Received: 19 April 2022 | Revised: 18 March 2024 | Accepted: 15 April 2024

DOI: 10.1111/1755-0998.13964

MOLECULAR ECOLOGY RESOURCES WILEY

RESOURCE ARTICLE

SEGUL: Ultrafast, memory-efficient and mobile-friendly software for manipulating and summarizing phylogenomic datasets

Heru Handika | **Jacob A. Esselstyn**

Museum of Natural Science and Department of Biological Sciences, Louisiana State University, Baton Rouge, Louisiana, USA

Correspondence
Heru Handika, Museum of Natural Science and Department of Biological Sciences, Louisiana State University, Baton Rouge, LA 70803, USA.
Email: hhandi1@lsu.edu

Handling Editor: Alana Alexander

Abstract

Phylogenetic studies now routinely require manipulating and summarizing thousands of data files. For most of these tasks, currently available software requires considerable computing resources and substantial knowledge of command-line applications. We develop an ultrafast and memory-efficient software, SEGUL, that performs common phylogenomic dataset manipulations and calculates statistics summarizing essential data features. Our software is available as standalone command-line interface (CLI) and graphical user interface (GUI) applications, and as a library for Rust, R and Python, with possible support of other languages. The CLI and library versions run native on Windows, Linux and macOS, including Apple ARM Macs. The GUI version extends support to include mobile iOS, iPadOS and Android operating systems. SEGUL leverages the high performance of the Rust programming language to offer fast execution times and low memory footprints regardless of dataset size and platform choice. The inclusion of a GUI minimizes bioinformatics barriers to phylogenomics while SEGUL's efficiency reduces economic barriers by allowing analysis on inexpensive hardware. Our support for mobile operating systems further enables teaching phylogenomics where access to computing power is limited.

KEYWORDS
alignment manipulation, alignment summary statistics, concatenation, phylogenomics, segul

1 | INTRODUCTION

The declining cost of DNA sequencing has increased the adoption of phylogenetic studies using thousands of loci, but this trend has also brought substantial analytical challenges. The most demanding parts of typical phylogenomic workflows, such as raw sequence read cleaning and adapter trimming (e.g. Fastp (Chen et al., 2018)), contig assembly (e.g. SPAdes (Bankevich et al., 2012)), sequence alignment (e.g. MAFFT (Katoh et al., 2002)), phylogenetic tree estimation (e.g. RAxML-NG (Kozlov et al., 2019) and IQ-TREE (Minh et al., 2020; Nguyen et al., 2015)), and species delimitation (e.g. BPP (Yang, 2015)), use high-performance programming languages (e.g. C and C++). However, other essential processes, such as alignment manipulation (e.g. filtering, splitting, extracting and concatenating) and summary statistic calculation (e.g. number of parsimony informative sites, percent missing data, etc), are typically carried out in interpreted languages, such as Python (e.g. Borowiec, 2016; Faircloth, 2016), R (e.g. Hutter et al., 2022), or Perl (e.g. Kück & Longo, 2014). The computational efficiency of this approach is limited by the requirement of an interpreter running alongside the application, type inference at runtime and garbage collection memory management, which together result in a high memory footprint. One exception to this pattern is the program goalign (Lemoine & Gascuel, 2021), which uses a compiled programming language and eliminates dependencies required

at runtime. Many of the program functions, however, operate on only a single file, forcing users to write custom scripts to work on thousands of genomic files.

With some exceptions (e.g. BPP (Yang, 2015) and FastQC (Andrews, 2010)), most commonly used phylogenomic programs are available only as command-line interface (CLI) applications. CLI programs use computer resources more efficiently than graphical user interface (GUI) applications, and they are also easier to develop than comparable GUI software, but they present barriers for scientists with limited computing knowledge or support. An approach using a high-performance programming language with a GUI would minimize the computing skills needed to study phylogenomics.

A fast, memory-efficient, reduced-dependency application for phylogenomic studies would enhance research efficiency and repeatability, while also improving accessibility for biologists with limited computing resources. Furthermore, efficient computing reduces the carbon footprint of bioinformatics (Grealey et al., 2022). Such applications often require programmers to use a fast, compiled programming language that allows fine control over how data are managed in computer memory. In this context, the two commonly used programming languages are C and C++. They require programmers to ensure valid memory access, correct variable type to store data, and ensure no data races (i.e. multiple cores/threads modify data concurrently), which make them challenging to use (Perkel, 2020). These code-correctness issues are difficult to avoid and represent common problems in phylogenetic software (Darriba et al., 2018). The recently emergent programming language, Rust, offers a memory-safe alternative to C/C++ (Köster, 2016; Perkel, 2020). It comes with efficient development tools (e.g. a package manager and a simple build system), guarantees valid memory access, does not require garbage collection, and prevents data races for multithreading applications. As a compiled programming language, Rust has zero dependency at runtime and can be distributed as a single executable CLI. Developing phylogenomic tools in Rust promises fast and efficient performance. Reducing dependencies minimizes conflict with other applications when used as part of analysis pipelines and leads to improved research reproducibility.

GUI development is more complicated than CLI development, especially when targeting multiple platforms. A common cross-platform approach uses Java (e.g. BEAST (Suchard et al., 2018), BEAST2 (Bouckaert et al., 2019), FastQC (Andrews, 2010)), but this strategy is often limited by the language's memory management. Furthermore, it is challenging to maintain a consistent user interface (UI) across operating systems (see TaxonDNA documentation, https://github.com/gaurav/taxondna (commit hash: 50584ac)). An alternative approach uses the Shiny package in R (e.g. phruta (Román-Palacios, 2023), treehouse (Steenwyk & Rokas, 2019)), but is less efficient because the application runs in the R environment and a browser. An emergent cross-platform framework, Flutter, promises mobile and desktop support with consistent UI across platforms. The programming language Dart, required to write Flutter applications, uses garbage-collected memory management, and features an excellent foreign-function interface to interact with higher-performance programming languages. Combining the Flutter framework and Rust language allows us to develop a cross-platform, high-performance GUI application for phylogenomics.

We developed the SEGUL (Sequence and Genomic Utilities) applications for phylogenomic data manipulation and summarization. They are available as a CLI, GUI and programming language library, with support for MacOS, Linux, Windows, iOS, iPadOS and Android. The application is supplemented by adaptive web-based documentation for easy navigation of SEGUL's various features across a wide range of devices. We designed SEGUL with beginners in mind, while still providing advanced features for experienced users. As such, SEGUL is suitable for both research and teaching. SEGUL's use of the Rust programming language combined with the Flutter-coded GUI provides consistent performance across supported operating systems.

