Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

Aligning distant sequences to graphs using long seed sketches

Amir Joudaki, Alexandru Meterez, Harun Mustafa, et al.

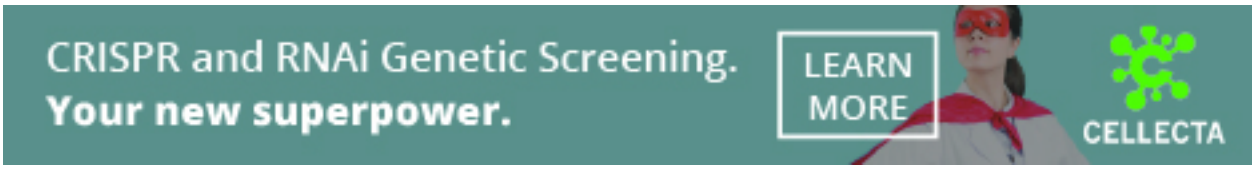
Genome Res. published online April 18, 2023
Access the most recent version at doi:10.1101/gr.277659.123

P<P Published online April 18, 2023 in advance of the print journal.

Accepted Manuscript Peer-reviewed and accepted for publication but not copyedited or typeset; accepted manuscript is likely to differ from the final, published version.

Creative Commons License This article is distributed exclusively by Cold Spring Harbor Laboratory Press for the first six months after the full-issue publication date (see https://genome.cshlp.org/site/misc/terms.xhtml). After six months, it is available under a Creative Commons License (Attribution-NonCommercial 4.0 International), as described at http://creativecommons.org/licenses/by-nc/4.0/.

Email Alerting Service Receive free email alerts when new articles cite this article - sign up in the box at the top right corner of the article or **click here.**



To subscribe to *Genome Research* go to:
https://genome.cshlp.org/subscriptions

Published by Cold Spring Harbor Laboratory Press

Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

Aligning distant sequences to graphs using long seed sketches

Amir Joudaki[1,2,*], Alexandru Meterez[1,*], Harun Mustafa[1,2,3],
Ragnar Groot Koerkamp[1], André Kahles[1,2,3,†], and Gunnar Rätsch[1,2,3,4,†]

[1] Department of Computer Science, ETH Zurich, Zurich 8092, Switzerland
[2] University Hospital Zurich, Biomedical Informatics Research, Zurich 8091, Switzerland
[3] Swiss Institute of Bioinformatics, Lausanne 1015, Switzerland
[4] ETH AI Center, 8092 Zurich, Switzerland
{firstname}.{lastname}@inf.ethz.ch
[*] Equal contribution.
[†] To whom correspondence should be addressed.

Abstract. Sequence-to-graph alignment is crucial for applications such as variant genotyping, read error correction, and genome assembly. We propose a novel seeding approach that relies on long inexact matches rather than short exact matches, and demonstrate that it yields a better time-accuracy trade-off in settings with up to a 25% mutation rate. We use sketches of a subset of graph nodes, which are more robust to indels, and store them in a k-nearest neighbor index to avoid the curse of dimensionality. Our approach contrasts with existing methods and highlights the important role that sketching into vector space can play in bioinformatics applications. We show that our method scales to graphs with 1 billion nodes and has quasi-logarithmic query time for queries with an edit distance of 25%. For such queries, longer sketch-based seeds yield a 4× increase in recall compared to exact seeds. Our approach can be incorporated into other aligners, providing a novel direction for sequence-to-graph alignment.

Keywords: sequence-to-graph alignment, tensor sketching, tensor embedding

Introduction

Our work focuses on *sequence-to-graph alignment*, defined as aligning a query sequence to a sequence graph (Garrison et al. 2018; Ghaffaari and Marschall 2019; Rautiainen and Marschall 2020). Sequence-to-graph alignment involves finding the minimal number of *editing operations* to transform the query to a reference sequence encoded in the graph. While there are various cost schemes for penalizing edits, edit distance assigns a cost of 1 to *substitution, insertion*, and *deletion* on the query.

Since the time complexity of optimal sequence-to-graph alignment grows linearly with the number of edges in the graph (Jain et al. 2020; Gibney et al. 2022), many approaches instead follow an approximate *seed-and-extend* strategy (Altschul et al. 1990), which operates in four main steps: i) *seed extraction*, which in its simplest form involves finding all substrings with a certain length, ii) *seed anchoring*, finding matching nodes in the graph, iii) *seed filtration*, often involving clustering (Chang et al. 2020; Rautiainen and Marschall 2020) or co-linear chaining (Karasikov et al. 2022; Almodaresi et al. 2021; J Ma et al. 2022; Chandra and Jain 2022) of seeds, and iv) *seed extension*, involving performing semi-global pairwise sequence alignment forwards and backwards from each anchored seed (H Li 2013). We will review the usage of exact seeds utilized in tools such as VG(Garrison et al. 2018) and GraphAligner (Rautiainen and Marschall 2020) and discuss their limitations in a high mutation-rate setting.

The following example highlights the limitations of *k-mers*, defined as substrings with length k, as candidates for seeding:

Example 1 (Limitations of k-mer seeds). We have a reference sequence, denoted as S, that is uniformly drawn from the set of DNA nucleotides {`A,C,G,T`}. We create a query sequence q, by copying a substring of S at some offset Δ, and substituting each nucleotide independently with probability r. In order to retrieve the offset Δ using k-mer matches, at least one k-mer in the query must be copied without mutations. The probability of having at least one intact k-mer in the query is bounded by $(|q| - k + 1)(1 - r)^k$ by union bound. We also expect to have $\mathcal{O}(|S||q|4^{-k})$ matches between all pairs of reference and query k-mers due to chance (see the formal analysis in Supplemental Material A). Informally, we have:

$$\text{recall} \lesssim \text{query length} \cdot (1-r)^k, \qquad \text{total matches} \approx \text{ref length} \cdot \text{query length} \cdot 4^{-k}. \qquad (k\text{-mer matches})$$

Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

The simple setting in Example 1 illustrates the challenge many alignment methods face in trading off recall with time complexity. Since using a large k for seed length reduces the number of spurious matches, methods such as BWA-MEM (H Li 2013), E-MEM (Khiste and Ilie 2015), deBGA (Liu et al. 2016), and PuffAligner (Almodaresi et al. 2021) find maximal exact matches (MEMs) between the read and the reference genome. However, the higher precision comes at the expense of fewer seeds and lower recall. For example, for $k = 31$, mutation rate $r = 0.25$, and query length 100 in Example 1, the expected recall is only $(100)(1 - 0.25)^{31} \approx 0.01$.

The alternative is to use shorter seeds to increase the likelihood of finding accurate alignments. Methods that use De Bruijn graphs as graph model (Bowe et al. 2012), or an auxiliary index (Sirén 2017), are restricted to finding seeds of minimum length k. To find seeds when the query has a high edit distance relative to the reference sequences, sequence-to-graph alignment tools will either set k to a small value (Limasset et al. 2016; Garrison et al. 2018; Rautiainen and Marschall 2020) or use a variable-order DBG model (Boucher et al. 2015; Belazzougui and Cunial 2019; Karasikov et al. 2020; Karasikov et al. 2022) to also allow for anchoring seeds of length less than k (Karasikov et al. 2020; Karasikov et al. 2022). However, shorter seeds generally lead to a more complex and connected graph topology which can lead to a combinatorial explosion of the search space. In the same setting as Example 1, with $|S| = 10^9, |q| = 100$ and $k = 12$, there will be approximately $10^9 \cdot 100 \cdot 4^{-12} \approx 5,960$ matches due to chance.

Due to the inherent shortcomings of short and long exact matches, Břinda, et al. (Břinda et al. 2015) propose *spaced seeds* (Califano and Rigoutsos 1993; Pevzner and Waterman 1995; B Ma et al. 2002; Burkhardt and Kärkkäinen 2003) to incorporate long seeds at higher mutation rates by masking out some positions in the seed. For example, using 0101 as a mask, "A$\underline{\text{C}}$G$\underline{\text{T}}$" will match with "G$\underline{\text{C}}$A$\underline{\text{T}}$". While spaced seeds were shown to be more sensitive than contiguous seeds without increasing the number of spurious matches, they assume that only mismatches occur in the alignment with no *insertions* or *deletions (indels)* (Mak et al. 2006). Another alternative to k-mers for sequence comparison is the use of strobemers (Sahlin 2021), which provide improved performance in terms of evenly distributed sequence matches, sensitivity to different mutation rates, and match coverage across sequences.

In this work, we propose using long inexact seeds based on Tensor Sketching (Joudaki et al. 2021). We use a succinct *De Bruijn graph (DBG)* (Bowe et al. 2012) as the graph model, and preprocess it in two main stages: i) A subset of nodes in the DBG is sketched into a vector space, while insertions, deletions, and substitutions are approximately embedded into the ℓ^2 metric. ii) To be able to efficiently retrieve similar sketch vectors, the sketches of nodes are stored in a *Hierarchical Navigable Small Worlds (HNSW)* (Malkov and Yashunin 2018) index. Aligning a query sequence involves three main stages: i) Compute sketch vectors for all k-mers in the query. ii) Use the HNSW index to find the nearest vectors as candidate seeds. iii) Extend these seeds backwards and forwards to find the best possible alignment. Figure 1 gives an overview of our sketch-based scheme.

