COMPASS: COmparative MSA Performance Analysis & Scoring System

CAP5510 Bioinformatics Final Project

Aelly Alwardi, Daen Khan Patan, Ibraheem Qureshi

2025-11-29

Abstract

Multiple sequence alignment (MSA) is an important tool in molecular and evolutionary biology, and there are several programs and algorithms available for this purpose [1]. This project benchmarks widely used MSA programs like MAFFT, T-Coffee, ProbCons, Clustal Omega, and MUSCLE on curated protein datasets from the BAliBASE reference alignment database [1]. Following Pais et al. (2014) [1], we will (thereby writing scripts) run the mentioned programs to evaluate alignment accuracy using sum-of-pairs (SP) and total-column (TC) scores and measure computational cost using metrics like runtime and peak memory. We will not implement any new algorithms or programs. Instead, we plan to create an observational survey (software suite) that displays metrics like accuracy and efficiency of the MSA programs.

Introduction

Multiple sequence alignment (MSA) is a fundamental operation in computational biology that arranges biological sequences to identify regions of similarity. These alignments reveal evolutionary relationships, help identify conserved functional domains, and serve as input for phylogenetic analysis and structure prediction. Since the 1980s, researchers have developed numerous MSA algorithms based on different computational strategies, including progressive alignment, iterative refinement, and consistency-based methods. Choosing the right MSA tool requires understanding trade-offs between alignment accuracy and computational cost. These differences matter because alignment quality directly affects downstream analyses: an incorrect alignment can lead to wrong conclusions about protein function or evolutionary history. Benchmark studies have shown that no single tool performs best across all scenarios, with different programs excelling at different sequence family types. This project implements an automated benchmarking pipeline for five widely used MSA tools: MAFFT, MUSCLE, Clustal Omega, T-Coffee, and ProbCons. We evaluate these programs on the BAliBASE reference alignment database, measuring accuracy using sum-of-pairs (SP) and total-column (TC) scores while tracking runtime and peak memory usage. Unlike algorithm development studies, this work provides an observational survey of current tool performance to help researchers select appropriate software for their needs. The complete system runs in a Docker container to ensure reproducibility, and all code is publicly available.

Source Data

We used BAliBASE 3.0 as our benchmark database. BAliBASE is a curated collection of protein reference alignments maintained by the European Bioinformatics Institute. Thompson et al. (2005) created these alignments by structurally superposing proteins from the Protein Data Bank, which provides more reliable standards than sequence-based methods. The database contains 386 protein families organized into six reference sets. Reference sets RV11 and RV12 hold equidistant families (125 total). Reference set RV20 contains families with orphan sequences (41 families). Reference set RV30 includes families with divergent subgroups (30 families). Reference sets RV40 and RV50 contain sequences with terminal and internal insertions (190 families combined). Each family includes 4 to 20 sequences with lengths between 50 and 800 amino acids. Each test case consists of two files: an unaligned FASTA file (.tfa format) containing the input sequences and a reference alignment file (.msf format) showing the structurally correct alignment. We used the complete dataset without filtering or subsetting. All 386 families were processed by all five MSA tools, generating 1,930 alignments total. We applied no preprocessing to the input sequences, testing tools on the original BAliBASE data as distributed. The MSA tools evaluated were MAFFT 7.5+, MUSCLE 5+, Clustal Omega 1.2.4, T-Coffee 13, and ProbCons 1.12. These versions were installed as Ubuntu system packages within our Docker environment.

Methodology

We are focused on mainly two experiments: Accuracy Assessment: where We run all MSA tools on the complete BAliBASE 3.0 dataset and compare the generated alignments against reference alignments. Then we measure the sum-of-pairs (SP) score and total column (TC) score to represent alignment quality for each tool, and Computational Efficiency: here, we run each tool across BAliBASE datasets while we monitor system resources to capture the runtime (seconds) and peak memory usage (MB). Then, we analyze computational cost scales with dataset size.

System Implementation

Our MSA benchmarking system operates through four sequential stages, all running inside a Docker container. This design ensures that anyone can reproduce our results on any computer without installing tools manually. The entire pipeline executes automatically when a user runs a single command. Daen setup this whole process (makefile, Dockerfile).