2 | IMPLEMENTATION

SEGUL includes a compiled, single executable, command-line application with no dependencies. This version supports Windows (native and Windows Subsystem for Linux), MacOS, and Linux. We provide a fully static executable for old Linux distributions (i.e. the distributions with GLIBC version lower than the Rust minimum requirement, see https://doc.rust-lang.org/nightly/rustc/platform-support.html) and a dynamically linked executable to GLIBC (https://www.gnu.org/software/libc/) for more optimized performance in newer Linux distributions. Users can install the pre-compiled executable provided in the source code repository or compile the application from the source code (see the Software Availability section below). The latter installation method expands SEGUL platform support to any operating system supported by Rust (https://doc.rust-lang.org/nightly/rustc/platform-support.html). The compiler also fine-tunes the resulting executable for the user's computer.

The GUI version is written in Dart using the Flutter framework. All demanding computations use the same Rust code base as our CLI and library. The application is available in beta supporting Windows, MacOS, Linux, iOS, iPadOS and Android. The current version is available to test using Apple TestFlight for iOS, iPadOS and macOS, and at the GitHub repository page for other operating systems (see Software Availability below). We expect to distribute the stable version via the official store of each operating system, such as AppStore for iOS and PlayStore for Android.

SEGUL is also available as a Rust library (called crates in the Rust programming terminology), allowing developers to access SEGUL functions through the application programming interface (API). We assigned the API version number by following semantic versioning 2.0.0 (https://semver.org/). For Rust, the installation process is as easy as typing 'cargo add segul' or by manually adding SEGUL as a dependency in the Rust cargo.toml file. The versality of the Rust programming language and the performance of our library will be attractive for usage in slower programming languages (e.g. Python and R). For these languages, programmers can bind SEGUL via the C programming interface. In R and Python, however, Rust support

is growing due its value for developing high-performance software without the challenging aspects of C/C++ memory management. Packages exist to simplify the binding of Rust. In R, programmers can use the rextendr package; in Python, PyO3 is available. Python or R code can also interact with the SEGUL CLI using the operating system process. While SEGUL CLI is a single executable with no dependency, for published software, interacting through the SEGUL API yields a cleaner application design and avoids common dependency issues, such as improper setup of environment variables. We used a similar approach to develop the GUI version. We provide more detailed instructions in our application documentation.

Cross-platform quality documentation is essential in modern application development. We leverage the open-source static-site generator, Docusaurus (https://docusaurus.io/), to rapidly build a cross-platform website interface for the application documentation. The content of the documentation is written in Markdown, while the rest of the website is written in TypeScript. Our documentation source code is available inside the 'website' folder in the same repository as our GUI code (https://github.com/hhandika/segui). We use the front-end cloud platform, Vercel (https://vercel.com/), to host and automatically deploy the website after modifications in the main branch of the source code on GitHub.

3 | FEATURE AND USAGES

SEGUL development focuses on improved efficiency when working with thousands of alignment files, enabling analysis on computers with limited capabilities. We achieve this goal by reducing execution time and RAM usages. SEGUL has a growing list of features for phylogenomic data manipulation and summary statistic calculation (Table 1). Although phylogenomic analyses are growing in frequency, Sanger sequencing remains common in certain circumstances. Several features of our application, such as alignment concatenation, sequence removal and translation, are applicable to both phylogenomic and Sanger datasets.

All versions of SEGUL work on FASTA, NEXUS, and relaxed PHYLIP for alignment and processed sequence files, NEXUS and RaXML standards for partitions, FASTA files for contiguous (contig) sequences, and uncompressed and GNU Gzip compressed FASTQ files for raw-read sequences. All sequence and alignment files are supported in interleaved and sequential format. Except for alignment splitting, all the features support multiple input files. All application versions provide detailed log-file output. SEGUL-critical and some non-critical functions are tested using the unit and integration test system provided by the Rust and Dart programming languages. We establish a continuous integration (CI) system using GitHub Actions (https://github.com/features/actions). This system is designed to automatically validate any modifications made to the code. We ensure that failures in the designed tests are publicly displayed in the source code repository. Furthermore, our GitHub Actions CI system facilitates the release of CLI applications to all supported operating systems. Additionally, it enables the distribution of GUI applications for operating systems that permit installations from external sources, including Windows, Linux, and Android.

TABLE 1 Feature comparison among SEGUL v0.20.0, AMAS v1.02, and goalign v0.3.5.

Features	SEGUL	AMAS	Goalign
Alignment compression			Yes
Alignment concatenation	Yes	Yes	Limited
Alignment consensus			Yes
Alignment conversion	Multiple files	Multiple files	Single file
Alignment drawing			Yes
Alignment filtering	Yes		
Alignment masking			Yes
Alignment shuffling			Yes
Alignment splitting	Yes	Yes	Yes
Alignment summary statistics	Rich	Moderate	Limited
Alignment to unaligned sequence conversion	Planned		Yes
Alignment transposal			Yes
Genomic summary statistics	FASTQ reads, contigs		
Duplicate sequence removal	Planned		Yes
Multi-sequence alignment			Yes
Open reading frame finder			Yes
Partition format conversion	Yes		
Sample distribution mapping	Yes		
Sequence ID extraction	Yes		
Sequence comparison			Yes
Sequence extraction	Multiple files		Single file
Sequence ID renaming	Multiple files		Single file
Sequence removal	Multiple files	Multiple files	Single file
Sequence replication		Multiple files	Single file
Sequence translation	Multiple files	Multiple files	Single file
Sequence trimming	Planned		Yes
Sequence unique ID parsing	Yes		
API support	Yes	Yes	Yes
GUI version availability	Yes		
Mobile support	Yes		

The command-line version features an informative terminal output with information on the application input, processing stages, and output. We provide default commands that eliminate the need for users to type them for common scenarios. For example, when the input is DNA sequences, users do not need to type the '–datatype' command. Since version 0.19.0, SEGUL automatically detects the input format based on the file extension, eliminating the need for using '–input-format'. Multiple file outputs will always be written to a directory. The applications do not automatically overwrite existing files but do provide an overwrite option for automated phylogenomic pipelines.

The GUI version provides interactive access to SEGUL features, while offering similar performance to its CLI sibling. We leverage Flutter cross-platform and adaptive UI support to adapt UI elements

to different screen sizes and platforms. We followed the open-source Material 3 design system (https://m3.material.io/) that provides guidelines for creating adaptive, accessible, cross-platform applications. For example, on a small screen, such as a smartphone, features are accessed through a bottom navigation bar, whereas on a medium screen (e.g. tablets and foldable smartphones) and a large screen (e.g. laptops and desktops) we use a navigation rail and navigation drawer, respectively. The narrower navigation component (navigation rail) and the wider version (navigation drawer) are placed vertically on the left side of the screen to exploit greater screen width. We also implement Material 3 colour schemes to adapt the application colour palette in all supported platforms. This integration provided by Flutter leverages the accessibility features of Material 3 to enhance user experience and ensure optimal rendering for people with visual impairment. We also provide support for dark mode to reduce eye strain when running our GUI application in a low-light environment. For platforms that support dynamic colour rendering, our application colours will adapt to the user's operating system settings. For instance, on macOS and Windows, the application colour palette will be based on the operating system accent colour setting. On Android and Linux, the colour palette will be based on the users' operating system wallpaper. The dynamic colour is unsupported on iOS and iPadOS. On macOS, our application uses the Apple App Sandbox feature (https://developer.apple.com/documentation/security/app_sandbox) to enhance application security and protect user data. The App Sandbox will eventually also allow us to distribute the application in the Mac App Store, which simplifies installation and software update. A similar approach is implemented for iOS, iPadOS, and Android.

