`string2string Studio`: An Interactive, In-Browser Platform for String-to-String Algorithms

Mirac Suzgun
Stanford University

James Zou
Stanford University

Stuart M. Shieber
Harvard University

Dan Jurafsky
Stanford University

Abstract

We present **`string2string Studio`**, an interactive in-browser platform for string-to-string analysis across natural language processing, computational biology, and the digital humanities. The system integrates six main modules (alignment, distance, similarity, search, generation metrics, and BLAST homology search), operating at character, word, token, line, and residue levels. Its C++-based algorithms compile to WebAssembly, so core operations run locally by default without any installation or data upload. The interface reports scores with their "evidence" (alignments, edit paths, metric matches, search hits, and homology traces), making methods inspectable, debuggable, and comparable on shared inputs. Internal benchmarks show speedups of up to $2{,}500\times$ over the Python predecessor, faster global/local alignment than a general-purpose native C aligner, and exact agreement with independent references under declared settings. For homology search, the scoped client-side `blastn` path closely matches NCBI BLAST+ rankings and statistics under matched parameters. A curated showcase and *Learn* mode present canonical algorithms and metrics as reusable demonstrations. `string2string Studio` is open-source and freely available at `string2string.org`.[1]

1 Introduction

String-to-string algorithms appear across many areas of computing. In natural language processing, they underlie spelling correction, near-duplicate and training-data contamination detection, and the evaluation of machine-translated and other generated texts. In computational biology, they align sequences, search for chromoses , and reconstruct phylogenies. In the digital humanities, they collate manuscript contents and track how texts change across editions and translations. While these applications differ in data type, scale, and convention, they all rely on a common set of operations: aligning strings, measuring differences, computing similarity, searching for exact/approximate matches, and scoring sequence-level outputs (Levenshtein, 1966; Needleman and Wunsch, 1970; Gusfield, 1997; Jurafsky and Martin, 2025, *inter alia*).

Despite this shared foundation, the tools for these operations, however, remain mostly scattered. Some tools are distributed as programming libraries; others as command-line tools, web services, or domain-specific applications. Many are pretty powerful, but they often make it difficult to compare methods on the same inputs, inspect intermediate structures, adjust parameters interactively, or move between textual and biological use cases. Our previous `string2string` library (Suzgun et al., 2024) sought to address part of this fragmentation by collecting many string-to-string algorithms under a unified Python API. But a typical library is not an interactive environment: it does not, on its own, let users see an alignment, inspect an edit path, compare metrics, export a visualization, or run analyses locally in the browser.

Here, we present `string2string Studio`, an interactive browser-based platform for string-to-string analysis (Figure 1). The system is organized around six modules: (a) *alignment*, (b) *distance*, (c) *similarity*, (d) *search*, (e) *metrics*, and (f) BLAST *homology search*. It supports multiple granularities, including character, word, token, line, and residue level. Its computational engine is implemented in C++ and compiled to WebAssembly (Haas et al., 2017); thus, the main operations run on the client, with no software to install and no data leaving the machine by default. The interface presents outputs as linked, inspectable, and exportable objects, enabling users to compare methods on the same inputs and examine the structure behind the scores.

This work makes four contributions. *First*, we in-

[1] Correspondence to: `msuzgun@stanford.edu`.

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 435–444
October 24-29, 2026 ©2026 Association for Computational Linguistics

