FlashAlign: Faster Long-Read Alignment with Better Chaining

Gong Chen[1,*] and Ian Korf[1,2,*]

[1]Genome Center, University of California, Davis, CA 95616, USA

[2]Department of Molecular and Cellular Biology, University of California, Davis, CA 95616, USA

[*]Corresponding authors: dsbchen@ucdavis.edu, ifkorf@ucdavis.edu

Preprint, October 21, 2026

Motivation: Nearly every long-read aligner follows the seed–chain–align recipe of minimap2: seed matches (anchors) between read and reference are collected from an index, anchors are chained colinearly by a dynamic programme, and base-level alignment after chaining. The chaining complexity is $O(hN)$ for N anchors with a look-back of h predecessors. For accurate long reads the anchor pool N is too large to chain effectively. For noisy long reads, the chaining is hurt by noisier anchors that create more look-back during backtracing. For a further step in high-throughput long-read aligner, it is required to make a better chaining.

Results: FlashAlign is an aligner for long genomic and long spliced RNA reads that improves chaining by introducing a new stage before such a recipe. It combines modified minimap and GraphMap diagonal voting for placement first before chaining to reduce chaining workload. It replaces minimizers with closed syncmers to skip reverse-strand searching. It also optimises colinear chaining and base alignment for throughput. FlashAlign reaches around twice the throughput of minimap2 on long genomic reads. On long spliced reads, FlashAlign reaches better accuracy while achieving slightly faster throughput than minimap2.

Availability and implementation: FlashAlign is implemented in C++17 with Python bindings and is available under the MIT licence at https://github.com/97YearsOldProgrammer/flashalign.

Contact: dsbchen@ucdavis.edu, ifkorf@ucdavis.edu

1 Introduction

With innovation of long-read sequencing technology developed from Pacific Biosciences (PacBio) or Oxford Nanopore Technologies (ONT), sequencing reads are becoming more dominant for sequence analysis. Long reads genomic sequence is useful for structural variant finding whereas long spliced RNA reads contributes to more novel exon and junction finding. The dominant aligner for these data is minimap2 (Li, 2018), which like most full-genome aligners before and after it (Chaisson and Tesler, 2012; Jain et al., 2020, 2022; Li and Homer, 2026; Ren and Chaisson, 2021; Sedlazeck et al., 2018; Wang and Li, 2026) follows the seed–chain–align recipe. Minimizers (Roberts et al., 2004) are indexed in a hash table. The colinear chaining algorithm leap over the sorted anchor pool to align the read coarsely (Li, 2018). Then, the base-level alignment (Suzuki and Kasahara, 2018) not just take the role of filling the gap between anchors while also extending towards left and right of the anchor chain. Lots optimisation been applied to minimap2, for example weighted seeds in repeats (Jain et al., 2020, 2022), faster chaining (Chandra et al., 2026; Kalikar et al., 2022; Li, 2021), acceleration of the dynamic programming (DP) stage (Kalikar et al., 2022; Suzuki and Kasahara, 2018), or replacing C by Rust (Wang and Li, 2026). Most recently minibwa (Li and Homer, 2026), successor of BWA-MEM from the author of minimap2, replaces minimap2 seeding and reaches a higher throughput over long accurate genomic reads. We will explain how that new tool also related to our work next.

The existing chaining has a structural cost. It is hidden for short reads but grows with read length. The chaining complexity is $O(hN)$ for N anchors with a look-back of h predecessors. Number of anchor is related to the query length because anchor is each individual index hit that belongs to the whole posting of a seed. In long reads, the amount of seed that a query retains is much more than short reads therefore it is a much larger anchor set. The chaining stage carries two things: finding where the read maps, and returning its optimal scored structure. This means, if we can ablated the role of where the long query is before chaining when mapping long reads, then chaining complexity would scaled to only to the query length despite the reference size. FlashAlign resolved that by compression diagonal from raw anchor before chaining to find where the read is before chaining.

It is not novel that computing diagonal from raw anchors as a plain idea. Wilbur and Lipman's k-tuple diagonals (1983) and FASTA's (1988), Ning et al.'s shift-sorted hits (2001), Kent's diagonal clumps (2002), Rasmussen et al.'s q-gram filter (2006), and Liao et al.'s seed-and-vote (2013) all assign a query to the diagonal collecting the most seed hits before computing any alignment. The idea reached long reads twice. GraphMap (Sović et al., 2016) applied it as a Hough transform from image analysis. Every hit votes for its diagonal. A graph-based anchor search is run over the three distinct partitions of the read. The original minimap (Li, 2016), minimap2 predecessor, used a colinear clustered minimizer hit along the diagonal for clustering with a 500 bp band. Its successor replaced the clustering with the global colinear chaining DP described above (Li, 2018), the original diagonal voting cannot tolerate even short indel drift. The idea is abandoned before because to our knowledge to make it work: a complete aligner need keep diagonal survive the indel drift, sequencing errors, and structural variants in long reads.

Here we present FlashAlign: an aligner for PacBio HiFi and ONT genomic long reads, and for spliced long RNA reads from both platforms. FlashAlign does not map short reads, because query need to be long enough to have enough anchor to be compressed into a plausible placement before chaining. Table 1 shows our relationship to other versatile long reads aligners.

Table 1. Comparison of long-read aligners.

Stage	FlashAlign	minimap2	minibwa	Winnowmap2
Indexing	hash table	hash table	FM-index	hash table
Seeding	closed syncmers	minimizers	SMEMs	weighted minimizers
Placement	diagonal vote	–	–	–
Chaining	local	global	global	global
Alignment	ksw2	ksw2	ksw2	ksw2 + MCAS

2 Methods

FlashAlign adds one stage, the vote, to the seed–chain–align procedure. It uses closed syncmers (Edgar, 2021) in place of minimizers, for easier access to the reverse strand by arithmetic. Vote means that instead of taking all valid seeds of a query, it samples a subset of the seeds to find an optimal location. Subread applied it to short reads (Liao et al., 2013). Unlike Subread, which takes overlapping seeds to find the optimal location, FlashAlign samples seeds sparsely over the query and applies a colinear chain to check the colinearity of those candidates. FlashAlign also applies a query partition to decide which combination of candidate chains explains the query, before the local chaining. The benefit of the vote replacing the placement role of a global colinear chain is that chaining can focus on a localised pool to improve wall time. A localised pool is further optimised by colinear chaining over exact diagonal clusters of anchors, while repetitive regions are optimised by a 2-d tree. For spliced long reads, since each intron makes a reference jump of the diagonal, FlashAlign votes to compress exons into bins, then scores a possible placement of where the locus is first, then applies chaining and base-level DP.