The SEGUL GUI supports all the features of the CLI version, while outputting similar log files to enable reproducibility. We also provide interactive windows to show the file inputs and output. The input window provides the list of the files, their sizes, the last modified time, and a delete button to remove the file. For plain-text and comma-separated value (CSV) output files, such as the output of summary statistics and sequence ID extraction function, we provide a built-in viewer to allow users to inspect the file output without having to leave the application. We also include a share function to simplify cross-device file sharing using operating system features. For example, on iOS and MacOS, users can send files between Apple devices via AirDrop without leaving SEGUL GUI. The file sharing function also allows users to transfer output files to other applications installed in the user operating system. Due to the interactive nature of GUI application, the default folder feature is unavailable for the GUI version. On desktop operating systems, the application uses the operating system directory selection UI that allows users to create a new directory. On mobile operating systems, SEGUL GUI only writes to its designated document directory mandated by the operating system. We provide a data usage menu in the application settings to allow users to manage the application data. We anticipate future developments to provide interactive statistics.

Current documentation is available in English. The site navigation components and its layout are adaptive to viewing in different screen sizes. Like our GUI application, our documentation also supports dark mode. We anticipate translation of the documentation to other languages, and we welcome public contributions toward this goal.

4 | FEATURE COMPARISON

We compare SEGUL to goalign (Lemoine & Gascuel, 2021) and AMAS (Borowiec, 2016) because they share similar performance and use cases. goalign is the most feature-rich (Table 1), however, most of its functions operate on a single file; two exceptions are alignment concatenation (multiple files) and sequence alignment (two files). AMAS and SEGUL, on the other hand, operate on single or multiple files. SEGUL CLI provides default commands, default output directory, and safety features to avoid overwriting existing files. The GUI version keeps the safety feature of the CLI application and provides an interactive menu to simplify user experience. AMAS will only provide notice after overwriting existing files and will only output to the current working directory. goalign does not provide any notice after overwriting existing files. The AMAS alignment splitting feature will always write the output files in the same folder as the input files. All applications provide APIs, but neither AMAS nor goalign offers a GUI. Several features (e.g. sample mapping, sequence unique ID parsing, and partition format conversion) are unique to SEGUL.

While several SEGUL features overlap with AMAS and goalign, SEGUL provides greater functionality. For instance, SEGUL generates summary statistics for alignment files, raw read sequences and contiguous sequence, whereas AMAS and goalign only support alignment files. The application's raw summary statistics provide a simple version of the statistics generated by FastQC (Andrews, 2010). SEGUL outputs CSV files and is designed to compare many raw read sequences quickly without an additional application (e.g. multiQC to summarize FastQC results). For alignment files, SEGUL always checks that the sequences within each alignment are the same length and by default checks that sequences contain only valid *IUPAC* characters. To speed up processing for most features, users can skip the *IUPAC* check using the '–datatype ignore' option, but the sequence length is always checked. AMAS does not check for *IUPAC* validity and the sequence length verification is optional. goalign checks both but does not generate alignment partitions. Another example, the SEGUL sequence removal feature supports regular expression, file and terminal input, while AMAS supports only terminal input. As noted above, the goalign sequence removal works on only a single file.

5 | PERFORMANCE COMPARISON

5.1 | Testing methodology

To highlight the performance of SEGUL, we compared features of the command line version SEGUL v0.20.0 to AMAS v1.02 (see Borowiec, 2016), the fastest alternative application and the most comparable with SEGUL (e.g. concatenation with partition output and multiple file input support across most features). Where

necessary, we test the applications under different settings to provide fair performance comparisons. For example, we test AMAS using the '–check-align' command, which SEGUL does by default. We include goalign v0.3.5 for the alignment concatenation performance test. We used six published genomic datasets (four DNA sequence datasets and two amino acid sequence datasets (AA)) with a range of taxon, site and character counts (Table 2). We downloaded the datasets either directly from the original sources or using BenchmarkAlignments scripts (https://github.com/roblanf/BenchmarkAlignments).

We tested each platform using all the alignment datasets, with each combination replicated five times. For sequence removal tests, we used the SEGUL ID extraction feature to get the list of sequence IDs in each dataset. We removed the first three taxa in alphabetical order. For the alignment splitting test, we concatenated all the alignments using SEGUL CLI and used the resulting files and their partitions to conduct the splitting test. All the datasets were held in the internal storage of the computer. We conducted the test using FASTA, NEXUS and PHYLIP input files supported by all tested applications.

We ran the test on a custom-built PC running Linux. We automated the testing process using SHELL scripts optimized for FISH SHELL (https://fishshell.com/). To compare the performance of the applications, we use GNU Time (https://www.gnu.org/software/time/). We recorded execution time, memory, and CPU core usages. SEGUL detects available CPU cores and uses them according to the current workload. AMAS and goalign, on the other hand, have settings for the core counts. We set AMAS and goalign to use all available cores in comparisons with SEGUL. Like SEGUL, the actual number of cores being used by AMAS and goalign would depend on workload, and to some extent, on the multithreading algorithm implemented by the programming language.

To compare the performance and to test the scalability of the GUI version relative to the CLI version of SEGUL, we conducted limited tests for the SEGUL GUI v1.0.0-beta6 on a Linux PC, an iPad, and a budget Android smartphone (Table 3). We used Oliveros et al. (2019) and Shen et al. (2018) datasets for GUI performance tests. These two datasets have the highest number of characters for DNA and amino acid sequences, respectively, among our six test datasets. For the tests on the desktop Linux, we also include raw read summary statistics of a whole genome sequence (WGS) of *Peromyscus eremicus* (NCBI SRA #SRR26062012, Table 2) provided by Baylor College of Medicine on NCBI Sequence Read Archive (SRA). We downloaded the genome file using SRA toolkit. We used fastq-dump application in SRA toolkit with '–split-files' command to extract the downloaded SRA file into two uncompressed FASTQ reads. The resulting files were each 338 gigabytes (GB) in size. Due to limited storage space in the testing computer, we stored the files in an external solid-state drive (SSD) (Table 3).