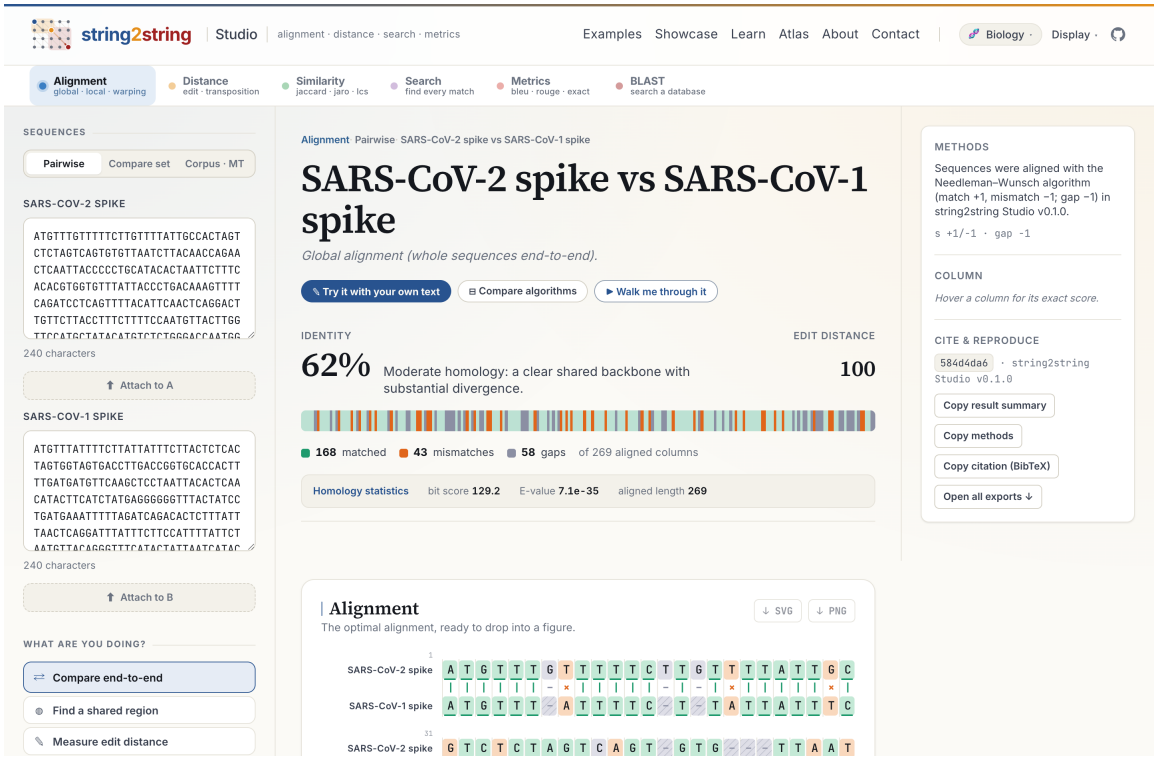

Figure 1: The `string2string Studio` workbench. The algorithm family is selectable across the top (alignment, distance, similarity, search, metrics, BLAST). Here, we illustrate the global alignment of a pair of coronavirus spike genes: the header reports identity and edit distance, and the identity track marks each of the 269 aligned columns as a match, substitution, or gap. The right panel records the method, citation, and export options; the interface adapts defaults and terminology to the user's field. Core computations run locally in the browser by default.

troduce, to our knowledge, the first browser-based workbench that integrates these methods in a single interactive environment without installation or programming, and that makes their outputs entirely inspectable—exposing alignments, edit paths, metric matches, search hits, and homology-search traces as linked visual objects rather than opaque scalars. *Second*, we describe a C++/WebAssembly engine for interactive client-side use that runs 100–2,500 times faster than its Python predecessor, outpaces a general-purpose native C aligner on global and local alignment, and agrees exactly with independent reference implementations. *Third*, we add a scoped client-side `blastn` path for homology search and evaluate its statistics against NCBI BLAST+ under matched parameters. *Fourth*, we provide a curated showcase and illustrations that packages classical and contemporary methods as executable, shareable illustrations for teaching, debugging, and cross-domain comparison (Figure 3).

2 Related Work

Our platform draws on several families of established tools & platforms. General-purpose libraries, including our predecessor (Suzgun et al., 2024), `textdistance`, `jellyfish`, and `RapidFuzz`, and optimized kernels, including `edlib` (Šošić and Šikić, 2017), SSW (Zhao et al., 2013), `parasail` (Daily, 2016), and `Biopython`'s aligner (Cock et al., 2009), provide implementations through programming interfaces. Domain-tailored systems naturally serve more specialized needs: NCBI BLAST (Altschul et al., 1990; Camacho et al., 2009) and EMBOSS for sequence search, Jalview (Waterhouse et al., 2009) for alignment display and editing, CollateX and Juxta for manuscript collation, and sacreBLEU (Post, 2018), jiwer, and Hugging Face `evaluate` for generation metrics. Interactive analysis systems in NLP, including VizSeq (Wang et al., 2019), compare-mt (Neubig et al., 2019), ExplainaBoard (Liu et al., 2021), MT-ComparEval (Klejch et al., 2015), MT-Telescope (Rei et al., 2021), the Language Interpretability Tool (Tenney et al., 2020), BERTViz (Vig, 2019), exBERT (Hoover et al., 2020), Seq2Seq-Vis (Strobelt et al., 2019), and the Annotated Transformer (Rush, 2018), show the value of making model and metric behavior visible. Browser-based scientific computation has recent precedent in bioinformatics (Akmal, 2019; Priyam et al., 2019; Ji et al., 2024).

System	Breadth	Multi-level	Interactive	Cross-method	Verified	In-browser	Speed
`string2string` library (Suzgun et al., 2024)	●	●	○				○
`edlib` (Šošić and Šikić, 2017)							●
`RapidFuzz`		○					●
Biopython aligner (Cock et al., 2009)	○						○
NCBI BLAST (Camacho et al., 2009)	○		○		●	○	●
sacreBLEU (Post, 2018)		○			●		●
Jalview (Waterhouse et al., 2009)	○		●			○	
compare-mt / VizSeq (Neubig et al., 2019; Wang et al., 2019)		○	●	○		○	
`string2string Studio` (this work)	●	●	●	●	●	●	○

Table 1: Representative tools by property (● present, ○ partial, blank absent). **Breadth**: spans several classes of the family (alignment, distance, similarity, search). **Multi-level**: runs at the character, word, token, and line level (●) or a subset (○). **Interactive**: results as interactive figures (●) or static plots/reports (○). **Cross-method**: several methods on one input in one interface. **Verified**: is, or is validated bit-for-bit against, a recognized reference. **In-browser**: computes client-side with no install (●), or a browser build is a transpiled port or server-backed (○). **Speed**: competitive with native code. `edlib` and `RapidFuzz` are faster on single primitives and NCBI BLAST searches at database scale; `string2string Studio` is alone in combining breadth and multi-level operation with a verified, cross-method, client-side engine.

The gap is *not* the absence of algorithms but the absence of a comprehensive environment around them: the fastest tools are usually narrow primitives, broad tools are libraries, and interactive tools are usually tied to one domain or backed by a server. Browser genomics projects show that native bioinformatics software can run on the web, but they expose genomics command-line tools rather than a cross-domain workbench, and systems like SequenceServer (Priyam et al., 2019) keep the computation server-side. `string2string Studio` thus targets a different point in its design space: a verified, client-side environment in which these heterogeneous algorithms are executable, inspectable, and shareable as reusable illustrations rather than isolated library calls or opaque scores (Table 1).