2.1 Definitions

An anchor is one of the hits that belong to a seed of a query and reference as an exact k-mer match. It contains query position q and reference position r. The *diagonal* of that hit is $d = r - q$. Let a read of length L originate from the reference, starting at r_0. Assume the query is affected by substitutions and by insertions and deletions of total net length δ. Every anchor of the true alignment satisfies

$$|d - r_0| \;\le\; \delta, \tag{1}$$

The anchors of a read map towards one diagonal unless the query drifts off the reference. A rival homology of the true placement would have anchors that spread over as many diagonals. Instead of sampling seeds in an overlapping manner, FlashAlign takes seeds from *tiles*, the segments of equal length that a query is cut into. We found that for ordinary long reads one seed per tile, 128 per read, is sufficient to express the diagonal structure of a read.

2.2 Whole-read voting

In the vote, each sampled seed votes for the diagonal bins of its reference hits, and the bins with the most votes are the likely placements. Inside each tile, an occurrence gate is applied. The first seed that satisfies the gate becomes admitted, and each tile has only one valid seed admitted into the vote.

Each indel drifts the diagonal, by up to the δ of eq. (1), which makes it hard to count the sampled seeds into exact diagonal values. Instead of correcting the diagonal for the drift, FlashAlign introduces a diagonal width W. The width is similar to the diagonal band of colinear chaining: a range of diagonals falls into the same bin. In practice W grows with the query length, $W = \lfloor L/s \rfloor$ clamped to $[W_{\min}, W_{\max}]$, with $(W_{\min}, s) = (64, 128)$ for ONT and $(48, 64)$ for HiFi, and $W_{\max} = 2048$, which we found optimal even for ultra-long reads. A bin is a compressed result of multiple anchors' diagonals: every posting r of an admitted seed at query position q goes into the bin $\lfloor (r-q)/W \rfloor$ of its contig (Algorithm 1, lines 6–7).

Algorithm 1. Whole-read diagonal vote for one strand

Input $\mathcal{S}$ the admitted seeds; W bin width; m the minimum support of a bin
Output P the ranked peaks

```
1  function DiagonalVote(S, W, m)
2      B ← ∅; e ← 0                                  ▷ bin map (support, epoch); epoch counter
3      foreach seed (q, postings) ∈ S do
4          e ← e + 1                                 ▷ one epoch per distinct query seed
5          foreach r ∈ postings do                   ▷ ascending
6              d ← (r − start) − q                   ▷ diagonal of the hit
7              κ ← pack(contig(r), ⌊d/W⌋)             ▷ bin key
8              if B[κ].epoch ≠ e then                ▷ new for this seed
9                  B[κ].support += 1; B[κ].epoch ← e
10     foreach bin (c, b) ∈ B do                     ▷ second compression
11         fold support of (c, b ± 1) into it        ▷ neighbour fold
12     drop bins with folded support < m             ▷ support floor
13     P ← empty list                                ▷ emitted peaks
14     foreach bin (c, b) ∈ B, by descending folded support do
15         if P has a peak (c, b′) with |b − b′| ≤ 1 then continue
16         append peak(c, median diagonal of bin b) to P
17     return P                                      ▷ ranked peaks
```

To further distinguish compressed diagonals from each other, the number of distinct seeds belonging to the compressed diagonal bin is counted as the number of supporters (lines 8–9). A histogram stores the one-dimensional diagonal space. In it, each bin is further summed with its nearby neighbours' support counts, as in lines 10–11. We found in practice that the fold helps the true placement weigh more compared to the noise. After the fold, bins with fewer than m supporters (3 for DNA, 2 for RNA) are dropped. In spirit, this is similar to minimap2 dropping a chain with fewer than n anchors as a valid chain candidate. The bins with the highest number of supporters are the most likely primary placements.

The survivors become peaks. A peak is the median diagonal of its bin and is discarded when it lies within one bin of a stronger peak. For spliced reads each peak is further compressed to show a possible placement locus (Section 2.6).

2.3 Candidate admission

After the voting there is a pool of peaks in the one-dimensional histogram. Let $v(p)$ be the number of seeds supporting a peak p in contig c and bin b,

$$v(p) \;=\; \big|\{\, q \;:\; \exists r,\;\; \mathrm{contig}(r) = c,\;\; \lfloor (r-q)/W \rfloor = b \,\}\big|, \tag{2}$$

The best peak is the one with the most seeds supporting it. Function Admit() in Algorithm 2 shows which peaks become candidates, from which the primary chain and its secondary chains (Section 2.4.3) can come. A peak with at least a quarter of the best peak's support is admitted outright (line 5). A weaker peak is dropped only when a peak more than four times stronger has its seeds in the same part of the read, overlapping by at least half of the shorter, as in minimap2's test for a secondary chain (lines 6–8); otherwise it is kept. The peak set after the admission filter is therefore returned to rank order (lines 9–10).

Algorithm 2. Candidate admission

Input $\mathcal{P}$ the ranked vote peaks; $v(p)$ the support of a peak, eq. (2); $M(p)$ the tiles whose admitted seed voted for p
Notation $p \sim p'$ when $M(p)$ and $M(p')$ overlap by at least half of the smaller
Constants $\rho = \frac{1}{4}$, $K = 16$

```
1  function Admit(P, ρ, K)
2      v_max ← max_{p∈P} v(p)                        ▷ support of the best peak
3      foreach p ∈ P, by descending v(p) do          ▷ strongest first
4          keep(p) ← true                            ▷ admitted by default
5          if v(p) ≥ ⌈ρ v_max⌉ then continue         ▷ near the best
6          foreach p′ ∈ P \ {p}, by descending v(p′) do
7              if v(p) ≥ ⌈ρ v(p′)⌉ then break        ▷ ratio test
8              if p ∼ p′ then keep(p) ← false; break ▷ rival test
9      C ← kept peaks in rank order, one per (contig, strand, bin)
10     return the first K of C on each strand        ▷ candidates
```

2.4 Local chaining

A chaining is global if the anchors it acquires require expansion of the seeds' postings. Its DP looks back over h predecessors for each of N anchors. Once a read's true placement is given by voting, FlashAlign chains inside the window of a selected candidate instead.