Stage 1: Data Acquisition

The first stage downloads and prepares the test data. When the system starts, the DataFetcher module checks whether the BAliBASE benchmark database exists on the local machine. If the data is missing, DataFetcher automatically downloads the compressed archive from the BAliBASE website. After downloading, the module verifies the file integrity and extracts all contents into the proper directory structure. BAliBASE organizes protein sequences into six reference sets (RV11, RV12, RV20, RV30, RV40, and RV50), where each set tests different alignment challenges. Like we mentioned earlier, the reference set RV11 contains sequences with similar evolutionary distances, while RV20 includes families with one very distant "orphan" sequence. Reference sets RV30 and RV40 test how tools handle subgroups with different divergence levels and long insertions, respectively. The DataFetcher creates separate folders for each reference set and places the

unaligned test sequences (.tfa files) and gold-standard reference alignments (.msf files) in their appropriate locations.

We initially developed a PrepareAPI module to manage tool installation through a YAML-based configuration system, which allows researchers to specify tool versions and download URLs in config.yaml. However, we ultimately scrapped it in favor of using Docker's package manager (apt-get) for tool installation instead. As this provides more reliable versioning and simpler dependency management. The Dockerfile installs all five MSA tools (MAFFT, MUSCLE, Clustal Omega, T-Coffee, and ProbCons) as system packages during container build, which ensures consistent tool availability across different environments. The DataFetcher verifies that all required tools are accessible on the system PATH before proceeding to the next stage.

Stage 2: MSA Execution

Stage two runs each alignment tool on every test sequence and measures computational performance. The MSARunner module serves as the execution engine, implementing specific command-line calls for each of the five tools we evaluate. For example, when running MAFFT, the system invokes mafft –auto input.tfa > output.fasta, which tells MAFFT to automatically select the best algorithm based on sequence characteristics.

The runner processes all 386 test cases from BAliBASE systematically. For each input file, it executes all five tools in sequence, creating separate output files named by tool and sequence identifier (e.g., BB11001_mafft.fasta, BB11001_muscle.fasta). This approach generates 1,930 total alignments across the complete benchmark suite. Resource monitoring happens during tool execution through two mechanisms. The MSARunner wraps each alignment command with the Linux /usr/bin/time -v utility, which tracks wall-clock runtime and peak memory consumption.

Some MSA tools generate auxiliary output files during alignment. Both MAFFT and Clustal Omega create phylogenetic guide trees stored as .dnd files, which represent the evolutionary relationships the tools inferred before aligning sequences. The MSARunner automatically collects these tree files and moves them to a dedicated dnd_files directory, preventing clutter in the main working directory.

The FormatConverter utility handles differences in output formats, using BioPython to read alignments regardless of whether tools output FASTA, Clustal, or other formats. Once again, Ibraheem was responsible for this whole stage.

Stage 3: Alignment Scoring

The third stage evaluates alignment quality by comparing tool outputs against BAliBASE reference alignments. The BaliScorer module implements the standard BAliBASE scoring protocol, which calculates two complementary accuracy metrics. The sum-of-pairs (SP) score measures what fraction of residue pairs are correctly aligned compared to the reference. For example, if a reference alignment shows residue A from sequence 1 aligned with residue B from sequence 2, the tool’s alignment receives credit only if it also aligns these same residues together.

The total-column (TC) score applies stricter criteria by requiring entire columns to match perfectly. A column receives a score of one only when every single residue in that column aligns exactly as shown in the reference alignment. If even one residue is misaligned, the entire column scores zero. This metric reflects

the fact that many downstream analyses, like phylogenetic inference, depend on having completely correct alignment columns.

The system combines accuracy scores with the runtime and memory measurements from Stage 2, creating a complete performance profile for each alignment. These results are stored in a pandas DataFrame with seven columns: reference set, sequence identifier, tool name, SP score, TC score, runtime in seconds, and peak memory in megabytes. The aggregated results are exported to a CSV file (benchmark_results.csv) that serves as input for the visualization stage. Ibraheem and Aelly were responsible for this stage as well.

Stage 4: Visualization and Analysis

The final stage transforms raw benchmark data into interpretable graphics. The ResultVisualizer module loads the CSV file from Stage 3 and generates three types of comparative plots using matplotlib and seaborn libraries. These visualizations help us (researchers) understand both overall tool performance and performance patterns across different alignment 3 scenarios.