The main contributions of this work are demonstrating that the Tensor Sketching method can be incorporated into a seed-and-extend sequence-to-graph alignment approach, which is a new direction for applying sketching methods in bioinformatics, and showing experimental results that demonstrate the effectiveness of our method in improving accuracy for high mutation rates, using both simulated and real meta-genome graphs.

Results

We implemented the MetaGraph Sketch (MG-Sketch) algorithm, which uses our novel sketch-based seeder, in the MetaGraph (Karasikov et al. 2020) framework. We primarily compare against MetaGraph Align (MG-Align), which is also implemented in MetaGraph, but uses an exact seeding scheme. We compare MG-Sketch against GraphAligner (GA) (Rautiainen and Marschall 2020), vg map (Hickey et al. 2020) and vg mpmap (Sibbesen et al. 2023) on both synthetic and real datasets (See Supplemental Material B for more details).

Synthetic Data Generation

We generate a synthetic dataset by initializing the sequence pool with $\mathcal{S}_0 \leftarrow \{s_0\}$, where s_0 is a random sequence $s_0 \sim \Sigma^N$, for some fixed length $N \in \mathbb{N}^+$. At each level $i \in \mathbb{N}^+$, we randomly mutate all sequences in

Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

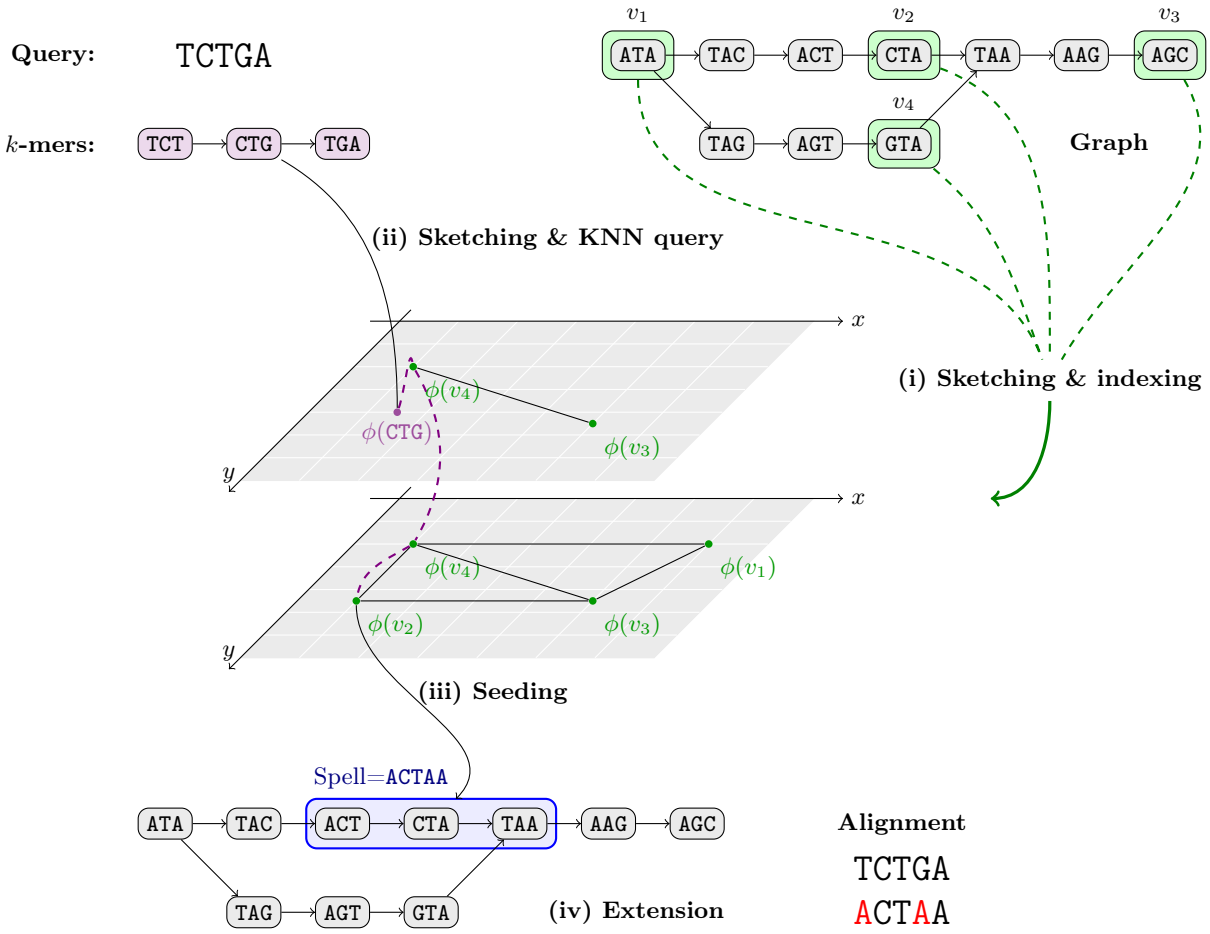


Fig. 1: Illustrating the four main stages of MG-Sketch for a sequence graph with $k = 3$ node lengths. (i) Preprocessing involves sampling a subset of nodes (highlighted in green) to ensure that every walk of length 3 contains at least one sampled node. Each sampled node is sketched using $\phi : \Sigma^k \to \mathbb{R}^2$, which approximates the edit distance. The sketches are stored in a hierarchical structure with multiple levels, and connections between nodes are navigated to refine nearest neighbor queries. (ii) Sketch every k-mer in the query (magenta) and find the nearest neighbors stored in the index. (iii) Seed an alignment at the node returned in the previous step. (iv) Extend forwards and backwards from each seed, finding the path in the graph that maximizes the alignment score of the query.

Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

the pool by independently substituting every character with 1% probability. We add the mutated sequences to the pool $\mathcal{S}_i \leftarrow \mathcal{S}_{i-1} \cup \{\text{mutate}(s) \colon s \in \mathcal{S}_{i-1}\}$, doubling the pool size. We repeat this process up to ℓ levels.

We build a De Bruijn graph G using MetaGraph (Karasikov et al. 2020), on the synthetic sequences $\mathcal{S}_\ell$ with $N = 1000$ and $k = 80$. Recall that all sequences in S_ℓ are of size N, resulting in $O(2^\ell N)$ nodes in the graph. Construction of S_ℓ ensures that approximately 1% of the nodes in G are "branching", i.e. they have at least one incoming or outgoing connection, thereby increasing the difficulty of the sequence-to-graph alignment.

To create the ground truth sequences for alignment, we start with a random walk w_i in the graph consisting of 250 edges. The reference sequences s_i are obtained by projecting the walk onto the graph, resulting in a spelled sequence of length $|\pi(w_i)| = 80 + 250 = 330$. Due to the size of the graphs and number of branches in our experiments, it is highly unlikely that the same path will be sampled multiple times. The query sequences q_i are obtained by applying mutations with rates of 5%, 10%, 15%, 20%, and 25% to the reference sequences s_i, where each mutation is i.i.d. and with probability $\frac{1}{3}$ is a substitution, insertion, or deletion. Conceptually, the alignment method f takes the query q_i and graph G as input, and returns a candidate spell $f(q_i, G)$ as output. We quantify the optimality of f by measuring the edit distance between the reference and the candidate spell: $\text{ed}\,(s_i, f(q_i))$. We define recall as $\text{ed}\,(s_i, f(q_i)) \leq |s_i|\lambda$, where $\lambda \in [0, 1]$ controls the accuracy of the alignment (lower is more accurate). For all experiments, we set $\lambda = 0.1$. Finally, we average the recall and alignment time per query over 500 samples at each mutation rate: $\text{Recall}_f(r) = \frac{1}{500}|\{i \in [500] : \text{ed}\,(s_i, f(q_i)) \leq \lambda|s_i|\}|$.