3 The Platform: `string2string Studio`

We have organized our Studio around six modules. The ***alignment*** module includes global, local, semi-global, affine-gap, linear-space Hirschberg (Hirschberg, 1975), banded, and dynamic-time-warping alignments.[2] The ***distance*** module includes Levenshtein, Damerau-Levenshtein, Hamming, Jaro-Winkler, and longest-common-subsequence distances. The ***similarity*** and ***search*** modules cover set- and vector-based similarity, exact and approximate lexical search, k-mismatch search, IUPAC-degenerate search, and both-strand sequence search. The ***metrics*** module implements generation-evaluation measures, including BLEU (Papineni et al., 2002), chrF/chrF++ (Popović, 2015), and ROUGE-1/2/L (Lin, 2004), with single- and multi-reference inputs and paired-bootstrap significance. The ***homology*** module implements the client-side BLAST workflow described in § 6. Following `string2string` (Suzgun et al., 2024), these run at multiple levels (e.g., character, word, token, line, and residue), so the same interface can compare token sequences, prose passages, biological sequences, and textual witnesses.[3]

The modules share one result schema. A computation returns the input level, parameters, scalar summaries, and the evidence behind those summaries: an alignment and path for alignment algorithms, an edit script for distances, matched spans for search, sufficient statistics for corpus metrics, or seeds, extensions, bit-scores, and E-values for homology search. This schema was a core design choice behind the interface. Because every result has the same basic shape (that is, score, evidence, parameters, and exports), the platform can easily compare several methods on one input, attach each number to the structure that produced it, reproduce a result from a link, and validate outputs against independent implementations.

The platform runs *locally* by default: algorithms execute in the browser, so users install nothing and their text or sequences stay on the machine unless they choose a remote database search. Local execution also makes interaction practical: changing a tokenization level, gap penalty, scoring scheme, or metric option recomputes the result without a server round-trip. The interface adapts defaults, terminology, and examples to the user's field, but the same methods remain available across fields.

Our showcase is dynamic: each curated example

[2] We are currently expanding both this module and others; we plan to integrate additional algorithms, including our own $\mathtt{RLACS}_k$ algorithm (Suzgun and Shieber, 2026), soon.

[3] Table 3 lists the implemented methods, levels, and views.

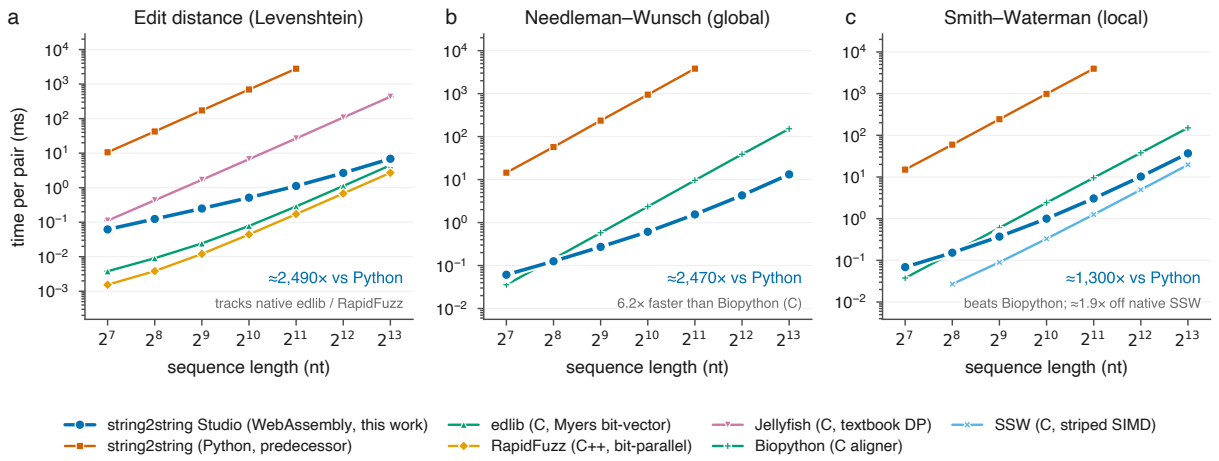

Figure 2: Runtime vs. sequence length (log–log; medians over 30 random pairs per length, Apple M1 Pro). Each panel covers one algorithm: edit distance (left), global alignment (center), and local alignment (right). Our Python predecessor (Suzgun et al., 2024) was measurable only to length 2^{11}; the annotated factors are read at length 2048. For global and local alignment, our C++/WebAssembly engine is faster than Biopython's general-purpose C aligner and within about a factor of two of the SIMD-C SSW library, with identical scores.

stores the inputs, level, parameters, result evidence, citation, and export profile needed to reproduce a computation, and starts from a concrete task—near-duplicate and plagiarism detection, translation and textual-witness comparison, text-generation evaluation, or genome comparison (e.g., the SARS-COV-1 and SARS-COV-2 spike genes)—so that users can change assumptions or parameters and watch the evidence change. A "publication" feature renders results as self-contained figures with citations and exports, so the same object can be inspected in the browser, shared by link, or moved into an article.

4 Engine

While designing our Studio, we sought to satisfy four factors at once: *precision*, *breadth*, *efficiency*, and *locality*. For instance, a server wrapper around our Python library would simplify engineering but weaken privacy and make interaction depend on network latency; similarly, a pure TypeScript implementation would be easier to inspect but too slow for long alignments and repeated recomputation. We separate the system into a typed algorithmic core, a browser dispatcher that selects the fastest available backend, and view components that consume the same result objects across domains.