Algorithm 3. Gathering a candidate's runs

Input $\mathcal{F}$ the read's closed syncmers (q, z, P): orientation bit z, postings P sorted by r
s the candidate's strand
$[\ell, u)$ the window, eq. (3) or (4)
Output R the runs in reference order, with extents, diagonal d and weight w
Constants $N = 200$ (human)
Notation $\mathrm{LB}(P, v)$ is the first posting of P at or after v; a seed is rescued when the vote admitted it past N

```
1  function GatherRuns(F, s, ℓ, u, N)
2      A ← ∅                                              ▷ anchor pool
3      foreach seed (q, z, P) ∈ F do
4          if |P| > N and not rescued then skip the seed  ▷ occurrence gate
5          i ← LB(P, ℓ),  j ← LB(P, u)                    ▷ two binary searches
6          foreach posting (r, z_r) ∈ P[i, j) do          ▷ ℓ ≤ r < u
7              if z ⊕ z_r ≠ s then skip it                ▷ other strand
8              append anchor (q, r) to A                  ▷ exact k-mer match
9      sort A by (r, q) and drop repeated anchors         ▷ radix sort
10     R ← empty list,  O ← empty map                     ▷ open run of each diagonal
11     foreach anchor (q, r) ∈ A do                       ▷ by r, then q
12         d ← r − q,  ρ ← O[d]                           ▷ diagonal and its open run
13         if ρ exists and r ≤ r_e(ρ) then                ▷ k-mers overlap or touch
14             w_ρ ← w_ρ + r + k − r_e(ρ)                 ▷ newly matched bases
15             q_e(ρ) ← q + k,  r_e(ρ) ← r + k            ▷ extend
16         else append (q, q+k, r, r+k, d, k) to R as O[d]  ▷ new run
17     sort R by (r_b, q_b, q_e)                          ▷ reference order
18     return R                                           ▷ w = r_e − r_b for every run
```

Equation (3) shows how the diagonal of a candidate turns into where the read is:

$$\big[\, d^*,\ \ d^* + L \,\big). \tag{3}$$

Equation (3) assumes that the read lies on one diagonal. An indel shifts the diagonal. To model the diagonal drift, we fit a line to the winner's seeds, $d(q) = d_0 + \lambda q$, where d_0 is the diagonal at the start of the read and λ the drift; the read then covers $(1+\lambda)L$ bases of the reference. The line is fitted after the vote and before the chaining, and it is used when enough of the winner's seeds agree on it and its slope lies inside $[-6\%, +10\%]$. A fit pinned at either end of that range is refused, since a line fitted across the copies of a repetitive element has a slope of its own that says nothing about drift. When the line is used, the winner's window is widened to hold both eq. (3) and the reference span of the line:

$$\big[\, \min(d^*, d_0) - p,\ \ \max\big(d^* + L,\ d_0 + (1+\lambda)L\big) + p \,\big). \tag{4}$$

Both windows are padded by $p = 512\,\mathrm{bp}$ on each side so that the terminal anchors are covered.

The closed syncmers of the read are fetched there by a binary search over the sorted posting array. To further reduce chaining complexity, the anchors are grouped into runs, maximal sets of anchors on one diagonal ($\Delta d = 0$) whose k-mers overlap or touch, and only runs are chained. The chain uses minimap2's chaining recurrence and backtrace without its heuristic look-back limits (Li, 2018, Section 2.1), applied to runs instead of single anchors (Section 2.4.1).

2.4.1 *Chaining over runs*

Function GatherRuns() in Algorithm 3 shows how the anchor set of a candidate is collected before the colinear chain. The base element FlashAlign chains is a run, not a single anchor. A run is a maximal exact match on one diagonal, taken as the anchors give it, without greedy extension. We found that collapsing overlapping anchors of the same diagonal into runs before chaining does not hurt the accuracy of local chaining and increases the throughput.

FlashAlign chains over runs instead of anchors, applying minimap2's chaining recurrence (Li, 2018, Section 2.1). The same award of chain score whenever more base pairs are explained, and the same penalty for each change in diagonal as drift, are applied. Because each run is one diagonal value that a certain number of anchors belong to, a simple sorting algorithm can handle the relationship of the anchors within the run (lines 11–16).

Algorithm 4. Chaining a ladder: exact predecessor search over a tree of runs

Input R the pool sorted in reference order, with query end q_e, diagonal d and weight w
Output the chain score F and the predecessor of every run
Notation $\mathrm{score}(A, B) = F(A) + \mathrm{gain} - \mathrm{drift}$
Ceiling(n,B) is the best score stored in box n plus the most gain and minus the least drift any run in it could have with B

```
1  function ChainRuns(R)
2      build the tree: split R at the median, alternately by q_e and by d, a block
         of runs per leaf                                 ▷ once per pool
3      foreach B ∈ R do
4          best ← w_B;  pred ← ∅                          ▷ fresh start
5          Try(the last scored run on diagonal d_B)       ▷ incumbent, drift 0
6          foreach box n popped from a stack seeded with the root do
7              if Ceiling(n,B) < best, or = best and no run in n is after pred
                 then skip n
8              if n is a leaf then Try(each run A in it)
9              else push the worse child, then the better
10         F(B) ← best;  write F(B) and B's extents into every box on its
             path to the root                             ▷ activate
11 function Try(A)
12     if score(A, B) > best, or equal, pred ≠ ∅ and A after pred then best
         ← score(A, B);  pred ← A
```

Chaining over runs makes the complexity of chaining depend on how many interruptions there are in the read's exact-match structure. This helps FlashAlign to handle chaining in an exact manner instead of a heuristic approximation of the search depth. In practice, whenever a set of runs is below 2,048, we chain the pool by a linear scan back over it, because chaining over that is already cheap enough. Inside repetitive regions, where the run set blows up, we use an optimised 2-d tree search to keep exactness while reducing the searching complexity (Section 2.4.2). Neither keeps minimap2's heuristic limits on the iterations of the scan.