The accuracy comparison plot shows boxplots of SP and TC score distributions for each tool across all test cases. The efficiency comparison plot uses the same boxplot format to display runtime and memory usage distributions. This visualization reveals trade-offs between speed and resource consumption. Some tools may produce alignments quickly but require substantial memory, while others run slowly but use minimal resources.

The performance-by-reference-set plot disaggregates results to show how tools perform on different alignment challenges. This line plot displays mean SP scores on the y-axis with reference sets on the x-axis, drawing separate lines for each tool. The plot reveals whether certain tools excel at specific problems. For instance, one tool might handle orphan sequences (RV20) better than others while struggling with long insertions (RV40).

The visualization stage completes the pipeline, delivering both machine-readable data (CSV) and graphics (PNG) that demonstrate benchmark findings clearly. Daen and Aelly both contributed to this stage.

Pipeline Integration

The four stages connect through a main controller script (main.py) that orchestrates the complete workflow. When executed, this script calls each stage in sequence, passing data forward through the pipeline. The Docker container ensures that all dependencies (MSA tools, Python libraries, and system utilities) are available in compatible versions. Users can run the entire benchmark with a single command (make run) without manually managing data downloads, tool installations, or result processing.

This architecture separates concerns cleanly, making the system maintainable and extensible. Researchers can add new MSA tools by extending the MSARunner class with a new method, incorporate additional scoring metrics by modifying BaliScorer, or create custom visualizations by adding functions to ResultVisualizer. The modular design supports both replication of our specific benchmarking study and adaptation to related research questions. Again, Ibraheem was responsible for ensuring the adaptability of the program (and thereby the stage).

Aelly and Ibraheem also worked on the subsequent results, with Aelly taking care of the writing part.