Implementation. The core algorithms are reimplemented in C++ and compiled to two WebAssembly builds: a 128-bit SIMD build and a scalar fallback, selected at load time by feature detection. A separate TypeScript implementation follows the same specifications and serves both as documentation and as an independent reference. The engine aligns inputs too large for the full quadratic-space matrix with linear-space Hirschberg in a background worker, preserving optimality while keeping the interface responsive. The engine returns typed results rather than display-specific strings, so the same output can be rendered as an alignment, edit path, table, chart, export file, or record.

Speed. We benchmark our engine against the Python `string2string` library and specialized native libraries; all measurements are medians over 30 random sequence pairs per length on an Apple M1 Pro (ARM64). Against the Python predecessor, our engine is 100 to 2500 times faster, with the largest measured speedups at length 2048: roughly 2500× for edit distance and global alignment, and 1300× for local alignment. *See* Figure 2. These gains reflect both implementation and algorithmic changes, including bit-parallel (Myers, 1999) and SIMD kernels. Against native C baselines, our striped Smith–Waterman returns identical scores to SSW (Zhao et al., 2013) and runs within a factor of 1.8 at length 16,384, while our anti-diagonal SIMD Needleman-Wunsch is 6.2× faster than Biopython's general-purpose C aligner at length 2048.[4]

Space & verification. The full DP (dynamic programming) matrix, needed only when the user asks

[4] These comparisons are deliberately scoped. Specialized bit-parallel libraries such as `edlib` and `RapidFuzz` remain faster for edit distance, and we did not obtain a native SIMD-C global aligner on ARM64 because `parasail` did not compile. Our claim is therefore not that `string2string Studio` is the fastest implementation of every primitive, but that a broad, zero-install browser engine can be fast enough for interactive analysis while remaining close to optimized native code on representative alignment tasks.

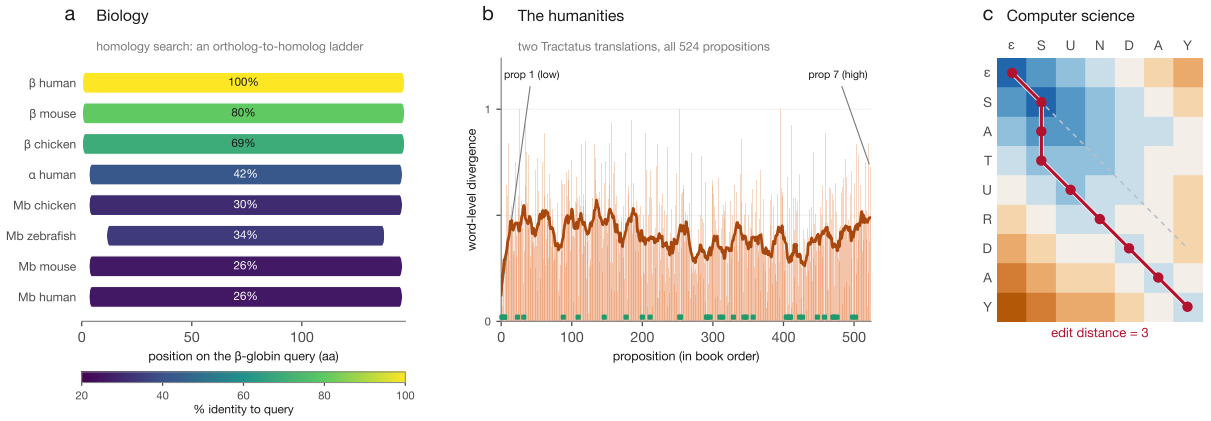

Figure 3: The same verified engine, rendered in the idiom of three disciplines. (a) Biology: a human β-globin query against a bundled landmark database returns a graded homology ladder—orthologs (mouse, chicken), the α-globin paralog, and distant myoglobin homologs—each bar colored by % identity. (b) Humanities: word-level divergence between two English *Tractatus* translations across all 524 propositions (faint bars) with a rolling-mean trend (line); the 35 identical propositions (7%) are marked green. (c) Computer science: the edit-distance cost field with the optimal path traced through it, each off-diagonal step an insertion or deletion.

to inspect the whole table, exceeds the 2GB WebAssembly heap at length 12K. Linear-space methods return the same optimum to length 40K, though arbitrarily very long inputs may take seconds rather than feel instantaneous. Verification follows the schema described above: each algorithm is checked against the `string2string` library and the TypeScript reference at char- and token-levels; SIMD kernels are checked against scalar kernels over parameter grids; generation metrics are checked against their named reference implementations; and the `blastn` path is checked against NCBI BLAST+ under matched settings (§7).

5 Visualization, Teaching, and Interaction

The Studio treats visualization as main part of the offerings, not as a second-thought. Each main result is rendered as a set of linked views (Figure 3). Selecting a column or span in one view highlights the corresponding evidence in the others and in a docked inspector; changing a gap penalty or tokenization level re-flows the result in place; zooming moves from an overview to individual symbols; and comparison views place several methods on the same input (for example, Levenshtein, Damerau–Levenshtein, and global alignment side by side). These interactions depend on the engine's local recomputation and on the shared result schema.

The same computation can be rendered in different disciplinary idioms. An alignment can appear as a per-column identity track (Figure 1), a dot plot, a synteny ribbon, an edit script, a version diff, a six-frame translation for coding sequence, a critical apparatus, or the full dynamic-programming matrix with the optimal path. For text-generation evaluation, `string2string Studio` computes corpus-level BLEU and chrF from summed sufficient statistics, allowing corpus-score differences to be attributed to individual segments without re-scoring subsets; a segment can then be opened to the matches and edits the metric rewards or penalizes, with a paired-bootstrap confidence interval on the system-level difference (Koehn, 2004). The point is not to replace expert judgment with a score, but to make the score more auditable.