2.4.2 *Exact predecessor 2-d search*

Whenever the size of the run set blows up, instead of using a heuristic limit on the look-back to bound its cost, FlashAlign splits the whole set into a 2-d tree, alternately split by the query end and by diagonal at the median, until each leaf holds at most sixteen runs (Algorithm 4, line 2). Inside the tree, each node is called a box, it's a union of the set of nodes below it and its own set of runs. A box also keeps the best score among its runs scored so far and the range of their query ends and diagonals (line 10). When we want to score a run B, before opening the tree, we score the last scored run on B's own diagonal, where the drift is zero. So the score of that is the best score we have so far (lines 4–5).

Because each box can compute its ceiling score for B from the information it holds, the stored best score plus the most a run in it could gain minus the least drift it could pay, it does not require opening the box to tell whether it is worth exploring. From the root, we can know if opening the whole pool could give us a better predecessor by direct comparison of the root's ceiling with the best score so far. If that test passes, we open the box and test its children the same way, the better child comes first (line 9). A box whose ceiling cannot beat the best so far is skipped with everything inside it (line 7).

When the walk reaches a leaf, its runs are scored against B, and the best of them replaces the best so far if it is better (line 8). When no box is left, the best so far is the best chain score that ends at B, and the run that gave it is B's predecessor (line 10). A box is skipped only when its ceiling cannot beat the best. So the result of the tree always equals a scan over the whole pool. In practice the SIMD form of the tree allows the search to return the same chain scores and predecessors as the sequential scan at about 1/30 of its cost.

2.4.3 MAPQ ratio of voting

For each primary chain, minimap2 estimates its mapping quality with an empirical formula,

$$\text{mapQ} \;=\; 40 \cdot \Big(1 - \frac{f_2}{f_1}\Big) \cdot \min\Big\{1,\; \frac{m}{10}\Big\} \cdot \log f_1, \tag{5}$$

where m is the number of anchors on the primary chain, f_1 its chaining score and $f_2 \le f_1$ the score of the best chain secondary to it (Li, 2018, Section 2.1.3). FlashAlign keeps eq. (5) and lets the vote weigh the rivals: f_2/f_1 becomes

$$x \;=\; \max\Big\{\frac{f_2}{f_1},\; \max_i \; \frac{v_i}{v_1} \cdot \frac{f_i}{f_1}\Big\}. \tag{6}$$

f_2 is the score of the best secondary chain. v_1 and v_i are the numbers of seeds that voted for f_1 and f_i, with f_i the score of the i-th secondary chain. The revised formula shows we just added in the number of seeds that vote for a chain instead of just using the chain score ratio. The vote ratio helps FlashAlign become more informative on MAPQ than minimap2 while mapping to an assembly genome such as HG002, where the ratios f_i/f_1 are near 0.97 to 0.99 (Figure 1).

2.5 Base-level alignment

FlashAlign vendored minimap2's ksw2 kernel `ksw_extd2_sse` (v2.31) together with the minimap2 scoring rows for HiFi and ONT. The rescoring of rivals over HiFi is the same as in minibwa (Li and Homer, 2026). The specific Z-drop rule, its end extension, and the two-piece affine gap cost and the Suzuki–Kasahara formulation are described in Li (2018, Section 2.2) and Suzuki and Kasahara (2018) and are not re-derived here.

FlashAlign optimised the base level alignment by applying ungapped scoring path whenever its score is more optimal locally than gapped path. The ungapped alignment path has been applied in minimap2's short reads mode and minibwa. FlashAlign continues to apply that idea to long reads with an equality check on base level before materialising the Smith–Waterman DP.

Let $a_i = (q_i, r_i)$ be an anchor, an exact k-mer match, and $d_i = r_i - q_i$ its diagonal. For two consecutive anchors define

$$\begin{cases} \Delta d_i = d_{i+1} - d_i \\ \;\; L_i = q_{i+1} - q_i - k \\ g(l) = \min\{O + lE,\; O_2 + lE_2\} \end{cases} \tag{7}$$

where Δd_i is the change of diagonal between the two anchors, L_i the gap between their k-mers, of equal length on both sequences when $\Delta d_i = 0$, and $g(l)$ minimap2's two-piece cost of an l-base gap (Li, 2018, eq. 3), with match score A, mismatch penalty B and a penalty ν for a pairing with an N.

FlashAlign first compares the L_i bases of the gap without gaps when $\Delta d_i = 0$. Let the ungapped score for $\Delta d_i = 0$ as S_0.

$$S_0 \;=\; AL_i - (A+B)\,X - (A+\nu)\,z \tag{8}$$

Given X mismatches and z pairings with N, S_0 is better than gapped path if and only if

$$S_0 \;\ge\; A(L_i - 1) - 2g(1). \tag{9}$$

To make equation (9) true, this has to be satisfied:

$$(A+B)\,X + (A+\nu)\,z \;\le\; A + 2g(1). \tag{10}$$

Let P be any other alignment of the gap pinned at the two anchors. Assume $I(P)$ as amount of base pair that is inserted, $D(P)$ as deleted bases and gap cost $G(P)$ for path P. $\Delta d_i = 0$ and g is subadditive, so

$$\begin{cases} I(P) = D(P) \ge 1 \\ G(P) \ge 2g\big(I(P)\big) \end{cases} \tag{11}$$

because once entering an anchor of diagonal the path has to go back to the same diagonal. Therefore

$$\text{score}(P) \le A\big(L_i - I(P)\big) - G(P) \le A(L_i - 1) - 2g(1) \tag{12}$$

and no gapped alignment of the gap scores above S_0. To compute the condition of equation (9), a linear scan between base pairs of two anchors that live in the same diagonal is therefore sufficient. If that inequality does not pass, for example $\Delta d_i \neq 0$ or one that fails eq. (9), then it is filled as in minimap2 (Li, 2018, Section 2.2) the SIMD dynamic programming of Suzuki and Kasahara (2018) under the two-piece gap cost and over minimap2's band. In practice, this helps FlashAlign reach a nicer throughput without hurting the downstream biological meaning.