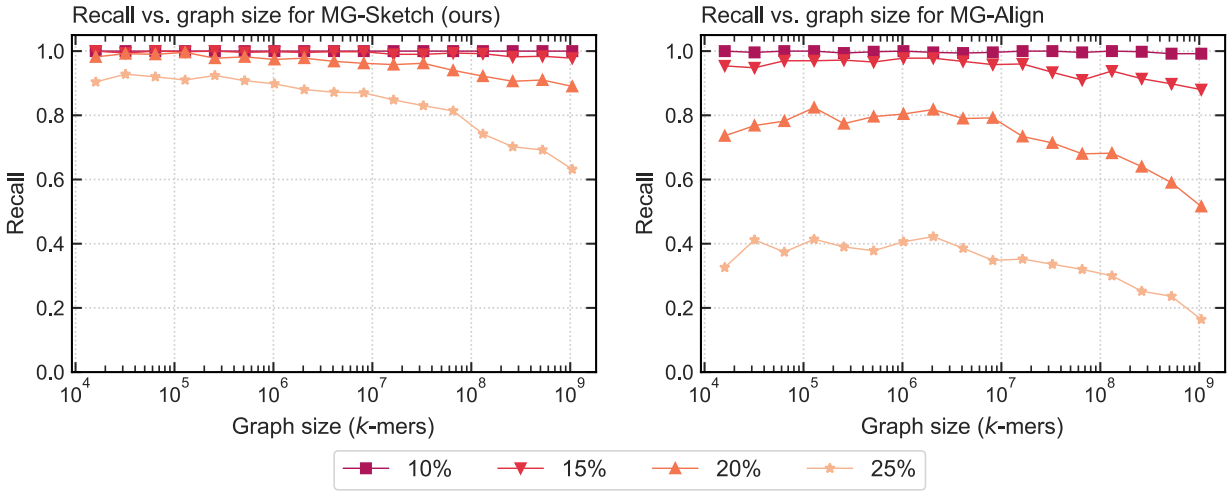


Fig. 2: Recall achieved across different mutation rates with increasing graph sizes for MG-Sketch (left) and MG-Align (right). We run MG-Sketch with $t = 6, w = 16, s = 8, D = 14$, and $K = 40$ neighbors. Query generation follows the same approach as explained in Synthetic Data Generation. We omit 0% and 5% mutation rate, as both methods achieve an almost perfect recall.

Sketch-based alignment achieves high recall in high mutation settings.

We show that MG-Sketch achieves a uniformly higher recall than MG-Align. Figure 2 shows that MG-Sketch outperforms MG-Align across all graph sizes, particularly in high mutation regimes. MG-Sketch reaches 3× and 1.8× higher recall than MG-Align on 25% and 20% mutation rates respectively, while getting a comparable or better recall in all other cases. If MG-Align does not find exact 80-mer matches, it falls back to shorter seeds, until a minimum seed length of 15. If we compute the expected number of hits analogous to Example 1, we get $(330/15)(1 - 0.25)^{15} \approx 0.29$. This is comparable to the empirical value for 25% mutation in Figure 2. Our results show that MG-Sketch exceeds this recall by a significant margin, highlighting the importance of long inexact seeds.

Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

Sketch-based aligner scales quasi-linearly with graph size.

We show that the high accuracy of MG-Sketch does not compromise scalability. To compare the scalability of our approach, we measured peak memory usage and average alignment time per query for graphs with $\ell \in \{4, \ldots, 20\}$, as outlined in Synthetic Data Generation. This generates graphs with 16000 up to approximately 1B k-mers, posing a challenge to the scalability of each method. We use the MetaGraph base graph for evaluating MG-Sketch and MG-Align (See Supplemental Material B for more details).

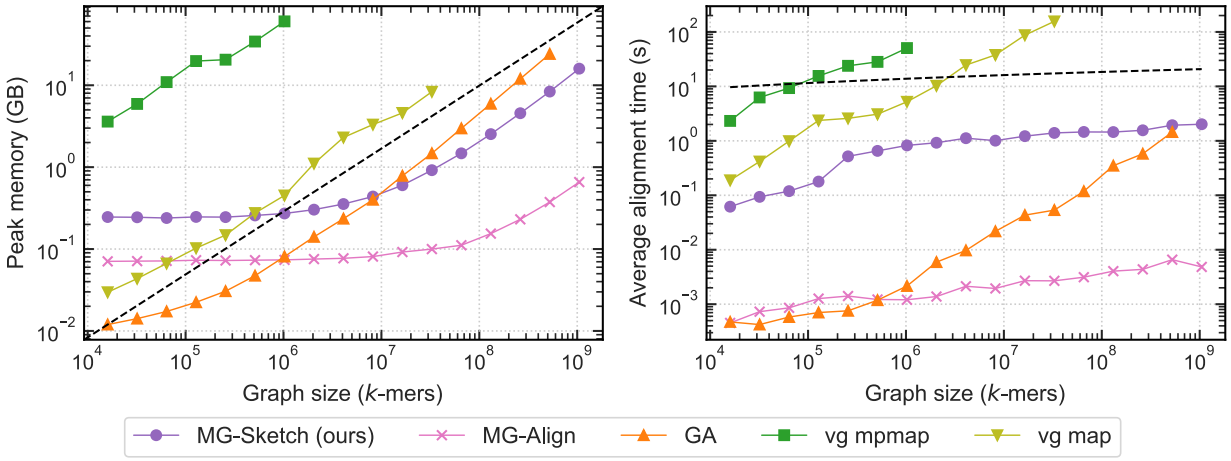


Fig. 3: Peak RAM usage and query time vs. graph size. Graphs are generated with $k = 80$ according to Synthetic Data Generation. We run MG-Sketch with $t = 6, D = 14, w = 16, s = 8$ and $K = 10$ neighbors, using query sequences with a mutation rate of 25%. Traces for vg mpmap and vg map are incomplete as they exceed time or memory limit. (left) Peak RAM usage vs. graph size, with the black dashed line indicating linear memory complexity. (right) Average alignment time vs. graph size, with the black dashed line indicating logarithmic time complexity. For recall comparisons see Figure 4.

Peak memory usage scales linearly with graph size. Peak memory usage imposes a hard limit on the size of graphs that a method can preprocess. In Figure 3, we observe that peak memory usage in MG-Sketch scales linearly with the graph size. In particular, for graphs with over 10M nodes, MG-Sketch requires less memory than all other methods except MG-Align. For MG-Align, lower memory usage comes at the cost of lower recall, when compared to MG-Sketch. In contrast, vg mpmap peak memory usage grows rapidly above 80GB. Therefore, we only evaluate vg mpmap for graphs with up to 1M nodes. While vg map requires less memory than vg mpmap, the runtime exceeded 24 hours for graphs with over 60M nodes. It is evident in Figure 3 that the peak memory usage of GA grows faster than linear, making MG-Sketch and MG-Align the only methods with linear memory complexity.

Query alignment time grows quasi-logarithmically with graph size. Given that sequence graphs have billions of nodes, it is crucial for the alignment time to avoid growing linearly with the graph size. Figure 3 demonstrates that MG-Sketch achieves an almost logarithmic scale with the graph size. In contrast, the average time for vg map and vg mpmap grow much faster than logarithmic. The alignment time for GA also grows faster than linearly, however, this includes the preprocessing time for building the minimizer seeder from the graph.

Notably, the scalability of MG-Sketch does not compromise the recall. Figure 4 demonstrates that for all graph sizes and mutation rates, MG-Sketch achieves a higher or equal recall compared to the other tools, with the exception of vg mpmap. While vg mpmap has higher recall at 25%, for graphs with over 1M it exceeds our memory limit of 80GB.

Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

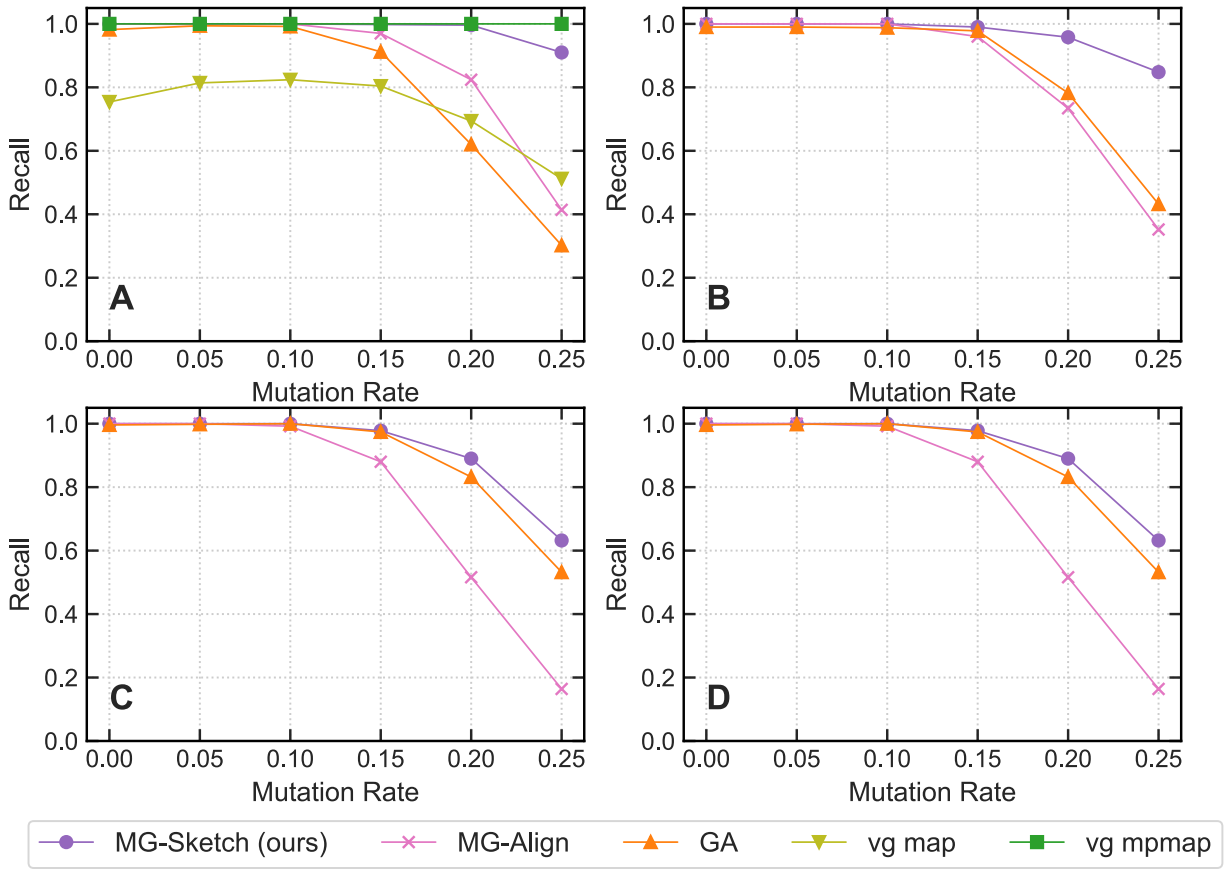


Fig. 4: Recall achieved across different mutation rates with increasing graph sizes for each baseline. The number of nodes in the graph for each plot is 100K (A), 10M (B), 100M (C) and 1B (D). Values are measured on the De Bruijn graph generated by MetaGraph. We run MG-Sketch with $K = 40$ neighbors and $D = 14, w = 16, s = 8, t = 6$. Query generation follows the same approach as explained in Synthetic Data Generation, with mutation rates $0\%, 5\%, 10\%, 15\%, 20\%, 25\%$.

Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

In scenarios with low mutation rates, as evident in Figure 2, both MG-Align and MG-Sketch achieve nearly perfect recall. However, MG-Align exhibits faster runtimes in these cases. The relatively slower performance of MG-Sketch in low mutation cases is due to the extension from inexact sketch-based seeds. The relatively slower performance of MG-Sketch in low mutation cases is not unexpected, considering our primary focus is on high mutation scenarios where most methods face challenges in achieving high recall.

Good scalability and recall translate into real-world applications.

To demonstrate our approach in a more practical setting, we generated a pan-genome De Bruijn graph containing 51,283 assembled virus genome sequences collected from GenBank (Mustafa et al. 2018). Graph construction was performed using the *MetaGraph* framework, utilizing a k of 80. The resulting graph contained 337,480,265 nodes. Similar to previous evaluations, we generated 500 query sequences by sampling random walks of length 330 from the graph, subsequently applying random mutations at rates ranging from 5 to 25 percent. Confirming our previous observations on scalability, the sketching-based approach was able to outperform both the MG-Align as well as the GA approaches, starting from mutation rates of 10% and 15%, respectively (Figure 5). Notably, for the highest mutation rate, MG-Sketch almost doubles the recall when compared to the next best approach.

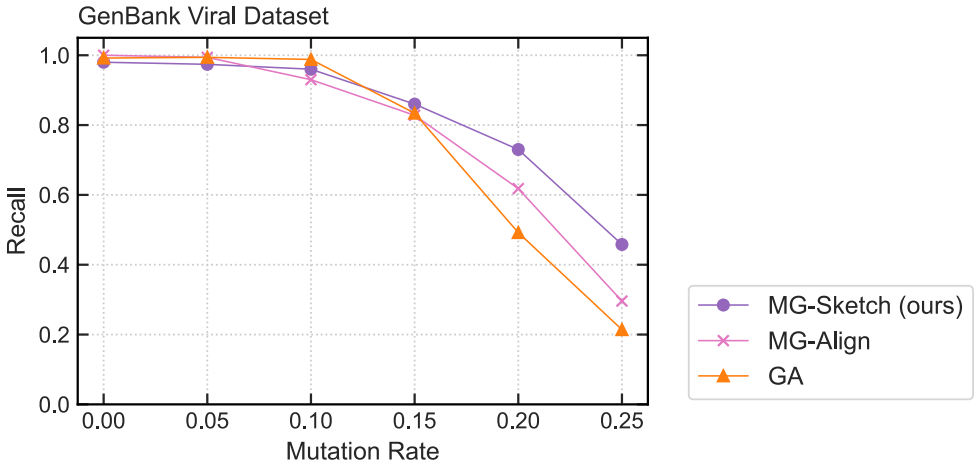


Fig. 5: Seeding recall for the GenBank viral graph. Recall is shown for MG-Sketch, MG-Align, and GA as purple, pink, and orange lines, respectively, for query sets of increasing distance, simulated with a random mutation rate ranging from 0% to 25%.

Sketches linearly preserve mutation rate. Tensor Sketching can be described as a dimensionality reduction technique for the Tensor Embedding of a sequence, which is defined as the number of occurrences of fixed-length t subsequences in the sequence (see Methods for details). Unlike k-mers, subsequences can appear with an arbitrary number of gaps in between, making them more robust to local changes in the sequence. If we assume constant subsequence length $t = O(1)$, a single insertion, deletion, or substitution, approximately affects $O(1/n)$ fraction of subsequences, where n is the length of the sequence.

We can formalise this intuition under a random sampling and substitution-only mutation regime, by proving that the expected Tensor Embedding distance scales linearly with the mutation rate. Since Tensor Sketching approximates the Tensor Embedding distance, it approximately preserves the mutation rate while keeping the time complexity linear as a function of the sequence length and t. Define (normalized) Tensor Embedding distance d_{te} between two sequences a and b as:

$$d_{te}(a,b) := \binom{n}{2t-1}^{-1} \cdot \|T_a - T_b\|_2^2. \tag{1}$$

Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

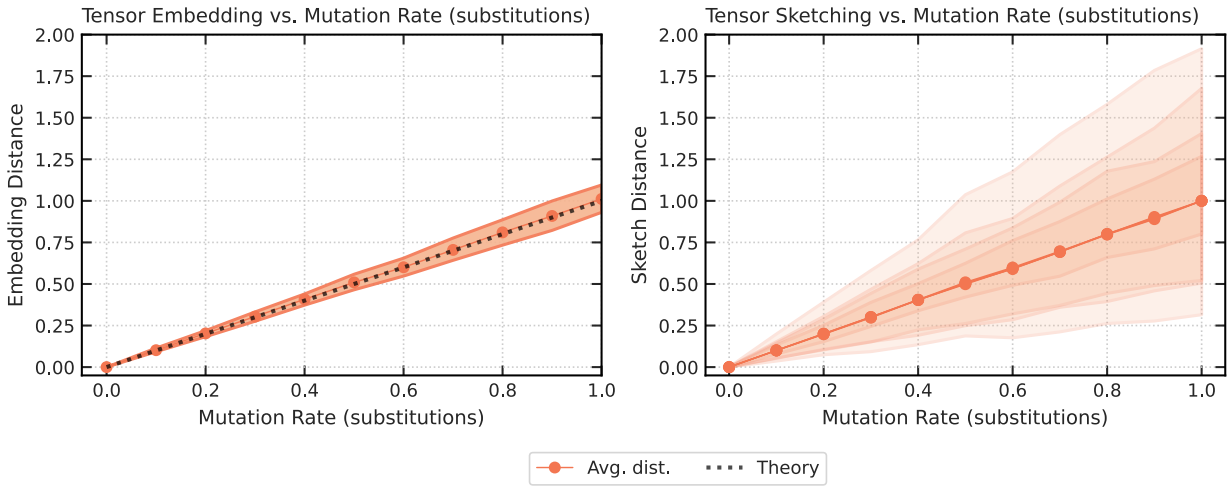


Fig. 6: Comparison of expected distances for Tensor Embedding (left) and Tensor Sketching (right) as a function of the substitution mutation rate, for sequences of length $n = 1000$. Each data point is computed using 100 independent sequence pairs, and the process is repeated 100 times to compute the mean shown as the solid trace, and the confidence interval shown as the shaded region, which covers the $5\% - 95\%$ range of the results. (left) The solid trace shows the mean Tensor Embedding distance with $t = 3$, while the dashed trace shows the theoretical lower bound from Lemma 1. (right) The solid trace shows the mean Tensor Sketch distance with $t = 3$ and $D = 8, 16, 32, 64$. The shaded regions correspond to increasing values of D, with the lightest shade corresponding to $D = 8$ and the darkest to $D = 64$.

The following lemma demonstrates that Tensor Embedding approximates edit distance:

Lemma 1. *Tensor Embedding preserves mutation rate (substitution only) under ℓ^2 norm :: Let a be a uniform random sequence of length n in Σ^n, and for a given mutation rate $r \in [0, 1]$ let b be a sequence where a_i is substituted by a new character $b_i \in Unif(\Sigma \backslash \{a_i\})$ with probability r and $b_i = a_i$ otherwise. Then:*

$$\mathbb{E}_{a,b}[d_{te}(a, b)] = (1 + o(1)) \cdot 2^{2t-1} \cdot |\Sigma|^{-t+1} \cdot r. \tag{2}$$

Note that the 2^{2t-1} factor does not depend on the sequences. Therefore, Lemma 1 provides a guarantee that the average distance of mutated pairs remains within a linear estimate of the mutation rate. Observe that the edit distance in this setting will be approximately nr. Therefore, we can, alternatively, refer to r as edit distance normalized by length. The proof of Lemma 1 is given in Supplemental Material C. For numerical validation of this bound, see Figure 6. While Lemma 1 is stated for substitution-only mutations, tensor embedding and sketching linearly preserve general mutation rates, including indels, up to a constant distortion factor (see Figure 7).