Every view exports to the formats its audience might expect: domain interchange formats such as FASTA, CLUSTAL, CIGAR, PAF, VCF, and a critical apparatus, as well as PDF, CSV, JSON, SVG, and LaTeX. Each figure also carries a reproducible link containing the relevant inputs and parameters. The same mechanism supports the public showcase and the "*Learn*" mode: users can step through the dynamic program behind edit distance or alignment, alter the cost model, and see how the matrix, path, alignment, and final score change together.

6 Homology Search in the Browser

The original `string2string` paper originally identified BLAST support as future work (Suzgun et al., 2024). We are very delighted to report that `string2string Studio` finally delivers it, adding a scoped homology-search path while preserving the platform's default locality. For a query sequence, the engine follows the familiar seed-and-extend structure of BLAST: it indexes short seeds,

finds candidate seed hits, extends them into local alignments, and reports bit-scores and E-values computed from Karlin-Altschul statistics (Karlin and Altschul, 1990; Altschul et al., 1990, 1997; Camacho et al., 2009). Because the output uses the same result schema as the rest of the platform, the user can inspect not only the final ranked hits but also the seeds, extensions, scores, parameters, and aligned regions behind them.

The module currently supports three modes. Searches can run against bundled/customized landmark databases, against a user-supplied database loaded locally, or, when the user explicitly chooses it, against remote NCBI. Local search lets a user screen an unpublished or potentially sensitive sequence without transmitting it; the optional remote path is appropriate when the relevant database is too large to ship with the browser. The result view mirrors familiar biological reports (a ranked hit table, per-query hit distribution, conservation skyline, and aligned hit ladder) while adding linked inspection and export. Figure 4 shows the browser report for a human β-globin query.

We also note that `string2string Studio` does not replace NCBI BLAST+ at database scale, and we do not claim full parity across all BLAST tasks, databases, or scoring regimes; the module's role here is to bring local, inspectable homology search into the same cross-domain workbench as the other methods, with its `blastn` statistics evaluated in § 7.

7 Evaluation

In every stage of our implementation, we have carefully tested our algorithms in terms of time & space against other established frameworks and libraries. Here, we briefly discuss our evaluation criteria and results. We shall remark that our engine is fast, efficient, and reliable for real-time interactions; that our results match declared references; and that our client-side `blastn` statistics overall agree with NCBI BLAST+ under our setting.

Runtime and memory. § 4 and Figure 2 compare runtime against the Python `string2string` library and against `edlib`, `RapidFuzz`, Biopython, and native SSW across lengths 2^7 to 2^{13}. All systems use the same inputs, hardware, and repetition protocol, and baselines are called through their recommended interfaces. We also report the input length at which the full dynamic-programming matrix exhausts the WebAssembly heap, because displaying the matrix is a stricter memory requirement than returning only the optimal score or alignment.

Reference agreement. Every core algorithm is compared against two independent implementations, namely the `string2string` library and the TypeScript reference, at both character and token levels. *See also* Appendix A. The generation metrics are checked against the reference named in the interface: BLEU against NLTK method 1, chrF/chrF++ against sacreBLEU, and ROUGE-1/2/L against `rouge-score`. Under the stated settings, the differences are exactly 0. We therefore report agreement with declared implementations and configurations; the tokenization and smoothing signature are recorded for reproducibility (Post, 2018).

BLAST concordance. For homology search, we ran the client-side `blastn` implementation against NCBI BLAST+ 2.17 on a six-sequence 16S rRNA database, using identical query sequences and classic-`blastn` parameters. The hit ranking is identical across the two systems; bit-scores differ by at most 0.43%; percent identity differs by at most 0.18 points; and primary-HSP E-values agree to the same order of magnitude, with differences expected from implementation choices and numerical flooring for extremely small values. Appendix A gives the database, queries, hits, and parameters.

8 Limitations

We highlight five main limitations. First, browser memory constrains the largest inspectable dynamic programs: the full-matrix view is unavailable beyond length $12{,}000$, though linear-space methods return the same optimum on longer inputs. Second, the metric suite is deliberately lexical and symbolic. Neural and semantic metrics such as BERTSCORE are outside the present local, zero-installation client. We hope to deploy neural-based approaches in the future. Third, the homology-search evaluation is intentionally narrow: bundled databases are illustrative, large-scale database search is delegated to an optional remote call, and only `blastn` statistics are validated against NCBI BLAST+ here. Fourth, the showcase and *Learn* mode are useful and insightful design contributions, but we have not evaluated learning interventions and efficacy: we have not run a formal user study and make no measured claim about learning gains, usability, or analyst productivity. Fifth, the examples in our Studio are weighted toward English and Latin-script text for now, and the runtime benchmark uses random sequence pairs; broader multilingual and similarity-stratified evaluations remain future work.

Acknowledgments

Our initial string2string library owes a debt of gratitude to the following individuals for their contributions, comments, and feedback: Federico Bianchi, Corinna Coupette, Sebastian Gehrmann, Tayfun Gür, Şule Kahraman, Deniz Keleş, Luke Melas-Kyriazi, Christopher Manning, Tolúlopé Ògúnrèmí, Alexander "Sasha" Rush, Kyle Swanson, and Garrett Tanzer. We shall also thank Deniz Keleş, Kyle Swanson, and Filippos Sytilidis, as well as the anonymous conference reviewers, for their helpful comments and suggestions. Suzgun also wishes to thank Sebastian Gehrmann, whose conversations and insights on HCI design principles many years ago helped shape the design and organization of the website. Suzgun gratefully acknowledges the support of an Google PhD Fellowship.