2.6 Spliced alignment

Unlike the genomic read problem, an intron is a reference jump that makes the diagonal of eq. (1) harder to compress, because each exon is a diagonal and the following intron shifts the diagonal by the length of the intron:

$$d_j \;=\; d_1 + \sum_{i=2}^{j} I_i, \tag{13}$$

FlashAlign adopts voting for RNA reads before the chaining stage. It still treats each exon as a unique peak, while each peak is no longer a placement but an exon hypothesis. The same splice gap model from minimap2 (Li, 2018) is applied for scoring to find the best coarse placement of the locus. Within each coarse placement, the exact anchors are recovered by interval search and chained with minimap2's intron-aware recurrence with an occurrence cap. Finally, the alignment uses the same splice-aware ksw2 kernel (Li, 2018) at minimap2's splice scoring. We found that applying the diagonal vote before minimap2's splice chaining and its splice DP helps the mapping performance of spliced long reads, as shown in Section 3.4.

3 Results

We evaluated the mapping accuracy on simulated reads and measured the throughput performance on real data. We tested FlashAlign v0.4.0, minimap2 v2.31-r1302 (Li, 2018), minibwa v0.7 (Li and Homer, 2026) and Winnowmap2 v2.03 (Jain et al., 2020, 2022), with each tool's recommended preset over HiFi and ONT long reads. Minibwa over ONT long reads mapping accuracy evaluation is not shown, since their GitHub suggested it does not work for long noisy reads (Li, 2026). All tools were run on a Linux workstation with an AMD Ryzen Threadripper 9970X 64-thread CPU at 5.0 GHz.

3.1 Mapping accuracy on simulated long reads

We used the Badread (Wick, 2019) for simulated long reads. For PacBio HiFi reads, we use the `pacbio2021` models and for ONT used the `nanopore2023` models. All interior knobs match how minibwa (Li and Homer, 2026) reported recipe for simulation long reads dataset. Evaluation is calculated by mapping errors compared to the simulated position to the mapping position.

Figure 1 reports mapping accuracy in the style of Li and Homer (2026) and Li (2018). A mapper of higher accuracy yields a curve closer to the upper-left corner. FlashAlign is more accurate than all other tools. This is plainly because most tools use the same inherent seed–chain–align recipe while FlashAlign added one more stage that can be informative enough for creating an improvement over the existing MAPQ formula.

We used the LRGASP simulated reads (Pardo-Palacios et al., 2024) for making the RNA panels (e,f). Because other existing RNA

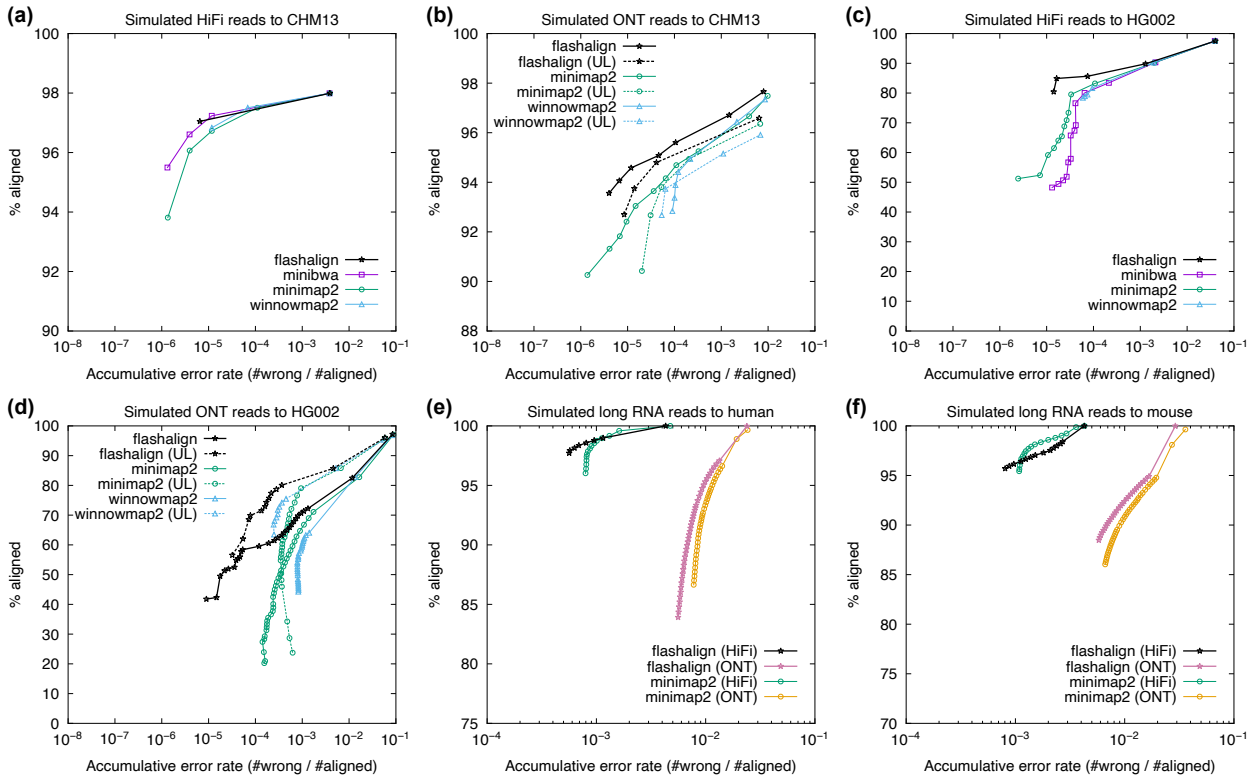


Figure 1. Mapping accuracy on simulated long reads. Reads are simulated from T2T-CHM13, the diploid HG002 genome or LRGASP transcript models and mapped back to the source reference. Suppose a read originates from interval G_s and is mapped to G_a. A read is considered as mapped correctly if $|G_s \cap G_a|/|G_s \cup G_a| \geq 10\%$. **(a,b)** HiFi and ONT reads simulated from T2T-CHM13. **(c,d)** HiFi and ONT reads simulated from the diploid HG002 genome. **(e,f)** LRGASP long RNA reads, human and mouse. Each ONT panel also carries that genome's ONT ultra-long cohort, as a second dashed arm of the same colour, labelled (UL).

long reads aligners don't have a proper MAPQ emission over the 0–60 range, the only comparison tool over RNA long reads is minimap2. All four lanes, human and mouse and HiFi and ONT, from LRGASP are being examined. From Figure 1e and f, as shown, generally, FlashAlign maps more while keeping the same or lower error rate as minimap2.