Discussion

In this work, we present a novel approach to sequence alignment called MG-Sketch that is resistant to mutations. Our method uses sketch-based seeds, and we demonstrate through theoretical analysis and empirical results that it scales to graphs with up to 10^9 nodes. While MG-Align and MG-Sketch share the same extension algorithm, our sketch-based seeds significantly improve alignment recall in high mutation rate regimes of over 15%. Conversely, in scenarios where exact matches can be constrained, such as low mutation rate cases, MG-Align exhibits faster runtimes while producing recall results that are comparable to MG-Sketch. This is because MG-Align avoids the overhead associated with extending from inexact sketch-based seeds. We believe that sketching schemes that preserve edit distance, such as Tensor Sketching, offer a promising direction for pushing the boundaries of sequence analysis to higher variation settings.

Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

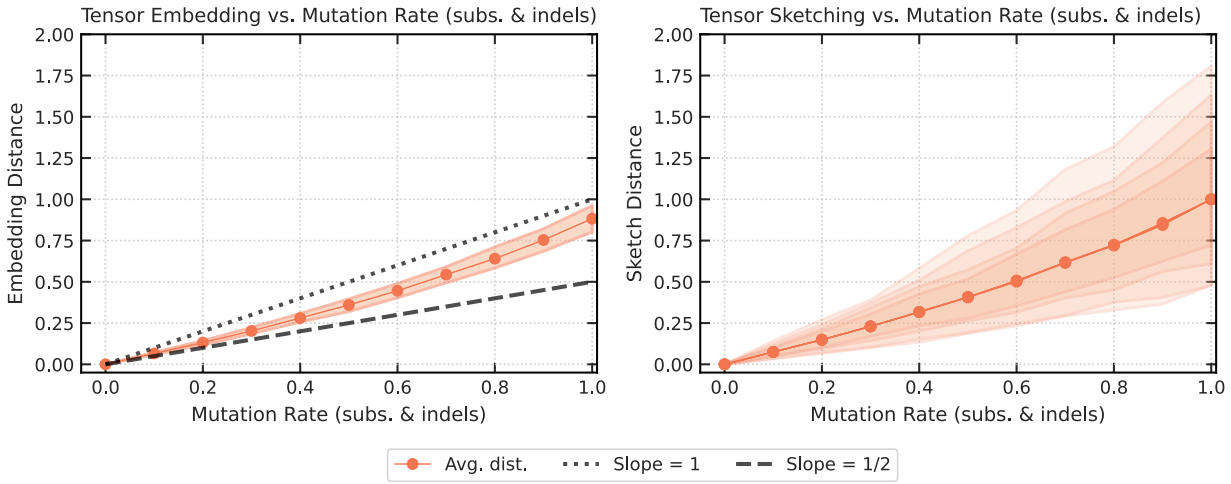


Fig. 7: Comparison of expected distances for Tensor Embedding (left) and Tensor Sketching (right) as a function of the substitution, insertion, and deletion mutation rate, with each mutation type having probability $\frac{1}{3}$. The solid trace and shaded region show the mean and confidence interval, respectively, computed similarly to Figure 6, for sequences of length $n = 1000$. (left) The solid trace shows the mean Tensor Embedding distance with $t = 3$, while the two dashed traces show the mean is between 1 and 1/2 slopes. (right) The solid trace shows the mean Tensor Sketch distance with $t = 3$ and $D = 8, 16, 32, 64$. The shaded regions correspond to increasing values of D, with the lightest shade corresponding to $D = 8$ and the darkest to $D = 64$.

In the following, we will compare and contrast our method with existing approaches, including discussing the strengths and weaknesses of each. Finally, we will explore potential avenues for future research in this exciting and rapidly-evolving field.

Suffix array data structures. Applications of suffix array data structures, such as Burrows-Wheeler transform (BWT) (Burrows 1994) and FM-INDEX (Ferragina and Manzini 2000) have improved the efficiency of sequence alignment and similarity searches by enabling the retrieval of matching substrings between a query and a sequence up to a fixed length k. The development of generalized suffix trees has extended this concept to sequence-to-graph alignment tools (Sović et al. 2016; Sirén 2017; Garrison et al. 2018; Minkin and Medvedev 2020), allowing them to construct an index of graph nodes to identify candidate matches for seed-and-extend. However, our work focuses on the high mutation regime, where these substring matching methods are inherently ill-suited, as we demonstrate in Example 1 and in our empirical evaluation presented in Results. In other words, while BWT and suffix array data structures have been transformative in sequence analysis, they struggle to handle highly mutated sequences. Our work presents a new approach that is optimized for these challenging scenarios.

Minimizers. Minimizers are a powerful technique for compressing sequence data without sacrificing crucial information. This approach has been used in many popular bioinformatics tools (H Li 2013; H Li 2016; H Li 2018). Another concept that builds on the idea of minimizers is the notion of syncmers (Edgar 2021), but modify the sampling criteria to ensure selection only depends on the k-mer content.

While we have not explored minimizers or syncmers in this work, using minimizers in MG-SKETCH can balance the trade-off between query processing time and accuracy. By replacing the k-cover in MG-SKETCH with a minimizer-based sampling criteria, and consequently fewer KNN queries and fewer attempted extensions. One natural minimizer criteria can be selecting a random orientation v in R^d and choosing the node with the minimum inner product value in a local neighborhood of the graph. This approach shows promise as a potential avenue for future research.

Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

Spaced seeds. Spaced seeds are a powerful tool for improving sensitivity in homology search and alignment, defined as a pattern of relevant and irrelevant positions (B Ma et al. 2002; Langmead and Salzberg 2012; Břinda et al. 2015). However, spaced seeds have two main limitations. First, they are not sensitive to indels. In our mutation model, where edit operations are equally likely, we expect to see approximately $\frac{2}{3}kr$ indels in every k-mer, which limits the effectiveness of spaced seeds when $k > \frac{3}{2r}$. The second shortcoming of spaced seeds is that using a single seed may increase the risk of no matches. To mitigate this risk, Pattern Hunter II proposed using a set of spaced seed patterns (M Li et al. 2004). However, finding an optimal set of spaced seeds is an NP-hard problem, and even finding a set of "good" spaced seeds is challenging (Choi et al. 2004; Choi and Zhang 2004). In the sequel, we discuss how sketching into a vector space can help circumvent this problem.

Indel-tolerant sketching. Sketching the edit distance remains a fundamental challenge for sequence analysis. To address this, several sketching methods have been developed, including Ordered MinHash (OMH) (Marçais et al. 2019), Strobemers (Sahlin 2021), and Tensor Sketching (Joudaki et al. 2021), all of which sketch non-contiguous parts of the string. In this context, it is informative to compare these methods and contrast their differences. In Strobemers of order t, the ℓ-mers of a sequence are considered in a series of non-overlapping windows, and t ℓ-mers that minimize a hash value are subsampled and concatenated. In OMH of order t, ℓ-mers of a sequence are sampled according to a total order and concatenated to produce the hash. The total order ensures that the t-tuple of ℓ-mers is uniformly drawn from the set of all ordered tuples.

It is worth noting that when $\ell = 1$, Tensor Sketch is highly similar to OMH and Strobemers, as all methods approximate the distance of ordered tuples between two strings. However, when $\ell > 1$, OMH and Strobemers become insensitive to higher mutation rates, as a single mutation can affect $2\ell - 1$ different ℓ-mers.

One advantage of Strobemers is that the distance between consecutive selected substrings has a uniform distribution under random sequence distribution, a property not guaranteed by OMH and Tensor Sketch. However, the fixed definition of windows in Strobemers relative to the starting position makes it more sensitive to frame-shifts compared to OMH and Tensor Sketch.

Future study of MG-Sketch

Sketching into R^d. Our work demonstrates the promising potential of sketching genomic data into vector spaces, a finding that has been supported by recent work in this area (Zhao et al. 2022). In light of these results, it may be helpful to broadly group sketching methods into "hash-based" and "vector-based" sketching schemes. While hashes provide a binary hit-or-miss result, vector spaces offer a more nuanced representation of genomic data.

Consider hash-based sketch $\phi_h(x) := (x_i)_{i \in h}$, where h is randomly drawn from $h \sim \{S \subset [n], |S| = m\}$, and vector-based sketch $\phi_v(x) = \sum_{i=1}^{n} v_i x_i$, where $v_1, \ldots, v_n \sim \{-1, 1\}$ are random sign hashes. If we randomly draw $x \sim \Sigma^n$ and mutate y by substituting each position with a probability of r, the expected hash-based similarity is $\mathbb{E}_h[\phi_h(x) = \phi_h(y)] = (1-r)^m$, while the expected vector-based sketch distance is $\mathbb{E}_v[(\phi_v(x) - \phi_v(y))^2] = r$.