Ethics Statement

By default, all the computations performed are local: analyzed text and sequences remain on the user's machine unless the user explicitly invokes a remote database search. This design is important for unpublished biological sequences, sensitive corpora, and classroom or research settings where users may not have permission to upload data.

BLAST is a trademark of the National Library of Medicine (NLM). Our implementation is independent and not endorsed by NCBI, the optional remote path is opt-in and attributed, and we cite the algorithm and statistics it reproduces (Altschul et al., 1990; Karlin and Altschul, 1990; Camacho et al., 2009). Any usage analytics are limited and disclosed. The Studio is released under a permissive open-source license.

Disclaimer

Portions of this manuscript were edited with the assistance of AI chatbots, used primarily to condense selected passages and improve the overall clarity, style, and flow. Every model-generated suggestion was ultimately reviewed by the authors, and only revisions limited to the word and sentence level were adopted; all substantive content, claims, and analyses remain the authors' sole responsibility.

We also disclose the use of Claude Code as a programming assistant in developing our new codebase and re-implementing algorithms from our earlier Python-based `string2string` library (Suzgun et al., 2024).

References

Robert Akmal. 2019. biowasm: WebAssembly modules for genomics. https://biowasm.com.

Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. 1990. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410.

Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman. 1997. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402.

Christiam Camacho, George Coulouris, Vahram Avagyan, Ning Ma, Jason Papadopoulos, Kevin Bealer, and Thomas L. Madden. 2009. BLAST+: architecture and applications. *BMC Bioinformatics*, 10(1):421.

Peter J. A. Cock, Tiago Antao, Jeffrey T. Chang, et al. 2009. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423.

Jeff Daily. 2016. Parasail: SIMD C library for global, semi-global, and local pairwise sequence alignments. *BMC Bioinformatics*, 17(1):81.

Dan Gusfield. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press.

Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. 2017. Bringing the web up to speed with WebAssembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 185–200.

Daniel S. Hirschberg. 1975. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343.

Benjamin Hoover, Hendrik Strobelt, and Sebastian Gehrmann. 2020. exBERT: A visual analysis tool to explore learned representations in transformer models. pages 187–196.

Daniel Ji, Robert Aboukhalil, and Niema Moshiri. 2024. ViralWasm: a client-side user-friendly web application suite for viral genomics. *Bioinformatics*, 40(1):btae018.

Daniel Jurafsky and James H. Martin. 2025. *Speech and Language Processing*, 3rd (draft) edition. Online manuscript released January 6, 2026.

Samuel Karlin and Stephen F. Altschul. 1990. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Sciences*, 87(6):2264–2268.

Ondřej Klejch, Eleftherios Avramidis, Aljoscha Burchardt, and Martin Popel. 2015. MT-ComparEval: Graphical evaluation interface for machine translation development. *The Prague Bulletin of Mathematical Linguistics*, (104):63–74.

Philipp Koehn. 2004. Statistical significance tests for machine translation evaluation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 388–395.

Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81.

Pengfei Liu, Jinlan Fu, Yang Xiao, Weizhe Yuan, Shuaichen Chang, Junqi Dai, Yixin Liu, Zihuiwen Ye, and Graham Neubig. 2021. ExplainaBoard: An explainable leaderboard for NLP. In *Proceedings of the 59th Annual Meeting of the ACL (System Demonstrations)*, pages 280–289.

Gene Myers. 1999. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM*, 46(3):395–415.

Saul B. Needleman and Christian D. Wunsch. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453.

Graham Neubig, Zi-Yi Dou, Junjie Hu, Paul Michel, Danish Pruthi, and Xinyi Wang. 2019. compare-mt: A tool for holistic comparison of language generation systems. In *Proceedings of the 2019 Conference of the North American Chapter of the ACL (Demonstrations)*, pages 35–41.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318.

Maja Popović. 2015. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation (WMT)*, pages 186–191.

Anurag Priyam, Ben J. Woodcroft, Vivek Rai, et al. 2019. Sequenceserver: a modern graphical user interface for custom BLAST databases. *Molecular Biology and Evolution*, 36(12):2922–2924.

Ricardo Rei, Ana C Farinha, Craig Stewart, Luisa Coheur, and Alon Lavie. 2021. MT-Telescope: An interactive platform for contrastive evaluation of machine translation systems. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 73–80.

Alexander M. Rush. 2018. The annotated transformer.

Martin Šošić and Mile Šikić. 2017. Edlib: a C/C++ library for fast, exact sequence alignment using edit distance. *Bioinformatics*, 33(9):1394–1395.

Hendrik Strobelt, Sebastian Gehrmann, Michael Behrisch, Adam Perer, Hanspeter Pfister, and Alexander M. Rush. 2019. Seq2Seq-Vis: A visual debugging tool for sequence-to-sequence models. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):353–363.

Mirac Suzgun, Stuart Shieber, and Dan Jurafsky. 2024. string2string: A modern python library for string-to-string algorithms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 278–285, Bangkok, Thailand. Association for Computational Linguistics.

Mirac Suzgun and Stuart M. Shieber. 2026. Efficient approximate string matching: The restricted longest approximately common substring problem with k-edits. Forthcoming.