3.2 Mapping performance

We used several open source real reads dataset to evaluate our throughput performance on long reads compared to other aligners. For long genomic reads, we used the HG002 PacBio HiFi reads (https://bit.ly/hg002-hprc-revio), the HG002 Nanopore R10 reads (s3://ont-open-data/giab_2023) and the HG002 Nanopore R10.4.1 ultra-long reads (HPRC bucket s3://human-pangenomics, submission UCSC_GIAB_R1041_nanopore, HG002_R1041_UL). For spliced long reads, we used the H69 cell line reads of the LongBench dataset (You et al., 2025) (s3://longbench-data/raw/fastq/): PacBio Kinnex and ONT PCR-cDNA to GRCh38 with GENCODE 44.

FlashAlign is the fastest tool for handling long genomic reads (Figure 2). It is 1.7 to 2.6 times as fast as minimap2 and the fastest tool on all three tested long-read datasets. The voting mechanism allows chaining to become much more versatile. On long spliced RNA reads, FlashAlign is equal to or faster than minimap2. This is because the voting mechanism allows FlashAlign to align spliced reads of longer length faster than minimap2. The longer the reads in the dataset, the faster FlashAlign behaves on the spliced RNA dataset.

Table 2. Long-read structural variant calling on HG002 against the GIAB v5.0q truth. R recall, P precision.

	HiFi 37×			ONT 60×		
	R	P	F1	R	P	F1
FlashAlign	**0.797**	0.938	**0.862**	**0.756**	0.931	**0.835**
minimap2	0.785	**0.944**	0.857	0.749	**0.941**	0.834
minibwa	0.776	0.916	0.841	0.740	0.925	0.822
Winnowmap2	0.745	0.924	0.825	0.730	0.933	0.819

Table 3. Evaluation of junction accuracy on simulated ONT direct RNA reads

	FlashAlign	minimap2
# aligned reads	19 969 270	19 783 800
# chimeric alignments	7 030	6 635
# non-spliced alignments	4 216 935	4 242 046
# aligned introns	79 892 314	79 290 562
# novel introns	11 464	26 079
% exact introns	96.2	96.1
% approx. introns	99.1	99.1

Notes: The 20 million LRGASP simulated mouse ONT direct RNA reads (Pardo-Palacios et al., 2024) were mapped to the mouse genome GRCm39 with SIRV spike-ins with the following tools and command options: FlashAlign ('`--preset splice -u f`', index '`-k14`') and minimap2 ('`-x splice -uf -k14`'). No gene annotation was given to either tool. The alignments were compared to the annotation the reads were simulated from: GENCODE M27 basic plus the human GENCODE 38 transcripts that LRGASP placed as novel isoforms. A predicted intron is novel if it has no overlaps with any annotated introns. An intron is exact if it is identical to an annotated intron. An intron is approximate if both its 5′- and 3′-end are within 10 bp around the ends of an annotated intron. Chimeric alignments are supplementary alignments as defined in the SAM spec.

3.3 Structural variant calling

We aligned public long reads of HG002 from two platforms, PacBio Revio HiFi at ~37× and ONT R10.4.1 SUP at 60×, to T2T-CHM13v2.0 (Nurk et al., 2022) with FlashAlign, minimap2 2.31 (Li, 2018), minibwa 0.7 (Li and Homer, 2026) and Winnowmap2 2.03 (Jain et al., 2022) at each tool's recommended long-read preset. As evaluated, FlashAlign reaches the same or more biologically meaningful SV events compared to minimap2 and other aligners at its more versatile throughput. We believe the exact form of chaining helps the performance of chaining become more accurate over long