We can see that $(1-r)^m$ is not sensitive to higher mutation rates when $r > 1/m$. Therefore, our work highlights the potential of vector-based sketches as a promising direction for future research in the genomics field. The benefits of vector sketches come at the expense of additional complexity in similarity search in high dimensions, known as curse of dimensionality, which will be discussed next.

Hardware optimization. Despite the curse of dimensionality, recent developments in search indices, such as HNSW (Malkov and Yashunin 2018), have made it possible to apply vector space searching to billions of samples in high dimensions. Vector representations are also more amenable to hardware-optimized operations, which is a potential advantage. Our approach utilizes FAISS (Johnson et al. 2019) and Tensor Sketching, both of which are optimized for GPUs, enabling efficient data parallelism. In fact, our CUDA-optimized Tensor Sketching implementation is 200× faster than the single-threaded CPU implementation (Groot Koerkamp 2021). By leveraging the CUDA-optimized implementations of FAISS and Tensor Sketching, we can take

Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

advantage of data parallelism on GPUs, resulting in improved performance and efficiency. Thus, hardware optimizations have become increasingly important in genomics research, and our approach takes advantage of the latest hardware innovations to deliver more efficient and effective results.

Seed filtering and chaining. There are also several areas for improvement that are beyond the scope of this work. For instance, we have not implemented any seed filtering techniques such as co-linear chaining (J Ma et al. 2022; Chandra and Jain 2022; Karasikov et al. 2022; Almodaresi et al. 2021; H Li et al. 2020). These approaches use a cut-off threshold rather than the number of neighbors for anchoring, which can reduce the number of anchors in sparse areas of the sketch space and decrease the number of false matches. Additionally, other subsampling methods such as spaced-minimized subsampling (Roberts et al. 2004; Vaddadi et al. 2019; Ekim et al. 2021) may provide additional benefits.

Theoretical guarantees. One desirable aspect of sketch-based methods is that they are randomized algorithms. Thus, one can improve recall through multiple independent runs and ensure theoretical guarantees such as those in Lemma 1. Further investigation of the guarantees of MG-Sketch may reveal additional insights, such as a probabilistic bound on deviations for Tensor Sketching and the inclusion of indels in the analysis.

Methods

Preliminaries

Terminology and notation. We use $[N] := \{1, \ldots, N\}$. Σ denotes the alphabet, e.g., the nucleotides {`A`,`C`,`G`,`T`}, or amino acids. For $k \in \mathbb{N}^+$, $\Sigma^\star$ denotes the set of all strings over Σ, and Σ^k denotes the subset of all strings with length k. Throughout the text, we use the terms string and sequence interchangeably to refer to members of $\Sigma^\star$. $q[i]$ and q_i denote the i-th character of the sequence, and $q[i:j] := q_i q_{i+1} \ldots q_j$ denotes the sliced substring from i, up to j. k-mers(s) denote the substrings of length k in string s: k-mers(s) := $\{s[i : i+k-1] : 1 \leq i \leq |s|-k+1\}$. $p \circ q$ denotes the concatenation of p and q. The *edit distance* ed(p, q), also referred to as the Levenshtein distance (Levenshtein et al. 1966), is defined as the minimum number of insertion, deletion, and substitution operations needed to transform one sequence into the other. The normalized edit distance, is defined as edit distance divided by maximum length $\widetilde{\text{ed}}(p, q) := \text{ed}(p, q)/\max(|p|, |q|)$. Throughout the manuscript, the term "mutation" refers to the normalized edit distance, serving as an abstraction for the combined biological sources of variation and errors in sequencing.

Succinct De Bruijn graph (DBG). The reference DBG is a directed graph over the reference sequences that encodes all substrings of length k as vertices and all substrings of length $k+1$ as directed edges. Formally, let $\mathcal{S} = \{r_1, \ldots, r_{|\mathcal{S}|}\}$ be the input sequences. The reference graph is a directed graph $G = (V, E)$, with vertices $V := \bigcup_{s \in \mathcal{S}} k\text{-mers}(s)$ and edges $E := \{(u, v, v[k]) \in V \times V \times \Sigma : u[2:k] = v[1:k-1]\}$. The label of edge $e = (u, v, v[k]) \in E$ is denoted by $l_e := v[k] \in \Sigma$.

Sequence-to-graph alignment. Define $\mathcal{W}_G$ as the set of all walks, where each walk $w \in \mathcal{W}_G$ is defined as a list of adjacent edges $w = \big((v_0, v_1, l_1), \ldots, (v_{|w|-1}, v_{|w|}, l_{|w|})\big) \in E^{|w|}$. Define the *spelling* of a walk as the concatenation of the first k-mer on the walk, with the labels of the rest of the edges on the same walk $\pi(w) := v_0 \circ l_1 l_2 \ldots l_{|w|}$. Thus, given the query sequence $q \in \Sigma^\star$ and the reference graph G, an optimal *alignment* belong to the set of walks which achieve the minimum edit distance between the query sequence and the spelling of the walk $\text{align}(q, G) := \text{argmin}_{w \in \mathcal{W}_G}\{\text{ed}(q, \pi(w)\}$. Note that $\text{align}(q, G)$ is defined as a set, rather than a single walk, since multiple walks can achieve the minimum value.

Estimating edit distance using Tensor Sketch.

Our approach to the seed extraction and anchoring steps of seed-and-extend (see Introduction) is based on an index of *k-mer sketches* rather than k-mers. Briefly, we compute Tensor Sketches (Joudaki et al. 2021)

Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

of the graph nodes and store them in a nearest-neighbor search index (Johnson et al. 2019) that maps each sketch to its corresponding graph node.

Tensor Sketching (TS) maps sequences to a vector space that embeds the edit distance into the ℓ^2 metric. Conceptually, TS can be described in two steps: i) Tensor Embedding (TE), which counts how many times each subsequence appears in the sequence, ii) Implicit tensor sketching, which lowers the dimension without constructing the larger tensor embedding space.

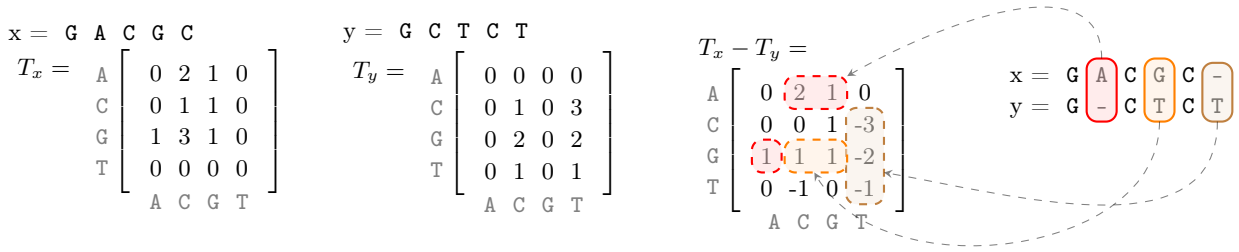


Fig. 8: Tensor Embedding illustration for $t = 2$. Observe that the substitution, insertion, and deletion, correspond to blocks of non-zero elements in the difference tensor.

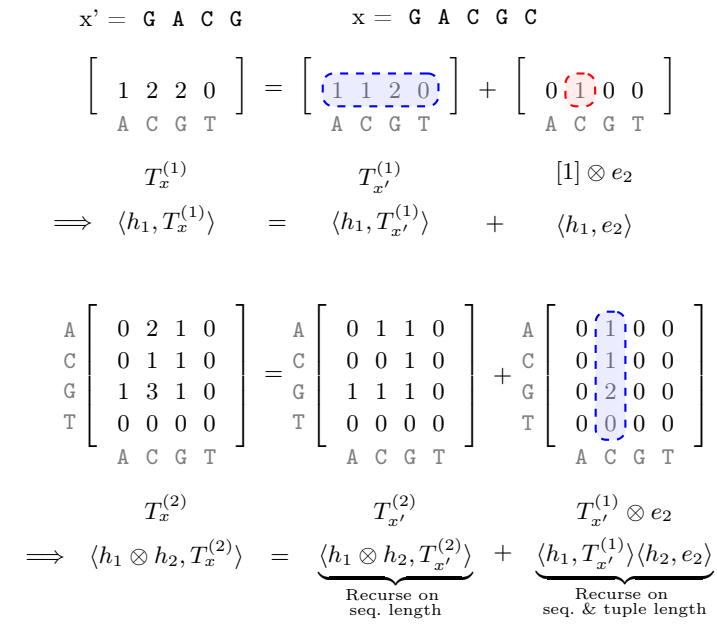


Fig. 9: Implicit sketching when the sketch tensor is rank-1. $T_x^{(t)}$ denotes the tensor embedding of order t: $T_x^{(t)}[a_{1\ldots t}] := \#\{i_1 < \cdots < i_t : x_{i_1 \ldots i_t} = a_{1\ldots t}\}$, and $h_1, h_2 \sim \{-1, 1\}^{|\Sigma|}$ denote sign hashes. $e_2 = [0, 1, 0, 0]$ is the indicator corresponding to the appended character "C".