Ian Tenney, James Wexler, Jasmijn Bastings, Tolga Bolukbasi, Andy Coenen, Sebastian Gehrmann, Ellen Jiang, Mahima Pushkarna, Carey Radebaugh, Emily Reif, and Ann Yuan. 2020. The language interpretability tool: Extensible, interactive visualizations and analysis for NLP models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (System Demonstrations)*, pages 107–118.

Jesse Vig. 2019. A multiscale visualization of attention in the transformer model. In *Proceedings of the 57th Annual Meeting of the ACL (System Demonstrations)*, pages 37–42.

Changhan Wang, Anirudh Jain, Danlu Chen, and Jiatao Gu. 2019. VizSeq: a visual analysis toolkit for text generation tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (System Demonstrations)*, pages 253–258.

Andrew M. Waterhouse, James B. Procter, David M. A. Martin, Michele Clamp, and Geoffrey J. Barton. 2009. Jalview version 2: a multiple sequence alignment editor and analysis workbench. *Bioinformatics*, 25(9):1189–1191.

Mengyao Zhao, Wan-Ping Lee, Erik P. Garrison, and Gabor T. Marth. 2013. SSW library: an SIMD Smith-Waterman C/C++ library for use in genomic applications. *PLoS ONE*, 8(12):e82138.

A Supplementary Figures and Tables

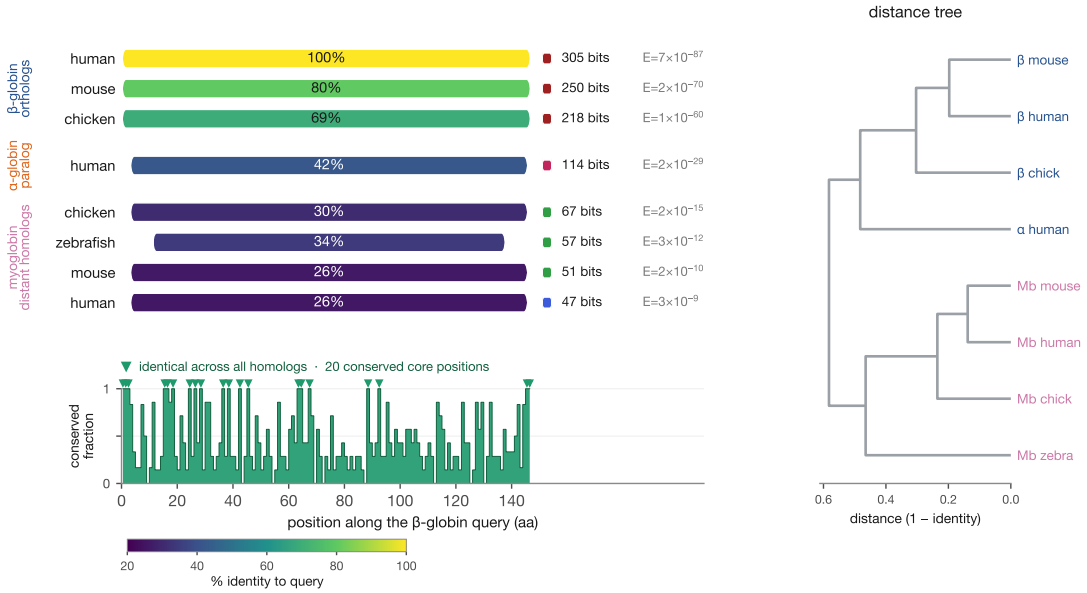

Figure 4: *Homology search in depth.* One human β-globin query, searched entirely in the browser against a 29-sequence landmark database, recovers the globin superfamily as a graded ladder: β-globin orthologs (mouse 80%, chicken 69% identity), the α-globin paralog (42%), and distant myoglobin homologs (26–34%), each bar coloured by percent identity and annotated with its bit-score and E-value. Below, the per-residue conservation skyline marks the 20 positions identical across all homologs; at right, the distance tree built from the same alignments recovers the globin and myoglobin clades. The report exposes the Karlin–Altschul parameters (λ, K) and E-values; quantitative concordance with NCBI BLAST+ is evaluated for `blastn` in Figure 6.

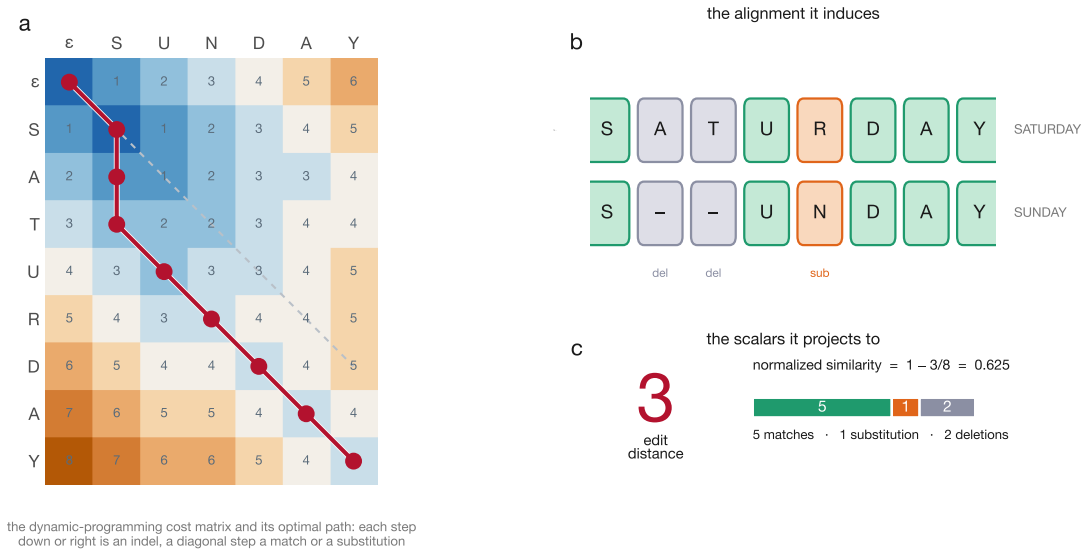

Figure 5: *One alignment, three lenses.* A reported edit distance is a projection of a larger object. For `SATURDAY` vs `SUNDAY`, the same computation is shown as (a) the dynamic-programming cost matrix with the optimal path traced through it, (b) the alignment that path induces, column by column (five matches, one substitution, two deletions), and (c) the scalars it projects to, including the edit distance of 3. Every metric in the suite is one such projection.