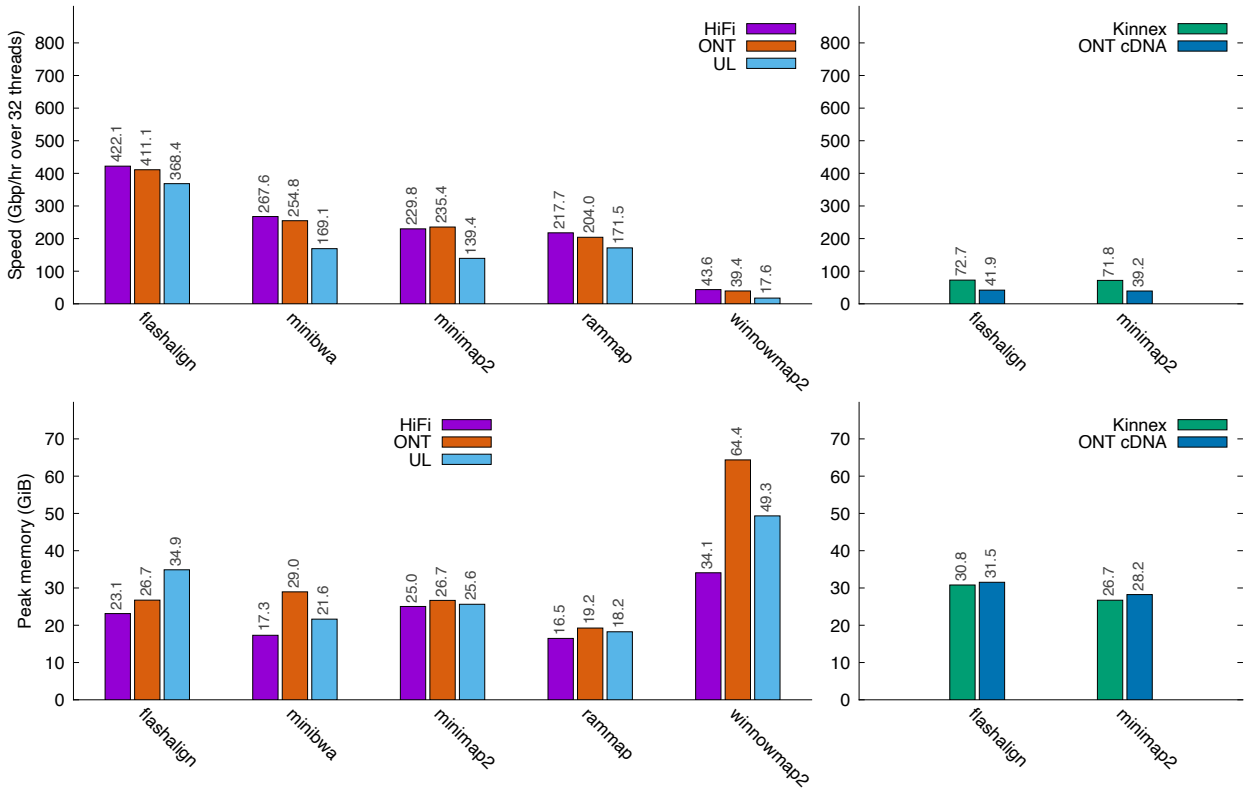


Figure 2. Speed and peak memory on real data. All tools in the panel are pinned to 32 threads, although Winnowmap2 consumed all 64 threads. PacBio HiFi and ONT are mapped to GRCh38 and ultra-long is mapped to T2T-CHM13v2.0. The RNA lanes are mapped to GRCh38 with GENCODE 44. All outputs of the aligners are sent to /dev/null.

reads.

SVs were then called with Sniffles2 2.8.0 (Smolka et al., 2024) at defaults and cuteSV 2.1.4 (Jiang et al., 2020) with `--genotype`. Then they are compared to the GIAB v5.0q draft benchmark (Genome in a Bottle Consortium, 2025; Zook et al., 2020) with Truvari 5.4.0 (English et al., 2022) using the benchmark's recommended settings (`--pick ac --passonly -r 2000 -C 5000`) with refine mode on (English et al., 2025). Table 2 gives the Sniffles2 numbers; cuteSV preserves the ordering. FlashAlign achieves better recall in finding structural variant events while losing a little precision compared to minimap2. Overall, FlashAlign is not only faster than minimap2 and other tools but ready for downstream analysis such as variant calling.

3.4 Spliced long reads

We evaluated our performance on spliced long reads against minimap2 on all four panels of spliced long reads from LRGASP. Overall, FlashAlign maps more reads than minimap2 (Table 3), because the vote finds a plausible locus before chaining. Its MAPQ is also better calibrated: at the same threshold it places more reads with a lower error rate. The exception is the human ONT cDNA panel, where FlashAlign is more conservative at Q60, keeping fewer reads with fewer errors.

4 Discussion

We developed FlashAlign for aligning genomic long reads and spliced long reads to accelerate science that involves long sequencing data. It creates a breakthrough of the existing idea of diagonal compression before chaining that existed in aligners back to the GraphMap (Sović et al., 2016) and minimap (Li, 2016) time. On genomic long reads FlashAlign is versatile enough to claim its novelty, and on spliced long reads FlashAlign achieves better mapping accuracy without hurting throughput. We found the same algorithm can also be applied to contig mapping. We plan to work on that to ship the feature in the FlashAlign package.

Acknowledgements

We are thankful to Heng Li for writing such a magical ksw2 base-level alignment kernel on top of the Suzuki–Kasahara formulation (Suzuki and Kasahara, 2018), and for his aligners minimap, minimap2 and minibwa (Li, 2016, 2018; Li and Homer, 2026), which gave us insights for making algorithm improvements.

Funding

[TODO:]

Data availability

FlashAlign source code:
https://github.com/97YearsOldProgrammer/flashalign
HG002 PacBio HiFi reads: https://bit.ly/hg002-hprc-revio
HG002 Nanopore R10 reads: s3://ont-open-data/giab_2023.05/
HG002 Nanopore ultra-long reads: s3://human-pangenomics/submissions/5b73fa0e-658a-4248-b2b8-cd16155bc157--UCSC_GIAB_R1041_nanopore/HG002_R1041_UL/
LongBench H69 reads: s3://longbench-data/raw/fastq/
LRGASP simulated RNA reads:
https://cgl.gi.ucsc.edu/data/LRGASP/data/simulation/

Conflict of interest

None declared.

References

Mark J. Chaisson and Glenn Tesler. Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory. *BMC Bioinformatics*, 13:238, 2012. doi: 10.1186/1471-2105-13-238.

Ghanshyam Chandra, Md Vasimuddin, Sanchit Misra, and Chirag Jain. Accelerating minimap2 for whole-genome alignment. *Bioinformatics*, 42(3): btag083, 2026. doi: 10.1093/bioinformatics/btag083.

Robert Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ*, 9:e10805, 2021. doi: 10.7717/peerj.10805.

Adam C. English, Vipin K. Menon, Richard A. Gibbs, Ginger A. Metcalf, and Fritz J. Sedlazeck. Truvari: refined structural variant comparison preserves allelic diversity. *Genome Biology*, 23:271, 2022. doi: 10.1186/s13059-022-02840-6.

Adam C. English, Egor Dolzhenko, Helyaneh Ziaei Jam, Sean K. McKenzie, Nathan D. Olson, Wouter De Coster, Jonghun Park, Bida Gu, Justin Wagner, Michael A. Eberle, Melissa Gymrek, Mark J. P. Chaisson, Justin M. Zook, and Fritz J. Sedlazeck. Analysis and benchmarking of small and large genomic variants across tandem repeats. *Nature Biotechnology*, 43 (3):431–442, 2025. doi: 10.1038/s41587-024-02225-z.

Genome in a Bottle Consortium. HG002 T2T-CHM13v2.0 draft benchmark v5.0q: small and structural variants. NIST Genome in a Bottle FTP release, 2025. URL https://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/release/AshkenazimTrio/HG002_NA24385_son/v5.0q/. Dataset release v5.0q, accessed 2026-09-04.

Chirag Jain, Arang Rhie, Haowen Zhang, Claudia Chu, Brian P. Walenz, Sergey Koren, and Adam M. Phillippy. Weighted minimizer sampling improves long read mapping. *Bioinformatics*, 36(Suppl_1):i111–i118, 2020. doi: 10.1093/bioinformatics/btaa435.