We used GNU Time to measure the RAM usage on Linux and collected the execution time provided by the application in the log file that measured only the time of executing the assigned task. The log file is available to access in the setting menu of the application. Our test with the CLI version showed the GNU Time measurements were identical to the internal measurement provided in the SEGUL log file. On smartphones and tablets, tracking accurate and comparable hardware usages with desktop applications is complex. Thus, we measured only the execution time provided directly by the application. We replicated each combination of tested feature and dataset five times. On the Android device, simultaneously inputting over two thousand files crashed the application. We split the datasets into multiple folders with a maximum of 1500 files each. The strategy allowed us to use 'select all' features in the input file screen and took advantage of the SEGUL 'add more files' feature to input all the files in the dataset. The data were kept in the internal storage of the device. The same strategy did not work for the iPadOS testing device when the data were in internal storage. It lost access to the data after reaching over ~3000 files, which caused issues for inputting

TABLE 2 Dataset sources, data type, taxon count, locus count, character count (missing characters + nucleotides), site count, file size and data source.

Datasets	Datatype	Taxon count	Locus count	Character count	Site count	File size	Dataset URL
Chan et al. (2020)	DNA	50	13,181	239,310,808	6,180,393	247 MB	https://doi.org/10.5061/dryad.8cz8w9gn7
Esselstyn et al. (2021)	DNA	102	4040	358,099,656	5,398,947	356 MB	https://doi.org/10.5281/zenodo.6459213
Jarvis et al. (2014)	DNA	49	3679	453,333,006	9,251,694	438 MB	http://gigadb.org/dataset/101041#
NCBI SRA #SRR26062012	DNA	1	WGS	243,874,896,842*	-	2 × 338 GB	https://www.ncbi.nlm.nih.gov/sra/SRR26062012
Oliveros et al. (2019)	DNA	221	4060	522,529,858	2,464,926	523 MB	https://doi.org/10.5061/dryad.2vd01gr
Shen et al. (2018)	Amino acid	343	2408	398,842,115	1,162,805	421 MB	https://doi.org/10.6084/m9.figshare.5854692
Wu et al. (2018)	Amino acid	90	5162	257,060,172	3,050,198	250 MB	https://doi.org/10.6084/m9.figshare.6031190.v2

Note: The asterisk (*) denotes a total base count for two reads. The file size is estimated based on NEXUS input alignments, except for NCBI SRA #SRR26062012, which are two uncompressed FASTQ reads of a whole genome sequence (WGS).

TABLE 3 Device specifications for each testing platform.

	Linux	Android	iPadOS
Model	Custom-built PC	Xiaomi Redmi Note 12	iPad Mini (6th generation)
Processor	AMD Ryzen 5900×	Qualcomm Snapdragon 685	Apple A15 Bionic
Core/threads	12/24	8/8	2P + 4E cores
Ram size	64 GB	8 GB	4 GB
Storage	Western Digital Black SN770 1 TB	128 GB	64 GB
GPU	EVGA GeForce RTX 2060 Super 8 GB XC Ultra	Adreno 610	Apple GPU (5-core)
External storage	Crucial X10 Pro 2 TB	–	Crucial X10 Pro 2 TB
Operating System	openSUSE Tumbleweed ×86_64	Xiaomi HyperOS 1.0.1.0 with Android 14	iPadOS 17.2
Kernel version	6.6.11-1-default	5.15.94	–

the Oliveros et al. (2019) dataset (4060 files). However, we were able to input the files simultaneously after we moved them to an external SSD. For consistency, we stored all the data for the iPadOS tests in the external SSD (Table 3).

To see how SEGUL CLI performance compared to the other applications, we calculated means and standard deviations of CPU time and RAM usages across replicate runs for each dataset and input formats for each tested feature (Figure 1). We calculated the CPU time by incorporating both the execution time and CPU thread usage. A single thread usage corresponds to 100 percent CPU usage. Therefore, the CPU time was computed using the following formula:

$$\text{CPU time (CPU seconds)} = \text{Execution time (seconds)} \times \frac{\text{CPU usage (\%)}}{100}$$

Because CPU usages were not available for GUI application, we used execution time to compare SEGUL CLI and GUI performance. We used R v4.2.3 with dplyr v1.1.2 and ggplot2 v3.4.2 packages to generate summary statistics and visualize the results, respectively. All scripts and raw data are available at (https://github.com/hhandika/segul-bench).

5.2 | Testing results

On average, SEGUL CLI used less CPU time than AMAS across all tested datasets, features, and settings (Figure 1). The starkest difference is for summary statistic calculations. SEGUL used 0.08 of CPU time (15.3 vs. 190.69 CPU seconds) that AMAS used, despite producing more statistics (Figure 1, Table S1). AMAS noticeably used more CPU time when using the '–check-align' setting. For example, on average across all datasets and input formats, AMAS '–check-align' used 40.5 times (135.05 vs. 3.33 CPU seconds) more CPU time than SEGUL for alignment concatenation. Even without the '–check-align' setting, AMAS used three times (10.03 vs. 3.33 CPU seconds) more CPU time (Figure 1, Table S1). When using '–datatype ignore', SEGUL CLI used less CPU time than AMAS across all tested datasets and input formats (Figure 1). It was 0.2 of CPU time (1.92 vs. 10.03 CPU seconds) for the alignment concatenation compared to the CPU time that AMAS used at default settings and 0.01 of CPU time (1.92 vs. 135.05 CPU seconds) compared to AMAS using the '–check-align' setting (Figure 1, Table S1). SEGUL CLI used 0.6 (3.33 vs. 6.05 CPU seconds) and 0.3 (1.92 vs. 6.05 CPU seconds) of CPU time that goalign used for alignment concatenation at the default and with '–datatype ignore' settings, respectively (Figure 1, Table S1).

Across all tests, SEGUL CLI used less RAM than AMAS. The starkest contrast is when converting alignments to different formats and removing sequences (Figure 1). On average across tested datasets and input formats, SEGUL used 0.05 of the RAM space that AMAS used for the two analyses (24.47 vs. 485.96 MB for alignment conversion and 24.65 vs. 502.14 MB for sequence removal) (Figure 1, Table S1). However, for the alignment splitting tests when the input files are in NEXUS format, the RAM space that SEGUL used was comparable to AMAS (Figure 1). Compared to goalign when concatenating alignment, SEGUL used 0.18 (489.87 MB), while AMAS used 0.64 (1700.87 MB) of the RAM space that goalign used (2661.39 MB) (Figure 1, Table S1). Using the '–datatype ignore' setting in SEGUL CLI did not have considerable effect on the RAM usage (Figure 1).

