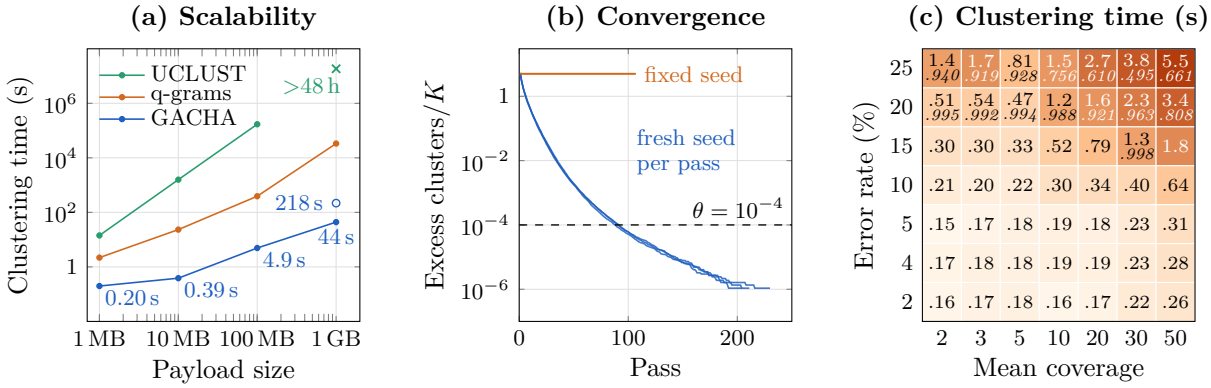


Figure 1: (a) Clustering time as the synthetic payload grows from 1 MB to 1 GB, for GACHA, q-grams (9.3 h at 1 GB) and UCLUST. The filled GACHA point at 1 GB stops once the cluster count is within 0.01% of the true count; the open point is the full run, stopped by the patience rule. UCLUST did not finish within 48 hours. (b) Excess clusters, $(\text{clusters} - K)/K$, after each pass on the 100 MB benchmark, for three runs with a fresh seed in every pass and three with one fixed seed; the dashed line is the 0.01% stopping level. (c) GACHA clustering time in seconds on 10,000 synthetic strands of 120 nucleotides, by mean coverage and total error rate (median of three seeds); where median Accuracy(0.95) is below 0.999 it is printed in italics under the time.

occasional false unions keep the patience counter from ending the run. Its accuracy also falls to 0.015–0.159 with misestimated targets; providing the target count is therefore advisable for challenging datasets and is generally feasible in DNA storage, where the number of designed strands is known.

Coverage and sequencing errors. We evaluate 10,000 synthetic strands at mean coverages of 2–50 and total error rates of 2–25% (Figure 1c). Median Accuracy(0.95) remains at least 0.998 for error rates up to 15%, but falls to 0.81–0.995 at 20% and 0.49–0.94 at 25%. At high error rates, overlapping within-source and between-source edit-distance distributions lead to false merges, which become more frequent as coverage increases. Subsampling real datasets to coverages of 2–10 preserves perfect accuracy on Microsoft, BlockSem, and Stanford, while Trellis-BMA achieves 0.978–0.999.

Threshold calibration. The valley-based threshold achieves accuracy within 0.0002 of both the label-derived threshold and Otsu's criterion [25] on 12 of 15 datasets. However, Otsu's criterion fails on the 10 MB synthetic dataset with 10% error and reduces accuracy by 0.139 on Trellis-BMA. Signature parameters were selected using calibration candidates from four datasets.

7 Conclusion

We presented GACHA, a scalable clustering algorithm for noisy DNA storage reads that combines independently randomized neighborhood search, size-adaptive agglomeration, and cascaded verification. Our analysis establishes geometric convergence under a positive lower bound on fragment-merge probability, while experiments demonstrate accurate clustering at archive scale, including 370 million reads in 44 seconds. Nevertheless, performance depends on reliable candidate discovery and distance thresholds, particularly when distinct source strands are highly similar, as observed on Trellis-BMA. Overall, GACHA demonstrates that reducing unnecessary comparisons, rather than merely accelerating them, is key to scalable DNA storage decoding.

References

[1] Church, G. M., Gao, Y. & Kosuri, S. Next-generation digital information storage in dna. *Science* **337**, 1628–1628 (2012).

[2] Ceze, L., Nivala, J. & Strauss, K. Molecular digital data storage using dna. *Nature Reviews Genetics* **20**, 456–466 (2019).

[3] Organick, L. *et al.* Random access in large-scale dna data storage. *Nature biotechnology* **36**, 242–248 (2018).

[4] Sharma, P. *et al.* Dna storage toolkit: A modular end-to-end dna data storage codec and simulator. In *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 144–155 (IEEE, 2024).

[5] Rashtchian, C. *et al.* Clustering billions of reads for dna data storage. *Advances in Neural Information Processing Systems* **30** (2017).

[6] Antkowiak, P. L. *et al.* Low cost dna data storage using photolithographic synthesis and advanced information reconstruction and error correction. *Nature communications* **11**, 5345 (2020).