Chirag Jain, Arang Rhie, Nancy F. Hansen, Sergey Koren, and Adam M. Phillippy. Long-read mapping to repetitive reference sequences using Winnowmap2. *Nature Methods*, 19:705–710, 2022. doi: 10.1038/s41592-022-01457-8.

Tao Jiang, Yongzhuang Liu, Yue Jiang, Junyi Li, Yan Gao, Zhe Cui, Yadong Liu, Bo Liu, and Yadong Wang. Long-read-based human genomic structural variation detection with cuteSV. *Genome Biology*, 21:189, 2020. doi: 10.1186/s13059-020-02107-y.

Saurabh Kalikar, Chirag Jain, Md Vasimuddin, and Sanchit Misra. Accelerating minimap2 for long-read sequencing applications on modern CPUs. *Nature Computational Science*, 2(2):78–83, 2022. doi: 10.1038/s43588-022-00201-8.

W. James Kent. BLAT—the BLAST-like alignment tool. *Genome Research*, 12(4):656–664, 2002. doi: 10.1101/gr.229202.

Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, 2016. doi: 10.1093/bioinformatics/btw152.

Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018. doi: 10.1093/bioinformatics/bty191.

Heng Li. New strategies to improve minimap2 alignment accuracy. *Bioinformatics*, 37(23):4572–4574, 2021. doi: 10.1093/bioinformatics/btab705.

Heng Li. minibwa: fast genomic read alignment. https://github.com/lh3/minibwa, README, 2026. accessed 2026-09-04.

Heng Li and Nils Homer. Fast genomic read alignment with minibwa. *arXiv*, 2026. doi: 10.48550/arXiv.2606.15357. URL https://arxiv.org/abs/2606.15357. arXiv:2606.15357 [q-bio.GN].

Yang Liao, Gordon K. Smyth, and Wei Shi. The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote. *Nucleic Acids Research*, 41(10):e108, 2013. doi: 10.1093/nar/gkt214.

Zemin Ning, Anthony J. Cox, and James C. Mullikin. SSAHA: a fast search method for large DNA databases. *Genome Research*, 11(10):1725–1729, 2001. doi: 10.1101/gr.194201.

Sergey Nurk, Sergey Koren, Arang Rhie, Mikko Rautiainen, et al. The complete sequence of a human genome. *Science*, 376(6588):44–53, 2022. doi: 10.1126/science.abj6987.

Francisco J. Pardo-Palacios, Dingjie Wang, Fairlie Reese, Mark Diekhans, et al. Systematic assessment of long-read RNA-seq methods for transcript identification and quantification. *Nature Methods*, 21:1349–1363, 2024. doi: 10.1038/s41592-024-02298-3.

William R. Pearson and David J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, 1988. doi: 10.1073/pnas.85.8.2444.

Kim R. Rasmussen, Jens Stoye, and Eugene W. Myers. Efficient q-gram filters for finding all ϵ-matches over a given length. *Journal of Computational Biology*, 13(2):296–308, 2006. doi: 10.1089/cmb.2006.13.296.

Jingwen Ren and Mark J. P. Chaisson. lra: A long read aligner for sequences and contigs. *PLOS Computational Biology*, 17(6):e1009078, 2021. doi: 10.1371/journal.pcbi.1009078.

Michael Roberts, Wayne Hayes, Brian R. Hunt, Stephen M. Mount, and James A. Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, 2004. doi: 10.1093/bioinformatics/bth408.

Fritz J. Sedlazeck, Philipp Rescheneder, Moritz Smolka, Han Fang, Maria Nattestad, Arndt von Haeseler, and Michael C. Schatz. Accurate detection of complex structural variations using single-molecule sequencing. *Nature Methods*, 15:461–468, 2018. doi: 10.1038/s41592-018-0001-7.

Moritz Smolka, Luis F. Paulin, Christopher M. Grochowski, et al. Detection of mosaic and population-level structural variants with Sniffles2. *Nature Biotechnology*, 42:1571–1580, 2024. doi: 10.1038/s41587-023-02024-y.

Ivan Sović, Mile Šikić, Andreas Wilm, Shannon N. Fenlon, Swaine Chen, and Niranjan Nagarajan. Fast and sensitive mapping of nanopore sequencing reads with GraphMap. *Nature Communications*, 7:11307, 2016. doi: 10.1038/ncomms11307.

Hajime Suzuki and Masahiro Kasahara. Introducing difference recurrence relations for faster semi-global alignment of long sequences. *BMC Bioinformatics*, 19(Suppl 1):45, 2018. doi: 10.1186/s12859-018-2014-8.

Jeremy R. Wang and Heng Li. Memory-safe high-performance sequence mapping with rammap. *bioRxiv*, 2026. doi: 10.64898/2026.05.26.726289.

Ryan R. Wick. Badread: simulation of error-prone long reads. *Journal of Open Source Software*, 4(36):1316, 2019. doi: 10.21105/joss.01316.

W. John Wilbur and David J. Lipman. Rapid similarity searches of nucleic acid and protein data banks. *Proceedings of the National Academy of Sciences*, 80(3):726–730, 1983. doi: 10.1073/pnas.80.3.726.

Yupei You, Ashleigh N. Solano, James Lancaster, Margaux David, Changqing Wang, Jia Wei Tan, Shian Su, Camilla Pasquali, Kathleen Zeglinski, Reza Ghamsari, Manveer Chauhan, Josie Gleeson, Yair D. J. Prawer, Jin Ng, Bénédicte Dubois, Isabelle Cleynen, Marie-Liesse Asselin-Labat, Nadia M. Davidson, Kate D. Sutherland, Michael B. Clark, Quentin Gouil, and Matthew E. Ritchie. Benchmarking long-read RNA-sequencing technologies with LongBench: a cross-platform reference dataset profiling cancer cell lines with bulk and single-cell approaches. *bioRxiv*, 2025. doi: 10.1101/2025.09.11.675724.

Justin M. Zook, Nancy F. Hansen, Nathan D. Olson, et al. A robust benchmark for detection of germline large deletions and insertions. *Nature Biotechnology*, 38:1347–1355, 2020. doi: 10.1038/s41587-020-0538-8.