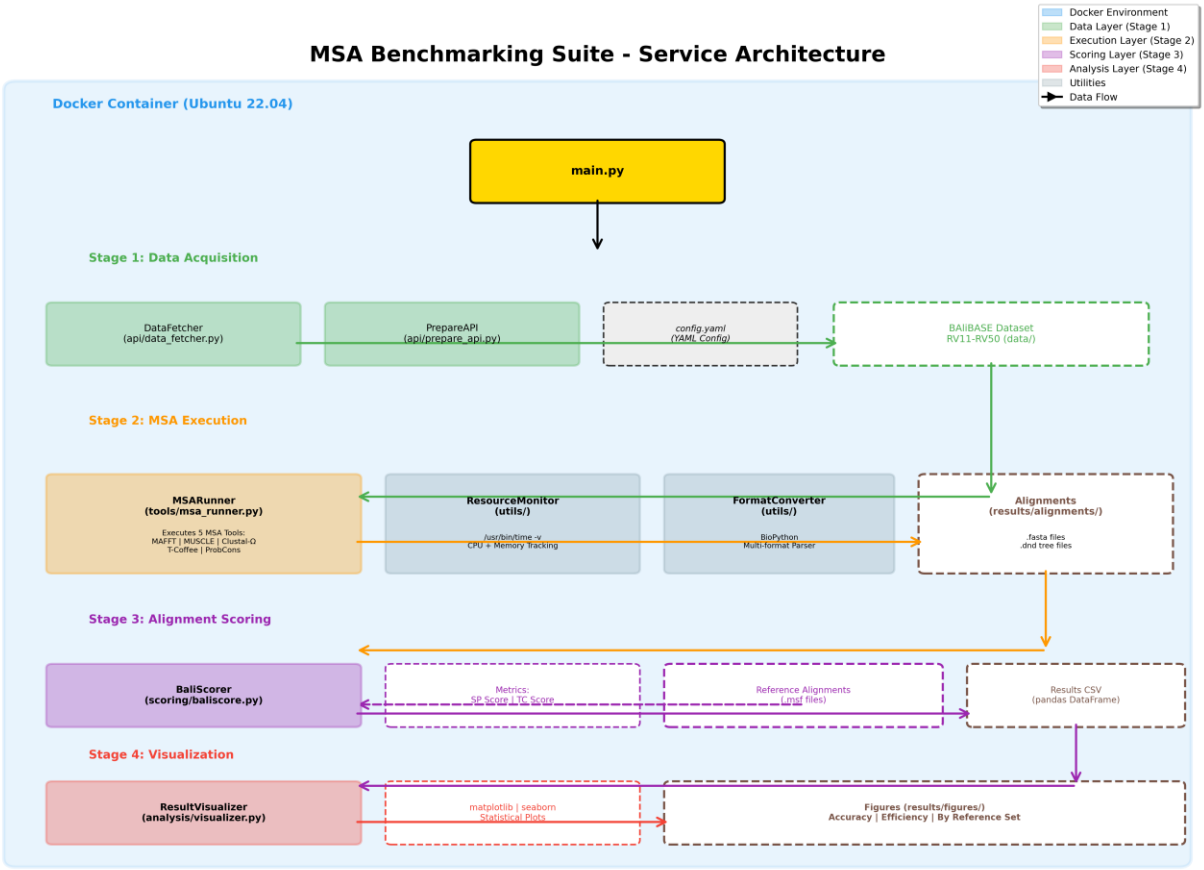

Figure 1: System Architecture Diagram

Results

After benchmarking the SP and TC scores of Clustal, MAFFT, MUSCLE, ProbCons, and T-Coffee, we have found that Clustal Omega had the best mean score in both categories. MUSCLE performs best in terms of resource usage, using the least memory and the shortest time, while T-Coffee uses the most resources.

	SP Score Mean	SP Score Std.	TC Score Mean	TC Score Std.
Clustal Omega	0.693	0.203	0.122	0.196
MAFFT	0.684	0.209	0.038	0.109
MUSCLE	0.690	0.194	0.017	0.054
ProbCons	0.586	0.254	0.103	0.193
T-Coffee	0.596	0.257	0.091	0.178

Table 1: SP and TC Score mean and std. for each tool

	Runtime Mean (s)	Runtime Total (s)	Memory Usage Mean (Mb)	Memory Usage Max (Mb)
Clustal Omega	5.852	2258.885	38.480	941.79
MAFFT	3.247	1253.355	30.256	926.45
MUSCLE	0.712	274.783	13.565	77.50
ProbCons	17.411	6720.699	41.677	528.95
T-Coffee	136.381	52642.879	582.864	2263.68