On average, SEGUL CLI was faster and more memory efficient than the GUI version (Figure 2). On identical hardware running on Linux, both versions had nearly equal execution times for three of the six tested features across the three tested datasets (Figure 2b). SEGUL CLI was 1.4 times (4.19 vs. 3.03 s, on average) faster for alignment concatenation and splitting using the Oliveros et al. (2019) dataset and 1.3 (2.51 vs. 3.56 s) and 1.2 times (4.07 vs. 4.9 s) faster, respectively, using the Shen et al. (2018) dataset (Figure 2b). The highest RAM usage difference was 20.8 times more on the GUI (187.85 MB) than the CLI version (9.04 MB) when summarizing WGS reads (Figure 2a). Compared to the mobile version, the desktop GUI version was faster than the mobile version. The starkest difference was when summarizing alignments. On average between both tested datasets, the desktop version was 5.3 (1.34 vs. 7.11 s) and 10.3 times (1.34 vs. 13.77 s) faster than on iPadOS and Android, respectively (Figure 2b).

6 | DISCUSSION

The field of phylogenomics relies on a broad range of efficient software for many operations in a typical workflow (e.g. raw read

assembly, tree inference), but some aspects of standard pipelines are neglected. We developed SEGUL to fill a need for software that can manipulate genomic files and calculate summary statistics in a manner suitable for computational experts and novices alike. SEGUL offers consistently fast operations, low memory demands, and efficient CPU use across all supported features. While the GUI version was slower and less memory efficient than the CLI version, it was still faster and less memory demanding than AMAS for most scenarios in default settings across most of tested datasets, except for the alignment splitting. Similarly, except for alignment splitting, RAM usage by either the CLI or GUI version was kept under 1 GB regardless of dataset size and tested feature. SEGUL's high performance and memory efficiency would enable processing large datasets with low-end computers. Computational efficiency will only increase in importance with routine adoption of whole genome sequencing for phylogenetics.

The choice of the Rust programming language enabled us to develop a high-performance, memory-efficient application, but algorithm implementation also plays a crucial role. For instance, summarizing two 338-GB files of WGS reads used significantly less RAM space compared to summarizing alignments with a total file size of <1 GB (Table S1). Our summarization algorithms for sequence reads are highly optimized for handling large file sizes. The files are parsed and processed by line, with each line immediately dropped from memory. We used stream statistics to further optimize RAM usages. For instance, we update the mean of read counts after each

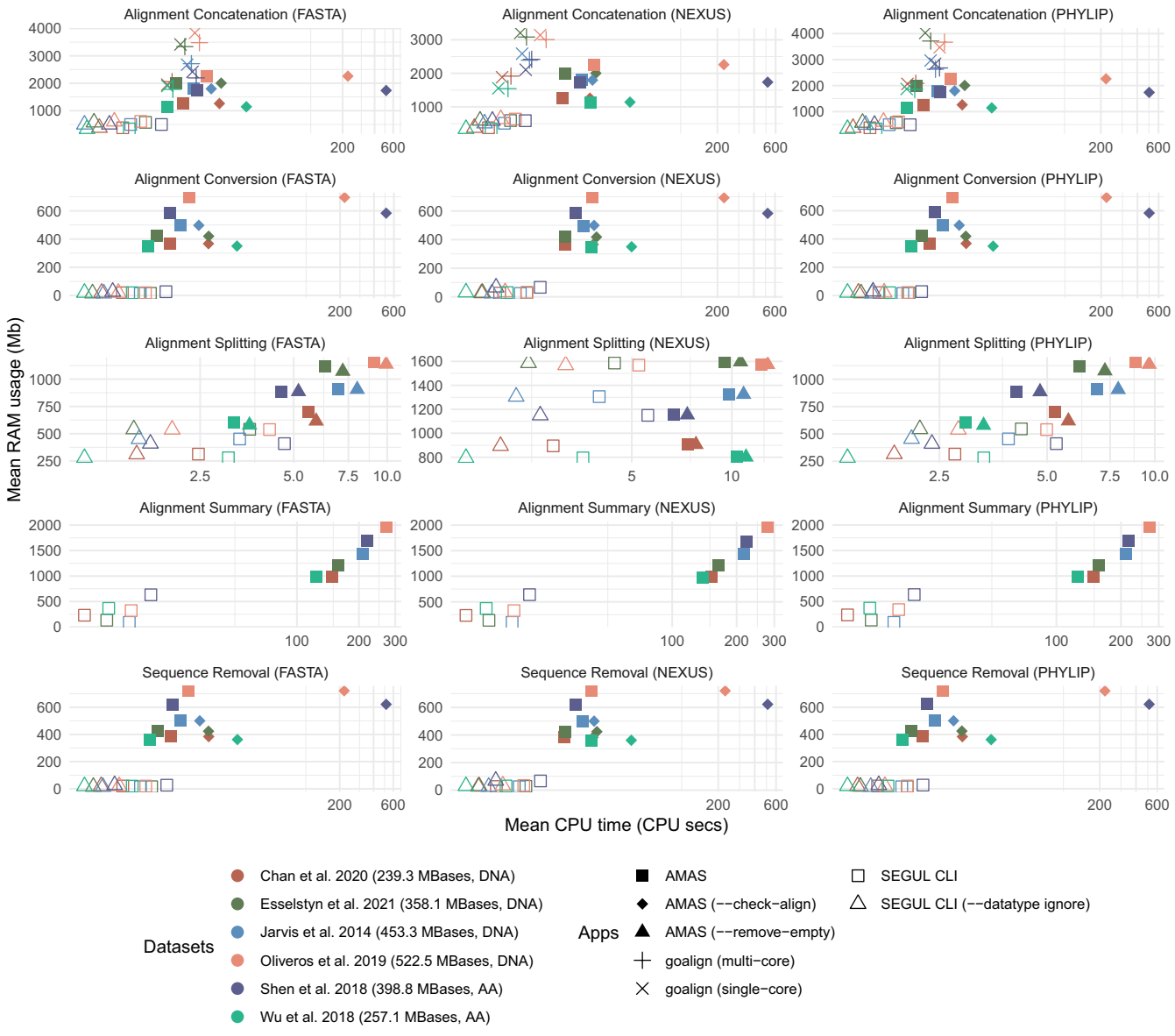

FIGURE 1 SEGUL (open symbols), AMAS (filled symbols), and goalign (line symbols) mean CPU time and RAM usage on all tested datasets across three supported input formats. We used logarithmic scale for the x-axis to simplify visualization. All analyses were completed on Linux. SEGUL (–datatype ignore) is not available for summary statistics. We only tested goalign for alignment concatenation. Mean execution time and CPU usages for all tested applications and datasets are available in Table S2.