Tensor Embedding. Given sequence $a \in \Sigma^n$, define $\mathcal{I}$ as set of increasing subsequences of length t: $\mathcal{I} := \{(i_1, \ldots, i_t) \in [n]^t : i_1 < \cdots < i_t\}$, and for all $s \in \Sigma^t$, define tensor embedding $T_a[s]$ as the number

Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

of occurrences of s as a subsequence in a: $T_a[s] := \#\{I \in \mathcal{I} : a[i_1, \ldots, i_t] = s\}$. We can view T_a as a $|\Sigma|^t$-dimensional tensor, with the alphabet as its indices. Given sequences a, b, we approximate the edit distance by $\|T_a - T_b\|_2^2$ up to a constant factor. Figure 8 illustrates how the embedding distance approximates edit operations.

Tensor Sketching. Tensor Sketching is an implicit, Euclidean-norm preserving dimensionality reduction scheme. Crucially, it projects $|\Sigma|^t$-dimensional tensors onto constant $D \in \mathbb{N}^+$ dimensions and linearly preserves ℓ^2 distances. We define the Tensor Sketching function $\phi : \mathbb{R}^{|\Sigma|^t} \to \mathbb{R}^D$, that embeds an $|\Sigma|^t$-dimensional tensor into $\mathbb{R}^D$. Given pairwise independent hash functions $h_i : \Sigma \to [D]$ and $s_i : \Sigma \to \{-1, 1\}$, define the tuple hash $H(A) := \sum_i^t h_i(a_i) \mod D$, and tuple sign hash $S(A) := \prod_i^t s_i(a_i)$, where $A = (a_i)_{i \le t} \in \Sigma^t$. Finally, the tensor sketch ϕ for a tensor $T \in \mathbb{R}^{|\Sigma|^t}$ is defined as $(\phi(T))_z := \sum_{A \in \Sigma^t : H(A) = z} S(A) T[A]$ for all $z \in [D]$.

Crucially, we have $\mathbb{E}\|\phi(T)\|_2^2 = \|T\|_2^2$ and bounded second moments $\mathrm{Var}(\phi(T)) \le \frac{\|T\|_2^2}{D}$ (See Lemma 7 of (Pham and Pagh 2013)). Figures 6 and 7 show that sketch distances approximation of embedding distances improve with respect to the sketch dimension D.

It is important to note that while computing $\phi(T)$ for a general tensor T takes $|\Sigma|^t$ time, when T is the tensor embedding of a string $T := T_x$, computing $\phi(T_x)$ can be carried out without ever constructing the tensor T_x. Figure 9 illustrates the main ideas behind "implicitly sketching" $\langle H, T_x \rangle = \sum_{A \in \Sigma^t} = H[A] T_x[A]$ when H is a rank-1 tensor. This illustrates that the embedding of x, which is constructed by appending a character to x', can be written as sum of the embedding of x', implying that we can compute the sketch recursively (see implicit sketching in Section 2.2 of (Joudaki et al. 2021) for details). Figures 6 and 7 show that the sketch distance approximation of embedding distance improves with respect to the sketch dimension D.

Finally, (Joudaki et al. 2021) reports that *Tensor Slide Sketch (TSS)*, which concatenates tensor sketches of windows w, using a stride Δ within a given k-mer, improves the sensitivity to smaller edit distances. Therefore, we used TSS for our seeding scheme (See Supplemental Material D for more details).

Anchoring seeds with a hierarchical search index.

The final pre-processing step is to build a k-nearest neighbor index of sketches. Conceptually, we can represent the search index as a function from vector space $\mathbb{R}^D$ to a list of graph nodes K-NN$(v) = (v_1, \ldots, v_K)$, for some pre-determined number of neighbors $K \in \mathbb{N}^+$. During the alignment, we anchor every seed in the query $s \in k$-mers(q), to the vertices returned by the index K-NN$(\phi(s))$.

Note that if v and u are within s steps on the graph, they share $\frac{k-s}{k}$ of their sequence content. To avoid indexing redundant information, we only store a k-cover of the graph, defined as a subset of vertices such that any walk on the graph with over k nodes contains at least one node in the k-cover. For any walk $W \in \mathcal{W}_G, |W| \ge k$, it holds that $W \cap k\mathrm{cover}(G) \neq \varnothing$. Observe that if the optimal alignment walk for a query has over k nodes, it suffices to sketch the k-cover to anchor at least one of its seeds.

While sketch vectors are more robust to mutations than exact seeds, retrieving nearest neighbors in a high dimensional space faces the *curse of dimensionality* (Köppen 2000). *Locality Sensitive Hashing (Indyk and Motwani 1998; Datar et al. 2004)* is the first method to get a constant approximation for the nearest neighbors problem, with a theoretically proven sub-linear time with respect to dataset size. However, to scale to billions of nodes in the genome graphs, it is crucial to store sketch vectors in a *Hierarchical Navigable Small Worlds* (HNSW) index (Malkov and Yashunin 2018). We use the efficient implementation of HNSW from Facebook AI Similarity Search (FAISS) (Johnson et al. 2019). The pseudo-code for the anchoring is presented in Algorithm 1.

Adjusting extension for misaligned anchors.

MetaGraph Align (MG-Align) follows a seed-and-extend approach, with a dynamic program to determine which path to take in the graph, producing a semi-global alignment. We made a few modifications to

Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

Algorithm 1: ANCHORING

Input : De Bruijn graph $G(V, E)$
Query $q \in \Sigma^\star$
Output : Anchors $A \subseteq k\text{-mers}(q) \times V^K$
Parameter: Neighbours $K \in \mathbb{N}^+$
$H \leftarrow \text{HNSW}()$
for v *in* k-COVER(G) **do**
  $\Phi \leftarrow \text{SKETCH}(v)$
  $H.\text{ADD}(\Phi)$
end
$A \leftarrow \{\}$
for s *in* k-MERS(q) **do**
  $\Phi \leftarrow \text{SKETCH}(s)$
  $A \leftarrow A \cup \{(a, H.\text{KNN}(\Phi))\}$
end
Return : A

adjust for misaligned anchors in the MG-SKETCH seeder. To demonstrate this issue, let $v_1, v_2, \ldots, v_M$ denote a list of adjacent k-mers on the graph, and let $s_1, \ldots, s_{|q|-k+1}$ denote seeds in the query $s_i := q[i : i+k-1]$. Let us assume that if s_i is anchored to v_j, the alignment will be optimal. Observe that $\text{ed}(s_i, s_{i+\delta}) \leq 2\delta$, obtained by deleting the initial δ characters from s_i and inserting the last δ characters of $v_{i+\delta}$. By the triangle inequality, if $\text{ed}(s_i, v_j) \leq d$, it holds that $\text{ed}(s_{i+\delta}, v_j) \leq d + 2\delta$. While Tensor Sketching preserves the edit distance, due to inherent stochasticity in sketching and retrieval, $s_{i+\delta}$ may be anchored to v_j, instead of s_i to v_j. This may produce an additional cost of 2δ during the forward and backward extension. To avoid this unnecessary cost, during the forward extension, indels occurring in the beginning of the query are free. If the forward extension completes, we initiate the backward extension from the position of the first matching positions.

Data access

The raw data used in this manuscript has been synthetically generated and is available at `https://doi.org/10.5281/zenodo.7769367`. The code to generate the synthetic data is available in Supplemental Code F. The data used in Figure 5 is an assembly of 51,283 virus genomes. The assembly and the individual acccession number of each viral genome in the assembly are available at `https://public.bmi.inf.ethz.ch/projects/2018/graph-anno/fasta/virus50000/`.

Software availability

The implementation of our method is available as open source software at `https://github.com/ratschlab/tensor-sketch-alignment/`. The code required to reproduce the experiments is available on the main repository, as well as in Supplemental Code F.

Competing interest statement

The authors declare no conflicts of interest.

Acknowledgements

A. J. is funded through Swiss National Science Foundation Project Grant #200550 to A. K. H. M. is funded as part of Swiss National Research Programme (NRP) 75 "Big Data" by the SNSF grant #407540_167331. A. J., H. M., and A. K. are also partially funded by ETH core funding (to G. R.). R.G. was funded through an ETH Research Grant (# ETH-17 21-1) to G.R.

Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

Author contributions: A.J. and A.M. came up with the conceptual and experimental design with help from H.M., A.M. implemented the method advised by H.M. and A.J., A.J. and R.G.K. proved Lemma 1., A.J. and A.M. drafted the manuscript with help from A.K., G.R., H.M., and R.G.K.

References

Almodaresi F, Zakeri M, and Patro R. 2021. PuffAligner: a fast, efficient and accurate aligner based on the Pufferfish index. *Bioinformatics* **37:** 4048–4055.

Altschul SF, Gish W, Miller W, Myers EW, and Lipman DJ. 1990. Basic local alignment search tool. *Journal of Molecular Biology* **215:** 403–410.

Belazzougui D and Cunial F 2019. Fully-Functional Bidirectional Burrows-Wheeler Indexes and Infinite-Order De Bruijn Graphs. In: *30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019)*. Ed. by N Pisanti and SP Pissis. Vol. 128. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 10:1–10:15.