[7] Qu, G., Yan, Z. & Wu, H. Clover: tree structure-based efficient dna clustering for dna-based data storage. *Briefings in Bioinformatics* **23**, bbac336 (2022). URL `https://doi.org/10.1093/bib/bbac336`. `https://academic.oup.com/bib/article-pdf/23/5/bbac336/45937680/bbac336.pdf`.

[8] Wang, P. *et al.* Duhi: Dynamically updated hash index clustering method for dna storage. *Computers in Biology and Medicine* **164**, 107244 (2023).

[9] Shao, Q. *et al.* Dna sequence clustering in high error rates via hash sketches fuzzy clustering for efficient stored data reconstruction. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 142–154 (Springer, 2025).

[10] Tarjan, R. E. Efficiency of a good but not linear set union algorithm. *Journal of the ACM* **22**, 215–225 (1975).

[11] Li, W. & Godzik, A. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics* **22**, 1658–1659 (2006). URL `https://doi.org/10.1093/bioinformatics/btl158`. `https://academic.oup.com/bioinformatics/article-pdf/22/13/1658/48838763/bioinformatics_22_13_1658.pdf`.

[12] Edgar, R. C. Search and clustering orders of magnitude faster than blast. *Bioinformatics* **26**, 2460–2461 (2010). URL `https://doi.org/10.1093/bioinformatics/btq461`. `https://academic.oup.com/bioinformatics/article-pdf/26/19/2460/48857155/bioinformatics_26_19_2460.pdf`.

[13] Ghodsi, M., Liu, B. & Pop, M. Dnaclust: accurate and efficient clustering of phylogenetic marker genes. *BMC bioinformatics* **12**, 271 (2011).

[14] James, B. T., Luczak, B. B. & Girgis, H. Z. Meshclust: an intelligent tool for clustering dna sequences. *Nucleic Acids Research* **46**, e83–e83 (2018). URL `https://doi.org/10.1093/nar/gky315`. `https://academic.oup.com/nar/article-pdf/46/14/e83/25509702/gky315.pdf`.

[15] Zorita, E., Cuscó, P. & Filion, G. J. Starcode: sequence clustering based on all-pairs search. *Bioinformatics* **31**, 1913–1919 (2015). URL `https://doi.org/10.1093/bioinformatics/btv053`. `https://academic.oup.com/bioinformatics/article-pdf/31/12/1913/49014295/bioinformatics_31_12_1913.pdf`.

[16] Ben Shabat, D., Hadad, A., Boruchovsky, A. & Yaakobi, E. Gradhc: highly reliable gradual hash-based clustering for dna storage systems. *Bioinformatics* **40**, btae274 (2024).

[17] Guo, A. J., Liang, C. & Hou, Q.-H. Deep squared euclidean approximation to the levenshtein distance for dna storage. In *International Conference on Machine Learning*, 8095–8108 (PMLR, 2022).

[18] Wei, X., Guo, A. J., Sun, S., Wei, M. & Yu, W. Levenshtein distance embedding with poisson regression for dna storage. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, 15796–15804 (2024).

[19] Abraham, H. *et al.* Beyond the alphabet: Deep signal embedding for enhanced dna clustering. *arXiv preprint arXiv:2410.06188* (2024).

[20] Sahlin, K. Effective sequence similarity detection with strobemers. *Genome research* **31**, 2080–2094 (2021).

[21] Roberts, M., Hayes, W., Hunt, B. R., Mount, S. M. & Yorke, J. A. Reducing storage requirements for biological sequence comparison. *Bioinformatics* **20**, 3363–3369 (2004).

[22] Broder, A. Z. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997*, 21–29 (IEEE, 1997).

[23] Hernández, M. A. & Stolfo, S. J. The merge/purge problem for large databases. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, 127–138 (1995).

[24] Myers, G. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM (JACM)* **46**, 395–415 (1999).

[25] Otsu, N. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics* **9**, 62–66 (1979).

[26] Merrill, D. & Grimshaw, A. High performance and scalable radix sorting: a case study of implementing dynamic parallelism for GPU computing. *Parallel Processing Letters* **21**, 245–272 (2011).

[27] Chandak, S. *et al.* Overcoming high nanopore basecaller error rates for dna storage via basecaller-decoder integration and convolutional codes. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 8822–8826 (2020).

[28] Srinivasavaradhan, S. R., Gopi, S., Pfister, H. D. & Yekhanin, S. Trellis bma: Coded trace reconstruction on ids channels for dna storage. In *2021 IEEE International Symposium on Information Theory (ISIT)*, 2453–2458 (IEEE, 2021).

[29] Sharma, P., Lim, C.-K., Lin, D., Pote, Y. & Jevdjic, D. Efficiently enabling block semantics and data updates in dna storage. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 555–568 (2023).

[30] Bar-Lev, D., Orr, I., Sabary, O., Etzion, T. & Yaakobi, E. Scalable and robust dna-based storage via coding theory and deep learning. *Nature Machine Intelligence* **7**, 639–649 (2025).

[31] Heckel, R., Mikutis, G. & Grass, R. N. A characterization of the DNA data storage channel. *Scientific Reports* **9**, 9663 (2019).