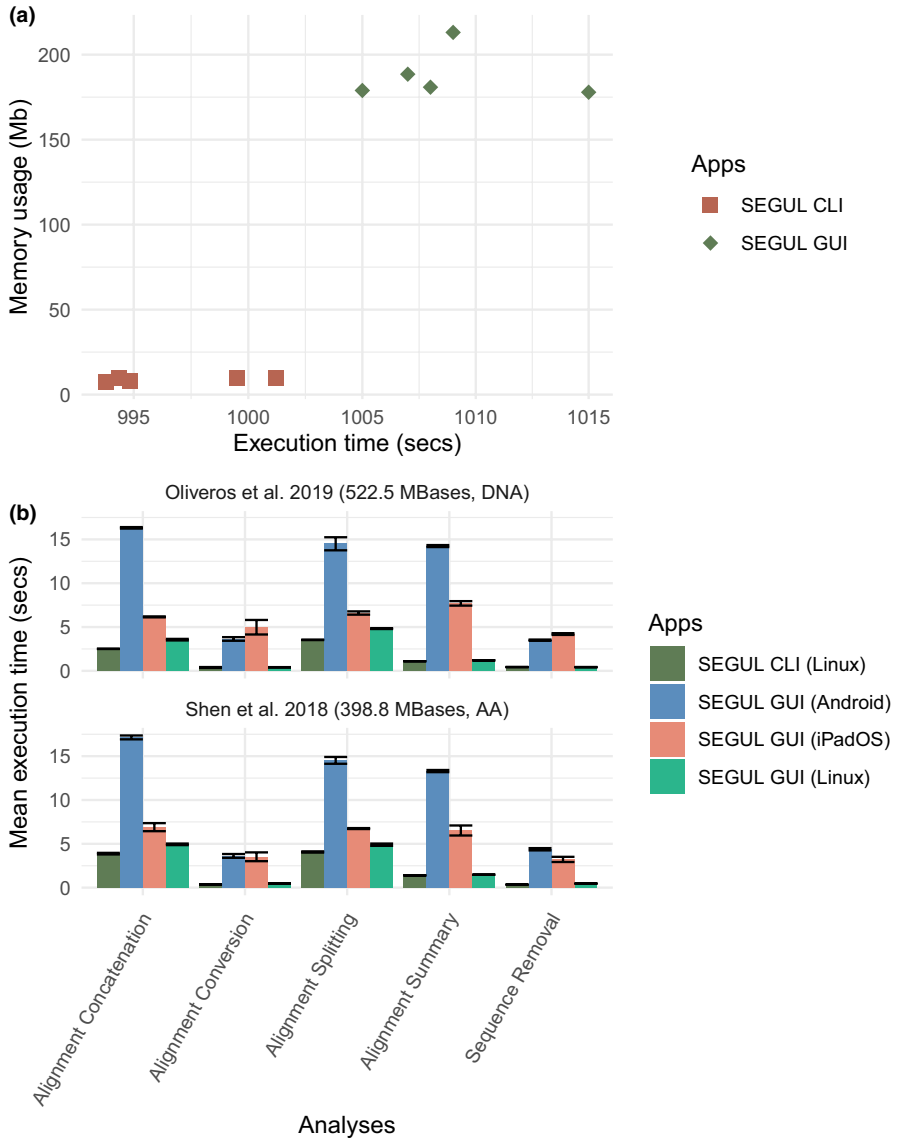

FIGURE 2 SEGUL CLI versus GUI comparison; (a) execution time and memory usage across five replicate runs on Linux for WGS read summary statistic (NCBI SRA # SRR26062012, Table 2); (b) mean execution time comparison across tested platforms and datasets. Error bars indicate ±1 standard deviation.

line of sequences to avoid storing many read sizes in RAM. For alignment concatenation, we use multiple cores only for getting unique IDs. The concatenation algorithm itself is single threaded. This allowed us to sort the loci alphanumerically, while reducing memory allocation. Concatenation with the '–datatype ignore' setting that eliminates expensive IUPAC character checks, was faster than the multithreaded AMAS, which does not check sequence lengths or IUPAC character validity. SEGUL always checks that each sequence in an alignment is the same length. We achieved memory efficiency for other features by carefully managing allocations. To our surprise, while using the compiled Go programming language, goalign used more RAM than AMAS with its interpreted Python programming language. The same as Python, Go uses garbage collected memory management. Compared to AMAS, goalign does more file checking, such as IUPAC character and alignment length checks. The checking may be inefficient, requiring significant memory allocation, and the data may be kept in the RAM longer than necessary. Another surprising result is that goalign was also slower when using the multicore setting (Figure 1). We suspect that the algorithm may be single threaded as shown with only 3.59 percent increase in CPU usage when using the multi-core setting (Table S1). The 3.59 percent increase may be used by the go multithreading algorithm (called goroutine) only for task scheduling between threads (Gao et al., 2023; Scionti & Mazumdar, 2017). Rust, on the other hand, uses a non-garbage-collected memory model (Perkel, 2020), which guarantees that when a variable is out of scope it will be dropped from RAM. We used the Rayon Rust library (https://docs.rs/rayon/latest/rayon/) to implement a high-performance multithreading algorithm that adjusts core usage based on data workload during execution. Our application lacked efficiency only when splitting alignments, especially when the input files were in a NEXUS format. NEXUS is more complex than PHYLIP and FASTA, which complicates file parsing (Maddison et al., 1997). In the case of alignment splitting, the parsed concatenated alignments were kept in memory as HashMap data structure for nearly the entire processing stage. It generated a new HashMap for each newly split alignment. We are actively working on optimizing the alignment splitting algorithm to improve memory efficiency.

There is an increasing trend of relaying to containerized applications (i.e. using Docker and Singularity) to solve dependency issues, but this fix makes it harder for computing novices to study phylogenomics. We take a different approach by minimizing dependencies to simplify installation and operation. For further convenience, we developed a GUI that retains efficiency and reproducibility. SEGUL support for mobile devices enables a new way to perform phylogenomic data manipulation and summary statistics. Both the tested mobile devices could execute large alignment datasets and were much faster than the other tested applications in some scenarios. For instance, SEGUL GUI execution time for removing sequences on the Android smartphone was 32.5 times faster than AMAS '–check-align', while AMAS was running on the much more capable desktop computer (Table S1). Our application facilitates teaching phylogenomics to students who use mobile devices (e.g. tablets) as their primary computers. Many authors publish their alignments (e.g. Jarvis et al., 2014; Oliveros et al., 2019, https://github.com/roblanf/BenchmarkAlignments) and phylogenetic tree estimation can be conducted on the web (DeSalle et al., 2020). Our application allows the entire process from alignment inspection and concatenation to phylogenetic tree estimation to be conducted from a mobile device.

Mobile devices are becoming more powerful, with some tablets, such as Apple's iPad Pro and iPad Air, using the same processors as their laptop and desktop counterparts. The only limitation is that mobile operating systems, including Android since SDK 30 (https://developer.android.com/about/versions/11/privacy/storage), forbid applications to access the file directly (using raw path) for security reason. File input in the SEGUL mobile version needs to be cached to a temporary location that the application can access. Depending on the operating system and the hardware capability, this process makes it slow to nearly impossible for inputting large numbers of files. For instance, we could input the Shen et al. (2018) data (2408 files), but not the Oliveros et al. (2019) (4060 files) on the iPad Mini when the file is stored internally. The Xiaomi smartphone allowed input of the Oliveros et al. (2019) data if the files were divided and loaded in two batches. We plan to provide a feature to allow input of a compressed file to overcome this limitation. SEGUL, aided by the Flate2 Rust library (https://docs.rs/flate2/latest/flate2/), comes with a cross-platform GNU Gzip parsing library for compressed sequence reads. Implementing similar support for compressed alignment files on SEGUL and other phylogenomic applications would not only benefit the mobile application but would also improve efficiency in analysing and storing phylogenomic datasets. Greater usage of compressed files would reduce energy costs and ecological footprints of running phylogenomic analyses (Grealey et al., 2022; Kothiyal et al., 2009).