Boucher C, Bowe A, Gagie T, Puglisi SJ, and Sadakane K 2015. Variable-Order de Bruijn Graphs. In: *2015 Data Compression Conference*, pp. 383–392.

Bowe A, Onodera T, Sadakane K, and Shibuya T 2012. Succinct de Bruijn graphs. In: *International workshop on algorithms in bioinformatics*. Springer, pp. 225–235.

Břinda K, Sykulski M, and Kucherov G. 2015. Spaced seeds improve k-mer-based metagenomic classification. *Bioinformatics* **31:** 3584–3592.

Burkhardt S and Kärkkäinen J. 2003. Better filtering with gapped q-grams. *Fundamenta informaticae* **56:** 51–70.

Burrows M. 1994. A block-sorting lossless data compression algorithm. *SRC Research Report, 124*.

Califano A and Rigoutsos I 1993. FLASH: A fast look-up algorithm for string homology. In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, pp. 353–359.

Chandra G and Jain C. 2022. Sequence to graph alignment using gap-sensitive co-linear chaining. *bioRxiv* doi: `10.1101/2022.08.29.505691`.

Chang X, Eizenga J, Novak AM, Sirén J, and Paten B. 2020. Distance indexing and seed clustering in sequence graphs. *Bioinformatics* **36:** i146–i153.

Choi KP, Zeng F, and Zhang L. 2004. Good spaced seeds for homology search. *Bioinformatics* **20:** 1053–1059.

Choi KP and Zhang L. 2004. Sensitivity analysis and efficient method for identifying optimal spaced seeds. *Journal of Computer and System Sciences* **68:** 22–40.

Datar M, Immorlica N, Indyk P, and Mirrokni VS 2004. Locality-sensitive hashing scheme based on p-stable distributions. In: *Proceedings of the twentieth annual symposium on Computational geometry*, pp. 253–262.

Edgar R. 2021. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ* **9:** e10805.

Eizenga JM, Lorig-Roach R, Meredith MM, and Paten B 2021. Walk-Preserving Transformation of Overlapped Sequence Graphs into Blunt Sequence Graphs with GetBlunted. In: *Connecting with Computability*. Ed. by L De Mol, A Weiermann, F Manea, and D Fernández-Duque. Cham: Springer International Publishing, pp. 169–177.

Ekim B, Berger B, and Chikhi R. 2021. Minimizer-space de Bruijn graphs: Whole-genome assembly of long reads in minutes on a personal computer. *Cell systems* **12:** 958–968.

Ferragina P and Manzini G 2000. Opportunistic data structures with applications. In: *Proceedings 41st annual symposium on foundations of computer science*. IEEE, pp. 390–398.

Garrison E, Sirén J, Novak AM, Hickey G, Eizenga JM, Dawson ET, Jones W, Garg S, Markello C, Lin MF, et al. 2018. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature biotechnology* **36:** 875–879.

Ghaffaari A and Marschall T. 2019. Fully-sensitive seed finding in sequence graphs using a hybrid index. *Bioinformatics* **35:** i81–i89.

Gibney D, Thankachan SV, and Aluru S 2022. The Complexity of Approximate Pattern Matching on de Bruijn Graphs. In: *Research in Computational Molecular Biology*. Ed. by I Pe'er. Cham: Springer International Publishing, pp. 263–278.

Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

Groot Koerkamp R 2021. 28000x speedup with Numba.CUDA. URL: https://curiouscoding.nl/phd/2021/03/24/numba-cuda-speedup/.

Hickey G, Heller D, Monlong J, Sibbesen JA, Sirén J, Eizenga J, Dawson ET, Garrison E, Novak AM, and Paten B. 2020. Genotyping structural variants in pangenome graphs using the vg toolkit. *Genome biology* **21:** 1–17.

Indyk P and Motwani R 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613.

Jain C, Zhang H, Gao Y, and Aluru S. 2020. On the Complexity of Sequence-to-Graph Alignment. *Journal of Computational Biology* **27:** 640–654.

Johnson J, Douze M, and Jégou H. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* **7:** 535–547.

Joudaki A, Rätsch G, and Kahles A. 2021. Fast Alignment-Free Similarity Estimation By Tensor Sketching. *bioRxiv* 2020–11.

Karasikov M, Mustafa H, Danciu D, Zimmermann M, Barber C, Rätsch G, and Kahles A. 2020. Metagraph: Indexing and analysing nucleotide archives at petabase-scale. *BioRxiv*.

Karasikov M, Mustafa H, Rätsch G, and Kahles A. 2022. Lossless indexing with counting de Bruijn graphs. *Genome Research*.

Khiste N and Ilie L. 2015. E-MEM: efficient computation of maximal exact matches for very large genomes. *Bioinformatics* **31:** 509–514.

Köppen M 2000. The curse of dimensionality. In: *5th online world conference on soft computing in industrial applications (WSC5)*. Vol. 1, pp. 4–8.

Langmead B and Salzberg SL. 2012. Fast gapped-read alignment with Bowtie 2. *Nature methods* **9:** 357–359.

Levenshtein VI et al. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In: *Soviet physics doklady*. Vol. 10. 8. Soviet Union, pp. 707–710.

Li H. 2013. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint arXiv:1303.3997*.

Li H. 2016. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics* **32:** 2103–2110.

Li H. 2018. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* **34:** 3094–3100.

Li H, Feng X, and Chu C. 2020. The design and construction of reference pangenome graphs with minigraph. *Genome Biology* **21:** 265.

Li M, Ma B, Kisman D, and Tromp J. 2004. PatternHunter II: Highly sensitive and fast homology search. *Journal of bioinformatics and computational biology* **2:** 417–439.

Limasset A, Cazaux B, Rivals E, and Peterlongo P. 2016. Read mapping on de Bruijn graphs. *BMC bioinformatics* **17:** 1–12.

Liu B, Guo H, Brudno M, and Wang Y. 2016. deBGA: read alignment with de Bruijn graph-based seed and extension. *Bioinformatics* **32:** 3224–3232.

Lyons R. 1998. A bird's-eye view of uniform spanning trees and forests. *Microsurveys in discrete probability* **41:** 135–162.

Ma B, Tromp J, and Li M. 2002. PatternHunter: faster and more sensitive homology search. *Bioinformatics* **18:** 440–445.

Ma J, Cáceres M, Salmela L, Mäkinen V, and Tomescu AI. 2022. Chaining for Accurate Alignment of Erroneous Long Reads to Acyclic Variation Graphs. *bioRxiv* doi: 10.1101/2022.01.07.475257.

Mak D, Gelfand Y, and Benson G. 2006. Indel seeds for homology search. *Bioinformatics* **22:** e341–e349.

Malkov YA and Yashunin DA. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* **42:** 824–836.

Marçais G, DeBlasio D, Pandey P, and Kingsford C. 2019. Locality-sensitive hashing for the edit distance. *Bioinformatics* **35:** i127–i135.

Minkin I and Medvedev P. 2020. Scalable multiple whole-genome alignment and locally collinear block construction with SibeliaZ. *Nature communications* **11:** 6327.

Mustafa H, Schilken I, Karasikov M, Eickhoff C, Rätsch G, and Kahles A. 2018. Dynamic compression schemes for graph coloring. *Bioinformatics* **35:** 407–414.

Downloaded from genome.cshlp.org on June 22, 2026 . Published by Cold Spring Harbor Laboratory Press

Pevzner PA and Waterman MS. 1995. Multiple filtration and approximate pattern matching. *Algorithmica* **13:** 135–154.

Pham N and Pagh R 2013. Fast and scalable polynomial kernels via explicit feature maps. In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 239–247.

Rautiainen M and Marschall T. 2020. GraphAligner: rapid and versatile sequence-to-graph alignment. *Genome biology* **21:** 1–28.

Roberts M, Hayes W, Hunt BR, Mount SM, and Yorke JA. 2004. Reducing storage requirements for biological sequence comparison. *Bioinformatics* **20:** 3363–3369.

Sahlin K. 2021. Effective sequence similarity detection with strobemers. *Genome research* **31:** 2080–2094.

Sibbesen JA, Eizenga JM, Novak AM, Sirén J, Chang X, Garrison E, and Paten B. 2023. Haplotype-aware pantranscriptome analyses using spliced pangenome graphs. *Nature Methods* 1–9.

Sirén J 2017. Indexing variation graphs. In: *2017 Proceedings of the ninteenth workshop on algorithm engineering and experiments (ALENEX)*. SIAM, pp. 13–27.

Sović I, Šikić M, Wilm A, Fenlon SN, Chen S, and Nagarajan N. 2016. Fast and sensitive mapping of nanopore sequencing reads with GraphMap. *Nature communications* **7:** 11307.

Vaddadi K, Srinivasan R, and Sivadasan N 2019. Read mapping on genome variation graphs. In: *19th International Workshop on Algorithms in Bioinformatics (WABI 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Zhao J, Both JP, Rodriguez-R LM, and Konstantinidis KT. 2022. GSearch: Ultra-Fast and Scalable Microbial Genome Search by combining Kmer Hashing with Hierarchical Navigable Small World Graphs. *bioRxiv* 2022–10.