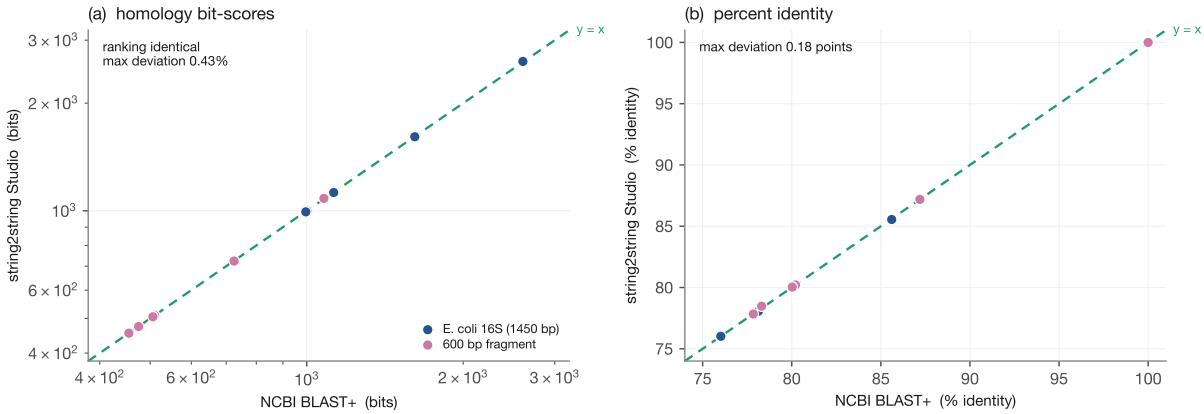

Figure 6: *Client-side* `blastn` *is faithful under the evaluated setting.* Our `blastn` run is compared with NCBI BLAST+ 2.17 on a real 16S rRNA database with identical parameters: (a) bit-scores and (b) percent identity for every hit lie on the line of equality (ranking is identical; bit-scores agree within 0.43% and identity within 0.18 points). The rest of the engine's exact agreement with its reference implementations (NLTK, sacreBLEU, `rouge-score`, and bit-exact parity with the `string2string` library) is reported in § 7.

	`string2string` library (Suzgun et al., 2024)	**`string2string Studio`** (this work)
Interface	Python API / command line	interactive web application, no code to write
Users	programmers	also non-programmers, across NLP, biology, and the humanities
Deployment	`pip install`	zero install; opens in a browser tab
Data locality	local	local, and never uploaded
Execution	interpreted Python	C++ compiled to WebAssembly, on the client
Speed (n=2048)	$1\times$ (baseline)	100 to $2500\times$ faster
vs. native C	–	$6.2\times$ over Biopython (global); within $1.8\times$ of SSW (local)
Correctness	–	bit-exact vs. the library and independent refs ($204/204$ per algorithm)
Generation metrics	embedding-based only	BLEU, chrF, ROUGE, CER/WER with paired-bootstrap significance
Homology search	named as future work	client-side BLAST, $\leq 0.43\%$ off NCBI BLAST+
Output	static plots	linked interactive figures; export to PDF/CSV/JSON/FASTA/...

Table 2: How our Studio extends its own predecessor. The library gathered the string-to-string algorithms behind one Python interface; `string2string Studio` keeps that coverage and adds an interactive, verified, client-side engine that is two to three orders of magnitude faster, together with the generation metrics and homology search the library did not provide, for programmers and non-programmers alike. Speed figures are medians over 30 random pairs at length 2048 on an Apple M1 Pro; correctness and BLAST concordance are defined in § 7.

Class	**Methods (selected)**	**Levels**	**Interactive view**
Alignment	Needleman–Wunsch, Smith–Waterman, semi-global, affine (Gotoh), banded, Hirschberg, DTW	char, word, token, line	DP matrix + path, dot plot, ribbon, edit script
Edit distance	Levenshtein, Damerau, Hamming, Jaro–Winkler, LCS	char, word, token	identity track, redline diff
Similarity	Jaccard, cosine, LCS ratio	char, word, token	score readout, method comparison
Search	naive, KMP, Boyer–Moore, Rabin–Karp, k-mismatch, IUPAC, both-strand	char, word, token	match map, both-strand track
Generation metrics	BLEU, chrF, chrF++, ROUGE-1/2/L, CER, WER, token-F_1	word, token, char	corpus + per-segment, bootstrap
Homology	`blastn`, `blastp`, Karlin–Altschul, seed-and-extend, MSA + tree	residue	hit map, conservation skyline, tree

Table 3: The string-to-string family as `string2string Studio` implements it: for each operation class, representative methods, the levels at which they run, and the interactive view the platform provides. Every method listed is in the released engine and is checked in continuous integration (Section 7).