We achieve a cross-platform, high-performance application by writing our GUI code in Flutter and handling expensive computation in Rust. Our implementation of this approach in the neglected aspects of typical phylogenomic workflows provides proof of concept in the development and scaling of high-performance applications with GUI interaction on desktop and mobile operating systems. Extension of our approach to other aspects of phylogenomic workflows would pave the way for more user-friendly software to study phylogenomics and enable teaching students with limited access to computational power.

7 | CONCLUSIONS

SEGUL is an ultrafast, memory-efficient tool to manipulate and generate summary statistics for phylogenomic datasets. It is consistently fast with low memory usages regardless of dataset, operating system, and CPU architecture, while providing extra features, such as a log file, a more informative terminal output, GUI application, and a broad array of summary statistics. SEGUL supports devices from smartphones, tablets, laptops and desktops, to high-performance computers. SEGUL's efficient use of computing resources, cross-platform support, and inclusion of a log file offers greater repeatability and accessibility than alternative applications.

AUTHOR CONTRIBUTIONS

H.H. conceived the project, designed the application, and wrote the code. J. A. E. wrote initial algorithms for some SEGUL features and provided feedback on the application design. Both authors wrote the manuscript and documentation.

ACKNOWLEDGEMENTS

We thank Andre E. Moncrieff, Austin S. Chipps, Carl R. Hutter, Darwin Morales-Martínez, Diego J. Elias, Giovani Hernández-Canchola, Glaucia C. Del-Rio, Roberta C. Canton, Samantha L. Rutledge, Sarin Tiatragul and Spenser J. Babb-Biernacki for their feedback on the application and its documentation. Spenser J. Babb-Biernacki improved the SEGUL application name. Financial support was provided by the National Science Foundation DEB 1754393 and the Alfred L. Gardner and Mark S. Hafner Mammalogy Fund. Several SEGUL features are inspired by Phyluce, AMAS, goalign and FrogCap. SEGUL benefited greatly from general-purpose libraries, particularly those provided by the Rust and Flutter programming communities. Alana Alexander and four anonymous reviewers provided helpful suggestions on an earlier version of this manuscript.

CONFLICT OF INTEREST STATEMENT

Both authors declare no conflicts of interest.

DATA AVAILABILITY STATEMENT

All datasets used for testing software performance are provided by the authors as described in Table 2. The code for performance tests is available at https://github.com/hhandika/segul-bench (https://doi.org/10.5281/zenodo.10655644).

SOFTWARE AVAILABILITY

SEGUL is open source and freely available under the Massachusetts Institute of Technology (MIT) licence. It is a cross-platform application and has been tested through automatic and manual testing on all supported platforms. Source code is available on GitHub at https://

github.com/hhandika/segul and https://github.com/hhandika/segui for the GUI version and the application documentation. We provide extensive documentation on installing and using all versions of the application. The documentation can be found at https://www.segul.app/. We recommend checking https://github.com/hhandika/segul for the most up to date documentation link. API documentation is available at https://docs.rs/segul/latest/segul/.

ORCID

Heru Handika https://orcid.org/0000-0002-2834-7175

REFERENCES

Andrews, S. (2010). FastQC a quality control tool for high throughput sequence data. http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. A., Dvorkin, M., Kulikov, A. S., Lesin, V. M., Nikolenko, S. I., Pham, S., Prjibelski, A. D., Pyshkin, A. V., Sirotkin, A. V., Vyahhi, N., Tesler, G., Alekseyev, M. A., & Pevzner, P. A. (2012). SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology: A Journal of Computational Molecular Cell Biology*, *19*(5), 455–477. https://doi.org/10.1089/cmb.2012.0021

Borowiec, M. L. (2016). AMAS: A fast tool for alignment manipulation and computing of summary statistics. *PeerJ*, *4*, e1660. https://doi.org/10.7717/peerj.1660

Bouckaert, R., Vaughan, T. G., Barido-Sottani, J., Duchêne, S., Fourment, M., Gavryushkina, A., Heled, J., Jones, G., Kühnert, D., De Maio, N., Matschiner, M., Mendes, F. K., Müller, N. F., Ogilvie, H. A., du Plessis, L., Popinga, A., Rambaut, A., Rasmussen, D., Siveroni, I., … Drummond, A. J. (2019). BEAST 2.5: An advanced software platform for Bayesian evolutionary analysis. *PLoS Computational Biology*, *15*(4), e1006650. https://doi.org/10.1371/journal.pcbi.1006650

Chan, K. O., Hutter, C. R., Wood, P. L., Jr., Grismer, L. L., & Brown, R. M. (2020). Target-capture phylogenomics provide insights on gene and species tree discordances in Old World treefrogs (Anura: Rhacophoridae). *Proceedings. Biological Sciences*, *287*, 20202102. https://doi.org/10.1098/rspb.2020.2102

Chen, S., Zhou, Y., Chen, Y., & Gu, J. (2018). Fastp: An ultra-fast all-in-one FASTQ preprocessor. *Bioinformatics*, *34*, i884–i890. https://doi.org/10.1093/bioinformatics/bty560

Darriba, D., Flouri, T., & Stamatakis, A. (2018). The state of software for evolutionary biology. *Molecular Biology and Evolution*, *35*(5), 1037–1046. https://doi.org/10.1093/molbev/msy014

DeSalle, R., Tessler, M., & Rosenfeld, J. (2020). Phylogenetic programs and websites. In *Phylogenomics* (pp. 213–222). CRC Press. https://doi.org/10.1201/9780429397547-20

Esselstyn, J. A., Achmadi, A. S., Handika, H., Swanson, M. T., Giarla, T. C., & Rowe, K. C. (2021). Fourteen new, endemic species of shrew (genus Crocidura) from Sulawesi reveal a spectacular Island radiation. *Bulletin of the American Museum of Natural History*, *454*(1), 1–108. https://doi.org/10.1206/0003-0090.454.1.1

Faircloth, B. C. (2016). PHYLUCE is a software package for the analysis of conserved genomic loci. *Bioinformatics*, *32*(5), 786–788. https://doi.org/10.1093/bioinformatics/btv646

Gao, C., Lv, H., & Tan, Y. (2023). Multithreading technology based on Golang implementation. *2023 3rd Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS)*, 603–608. https://doi.org/10.1109/ACCTCS58815.2023.00116