Table 2: Runtime and Memory Usage mean and std. for each tool

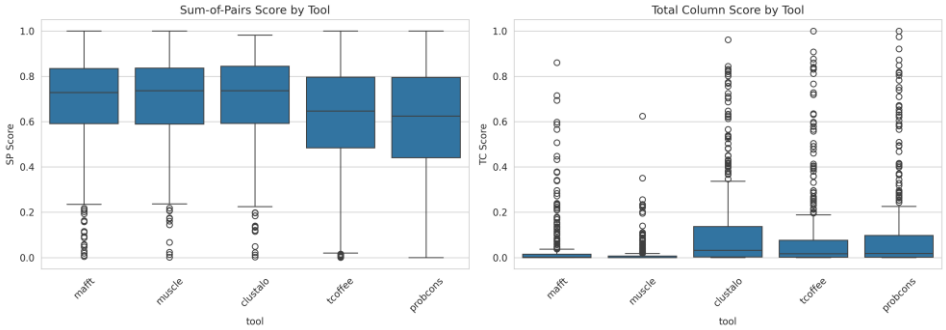

Figure 2: Box plot comparison of SP and TC Score between tools

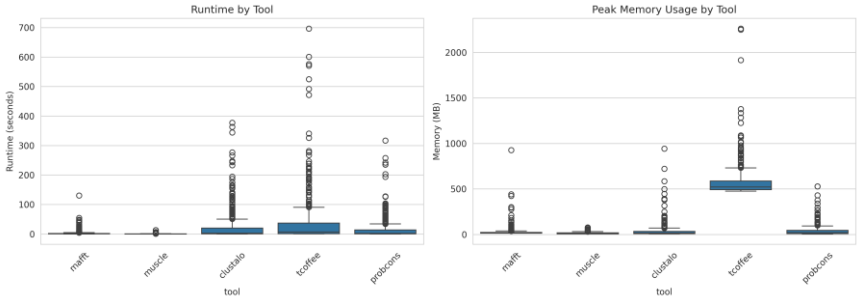

Figure 3: Box plot comparison of time and memory usage between tools

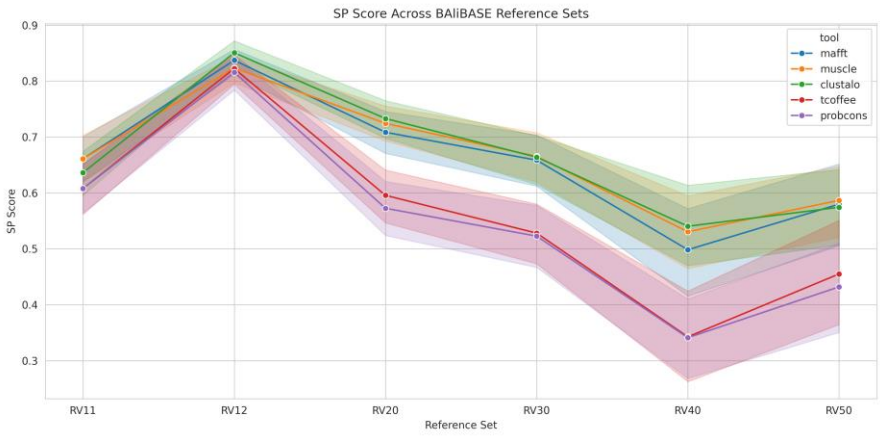

Figure 4: Comparison of SP score between the tools across BAliBASE datasets

Discussion

While Clustal Omega does have the best SP and TC score, the margins of error on Clustal Omega and other tools like MAFFT don't allow us to definitively say that Clustal Omega is the best preforming tool.

While MUSCLE uses the least resources, it also performs the worst in the TC score category. However, MUSCLE's high mean SP score and low margin of error mean it is probably the best choice for parties focused on SP score. Meanwhile, there is no clear best contender for TC score, as Clustal Omega performs the best but doesn't have any significant margin in all metrics.

While we do have our program isolated in a docker container to minimize difference between machines, certain metrics like runtime and memory usage are still affected. However, these differences should be constant throughout the entire program, so each tool's results should be consistently affected by variation in hardware.

Additionally, the program was run on the machines available to us, but these machines may not represent the actual conditions and environment that the alignment tools will be run in. In terms of both hardware configuration and software running.

Conclusion

Our project has effectively benchmarked 5 common software tools used for sequence alignment. We have found that MUSCLE uses the least resources in terms of memory and time and produces one of the best sum-of-pairs score. Meanwhile, there is no clear winner for a total-column score tool, as Clustal Omega performs best when accounting for resource use, but ProbCons is not too far behind. We've also found that T-Coffee uses the most resources and also produces some of the lowest scores, with a runtime of almost 23x Clustal Omega.

Our findings almost definitively show MUSCLE to be the best tool when one is concerned about the SP score, but more studies could be done to confirm our finding. Additionally, a breakdown of runtime and memory usage per computing the SP and TC score would help illuminate more properties of the tools.

Source Code

Source code for this project is available at:
https://github.com/ibrqures-uf/compass

References

1. Pais, F. S., Ruy, P. C., Oliveira, G., & Coimbra, R. S. (2014). Assessing the efficiency of multiple sequence alignment programs. *Algorithms for molecular biology : AMB*, *9*(1), 4. https://doi.org/10.1186/1748-7188-9-4

2. Julie D. Thompson, Desmond G. Higgins, Toby J. Gibson, CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice, *Nucleic Acids Research*, Volume 22, Issue 22, 11 November 1994, Pages 4673–4680, https://doi.org/10.1093/nar/22.22.4673

3. Edgar RC. MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Res. 2004 Mar 19;32(5):1792-7. doi: 10.1093/nar/gkh340. PMID: 15034147; PMCID: PMC390337.

4. Katoh K, Standley DM. MAFFT multiple sequence alignment software version 7: improvements in performance and usability. Mol Biol Evol. 2013 Apr;30(4):772-80. doi: 10.1093/molbev/mst010. Epub 2013 Jan 16. PMID: 23329690; PMCID: PMC3603318.

5. Sievers F, Wilm A, Dineen D, Gibson TJ, Karplus K, Li W, Lopez R, McWilliam H, Remmert M, Söding J, Thompson JD, Higgins DG. Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. Mol Syst Biol. 2011 Oct 11;7:539. doi: 10.1038/msb.2011.75. PMID: 21988835; PMCID: PMC3261699.

6. Thompson JD, Koehl P, Ripp R, Poch O. BAliBASE 3.0: latest developments of the multiple sequence alignment benchmark. Proteins. 2005 Oct 1;61(1):127-36. doi: 10.1002/prot.20527. PMID: 16044462.