Grealey, J., Lannelongue, L., Saw, W.-Y., Marten, J., Méric, G., Ruiz-Carmona, S., & Inouye, M. (2022). The carbon footprint of bioinformatics. *Molecular Biology and Evolution*, *39*(3), 1–15. https://doi.org/10.1093/molbev/msac034

Hutter, C. R., Cobb, K. A., Portik, D. M., Travers, S. L., Wood, P. L., Jr., & Brown, R. M. (2022). FrogCap: A modular sequence capture probeset for phylogenomics and population genetics for all frogs, assessed across multiple phylogenetic scales. *Molecular Ecology Resources*, *22*(3), 1100–1119. https://doi.org/10.1111/1755-0998.13517

Jarvis, E. D., Mirarab, S., Aberer, A. J., Li, B., Houde, P., Li, C., Ho, S. Y. W., Faircloth, B. C., Nabholz, B., Howard, J. T., Suh, A., Weber, C. C., da Fonseca, R. R., Li, J., Zhang, F., Li, H., Zhou, L., Narula, N., Liu, L., … Zhang, G. (2014). Whole-genome analyses resolve early branches in the tree of life of modern birds. *Science*, *346*(6215), 1320–1331. https://doi.org/10.1126/science.1253451

Katoh, K., Misawa, K., Kuma, K.-I., & Miyata, T. (2002). MAFFT: A novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research*, *30*(14), 3059–3066. https://doi.org/10.1093/nar/gkf436

Köster, J. (2016). Rust-bio: A fast and safe bioinformatics library. *Bioinformatics*, *32*(3), 444–446. https://doi.org/10.1093/bioinformatics/btv573

Kothiyal, R., Tarasov, V., Sehgal, P., & Zadok, E. (2009). Energy and performance evaluation of lossless file data compression on server systems. *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, Article 4, 1–12. https://doi.org/10.1145/1534530.1534536

Kozlov, A. M., Darriba, D., Flouri, T., Morel, B., & Stamatakis, A. (2019). RAxML-NG: A fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference. *Bioinformatics*, *35*(21), 4453–4455. https://doi.org/10.1093/bioinformatics/btz305

Kück, P., & Longo, G. C. (2014). FASconCAT-G: Extensive functions for multiple sequence alignment preparations concerning phylogenetic studies. *Frontiers in Zoology*, *11*(1), 81. https://doi.org/10.1186/s12983-014-0081-x

Lemoine, F., & Gascuel, O. (2021). Gotree/Goalign: Toolkit and go API to facilitate the development of phylogenetic workflows. *NAR Genomics and Bioinformatics*, *3*(3), lqab075. https://doi.org/10.1093/nargab/lqab075

Maddison, D. R., Swofford, D. L., & Maddison, W. P. (1997). NEXUS: An extensible file format for systematic information. *Systematic Biology*, *46*(4), 590–621. https://doi.org/10.1093/sysbio/46.4.590

Minh, B. Q., Schmidt, H. A., Chernomor, O., Schrempf, D., Woodhams, M. D., von Haeseler, A., & Lanfear, R. (2020). IQ-TREE 2: New models and efficient methods for phylogenetic inference in the genomic era. *Molecular Biology and Evolution*, *37*(5), 1530–1534. https://doi.org/10.1093/molbev/msaa015

Nguyen, L.-T., Schmidt, H. A., von Haeseler, A., & Minh, B. Q. (2015). IQ-TREE: A fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies. *Molecular Biology and Evolution*, *32*(1), 268–274. https://doi.org/10.1093/molbev/msu300

Oliveros, C. H., Field, D. J., Ksepka, D. T., Barker, F. K., Aleixo, A., Andersen, M. J., Alström, P., Benz, B. W., Braun, E. L., Braun, M. J., Bravo, G. A., Brumfield, R. T., Chesser, R. T., Claramunt, S., Cracraft, J., Cuervo, A. M., Derryberry, E. P., Glenn, T. C., Harvey, M. G., … Faircloth, B. C. (2019). Earth history and the passerine superradiation. *Proceedings of the National Academy of Sciences of the United States of America*, *116*(16), 7916–7925. https://doi.org/10.1073/pnas.1813206116

Perkel, J. M. (2020). Why scientists are turning to rust. *Nature*, *588*(7836), 185–186. https://doi.org/10.1038/d41586-020-03382-2

Román-Palacios, C. (2023). The 'phruta' R package and 'salphycon' shiny app: increasing access, reproducibility, and transparency in phylogenetic analyses. *bioRxiv* (p. 2023.01.11.523621) https://doi.org/10.1101/2023.01.11.523621

Scionti, A., & Mazumdar, S. (2017). Let's go: A data-driven multi-threading support. *Proceedings of the Computing Frontiers Conference*, 287–290. https://doi.org/10.1145/3075564.3075596

Shen, X.-X., Opulente, D. A., Kominek, J., Zhou, X., Steenwyk, J. L., Buh, K. V., Haase, M. A. B., Wisecaver, J. H., Wang, M., Doering, D. T., Boudouris, J. T., Schneider, R. M., Langdon, Q. K., Ohkuma, M.,

Endoh, R., Takashima, M., Manabe, R.-I., Čadež, N., Libkind, D., … Rokas, A. (2018). Tempo and mode of genome evolution in the budding yeast subphylum. *Cell*, *175*(6), 1533–1545.e20. https://doi.org/10.1016/j.cell.2018.10.023

Steenwyk, J. L., & Rokas, A. (2019). Treehouse: A user-friendly application to obtain subtrees from large phylogenies. *BMC Research Notes*, *12*(1), 541. https://doi.org/10.1186/s13104-019-4577-5

Suchard, M. A., Lemey, P., Baele, G., Ayres, D. L., Drummond, A. J., & Rambaut, A. (2018). Bayesian phylogenetic and phylodynamic data integration using BEAST 1.10. *Virus Evolution*, *4*(1), vey016. https://doi.org/10.1093/ve/vey016

Wu, S., Edwards, S., & Liu, L. (2018). Genome-scale DNA sequence data and the evolutionary history of placental mammals. *Data in Brief*, *18*, 1972–1975. https://doi.org/10.1016/j.dib.2018.04.094

Yang, Z. (2015). The BPP program for species tree estimation and species delimitation. *Current Zoology*, *61*(5), 854–865. https://doi.org/10.1093/czoolo/61.5.854

SUPPORTING INFORMATION

Additional supporting information can be found online in the Supporting Information section at the end of this article.

How to cite this article: Handika, H., & Esselstyn, J. A. (2024). SEGUL: Ultrafast, memory-efficient and mobile-friendly software for manipulating and summarizing phylogenomic datasets. *Molecular Ecology Resources*, *00*, e13964. https://doi.org/10.1111/1755-0998.13964