TE_Bench Script Descriptions

Workflow Stage 1: DNA Sequence Simulation

Associated Snakefile: Simulation_Snakefile

1. **garlic_model_build.sh**

This helper bash script is used in rule garlic_model_build to submit a SLURM job for createModel.pl with given parameters set by the workflow user.

2. **garlic_sequence_gen.sh**

This helper bash script is used in rule garlic_sequence_generation to submit a SLURM job for createFakeSequence_{dfam or repbase}.pl with given parameters set by the workflow user.

3. **split_inserts.py**

This Python script invoked in Simulation_Snakefile rule split_inserts reads the .inserts file output from the associated GARLIC (1) run and splits it into groups that correspond to simulated sequences using the pattern which precedes each list of inserted TEs (### ARTIFICIAL SEQUENCE (\d+) ###). The final output of split_inserts.py is a set of insert text files, one for every simulated sequence.

4. **seqkit.sh**

This bash script invoked in Simulation_Snakefile rule seqkit_split uses seqkit (version 2.10.1) (2) to read the multi-FASTA base.fasta and .fasta files output from the associated GARLIC run and split them into individual FASTA files, one for each simulated sequence.

Workflow Stage 2: Reference Annotation Generation

Associated Snakefile: AnnotationGen_Snakefile

1. **garlic_gff_gen.sh**

This bash script invoked in Annotation_Snakefile rule run_garlic_gff_generation generates GFF files for the sequences returned from a GARLIC simulation run. The GARLIC run log file outlines the locations and attributes of all inserted TE repeats within the associated simulated DNA sequence(s) which garlic_gff_gen.sh adapts into a readable, field-adopted '.gff' file format. garlic_gff_gen.sh first delineates the log output boundaries for each sequence simulation defined within the GARLIC run log file. Next, each TE insertion instance is counted per simulated sequence and categorized based on nesting. Descriptions of TEs involved in nesting and TEs that are not, including their sequence strings and attributes, are held separately for downstream steps. Then start and stop positions are defined for every TE. GARLIC run log file records the final inserted sequence of each nested element, including internally nested TEs, but does not delineate each internally nested TE's start and stop position within the nested element. To define these positions, the stop location of each full-length nested element is used to pull the full nested sequence and the original TE insert name from the GARLIC run log file to generate a reference FASTA file. Then, the corresponding nested element sequences within each host TE are recorded in a separate query FASTA file using GARLIC-generated commas and brackets as nested sequence delimiters. A NCBI Blast+ (3) blastn step, with stringent -perc_identity 100, -qcov_hsp_perc 100 and -word_size 4 flags, is used on the reference-query FASTA pairs to define the exact location of all TEs included within a nested TE group. The final output of garlic_gff_gen.sh is one 9-column GFF file for each simulated sequence with rows for every TE insertion with the simulated sequence name, simulation program name, repeat name, start position, stop position, score (100), strand, phase (.), and GARLIC-specific TE descriptors. Routine checks are printed as garlic_gff_gen.sh runs to a Snakemake rule run_garlic_gff_generation log file to enable tracking of progress and script behavior.

2. **sequence_start_stop.awk**

This awk script is used by garlic_gff_gen.sh to define the start and stop positions for true nested elements inserted by GARLIC after running NCBI Blast+ blastn.

3. **nested_gff.awk**

This awk script is used by garlic_gff_gen.sh to write the final GFF file lines for nested TE elements in the simulated reference sequence.

4. **Garlic_gff_to_csv.R**

This R script invoked in AnnotationGen_Snakefile rule garlic_to_csv converts a 9-column GARLIC-generated '.gff' file into a standardized 8-column '.csv' file for downstream analysis. The input '.gff' is read as a tab-delimited table and reformatted so that sequence names are replaced with a provided sequence identifier and a constant source label ("Reference (Garlic)"), while genomic start and end coordinates are reassigned to standardized columns. The script parses the original annotation file to extract TE family, type, and subtype information, splitting structured labels into separate columns and handling missing subtype information where necessary. TE classifications are then normalized according to the Dfam classification system into a reduced scheme, merging categories such as SINE and Retroposon into "LINE-dependent," RNA-derived elements into "Non-TE," and resolving additional special cases and unknown entries. The script enforces a final set of valid TE classes and raises an error if unexpected values remain, filters out entries exceeding a maximum coordinate threshold, and writes the cleaned and standardized annotation table to an output '.csv' file.

Workflow Stage 3: Benchmarking

Associated Snakefile: Benchmark_Snakefile

1. **RM2_gff_to_csv.R**

This R script invoked in rule rm_to_csv converts a RepeatModeler2 (4) -generated 9-column '.gff' file into a standardized 8-column '.csv' file for downstream analysis, using both the input '.gff' and a corresponding '.fasta' file for classification recovery. The '.gff' file is read into a data frame, metadata rows are removed, sequence names and source labels are standardized using a provided sequence identifier and "RepeatModeler2," and genomic start and end coordinates are reassigned to standardized columns. Family identifiers are extracted from '.gff' attributes and merged with TE type information parsed from '.fasta' headers, allowing classification labels to be assigned to each element. Subtype information is separated where present, and TE types are normalized according to the Dfam classification system into a reduced scheme in which RNA-derived elements are collapsed into "Non-TE," SINE and Retroposon are merged into "LINE-dependent," and additional cases are standardized or marked as unknown. The script validates that only expected TE classes remain and writes the cleaned annotation table to an output '.csv' file.

2. **EG_gff_to_csv.R**

This R script invoked in rule eg_to_csv converts a 9-column Earl Grey (5) '.gff' file into a standardized 8-column '.csv' file for downstream analysis. The input '.gff' is read into a data frame, sequence identifiers and source labels are standardized using a provided sequence identifier and "Earl Grey," and genomic start and end coordinates are reassigned to standardized columns. Annotation fields are split to extract TE class and subclass information, while family identifiers are parsed from '.gff' attributes when available. The resulting TE type column is then normalized according to the Dfam classification system into a reduced classification scheme in which SINE and Retroposon are merged into "LINE-dependent," simple repeats and low-complexity regions are collapsed into "Non-TE," and other categories are standardized

accordingly. The script validates that only expected TE classes remain and writes the cleaned annotation table to an output '.csv' file.

3. **EDTA_gff_to_csv.R**

This R script invoked in edta_to_csv converts a 9-column EDTA (6) '.gff' file into a standardized 8-column '.csv' file for downstream analysis. The input '.gff' is read into a data frame, sequence identifiers and source labels are standardized using a provided sequence identifier and "EDTA," and genomic start and end coordinates are reassigned to standardized columns. TE classification fields are parsed from structured '.gff' attributes to extract type, subtype, and family/name information. The resulting TE type column is normalized according to the Dfam classification system into a reduced classification scheme in which SINE is collapsed into "LINE-dependent" and MITE into "DNA." The script validates that only expected TE classes remain and writes the cleaned annotation table to an output '.csv' file.

4. **filter_csv.R**

This R script called in selective_filter_csvs filters cleaned 8-column '.csv' files on values and column variables passed by the user, natively LTRs in column V5.

5. **generate_stats.py**

This Python script is called in rules comprehensive_calculate_statistics and selective_calculate_statistics and compares each test annotation '.csv' file to a reference annotation '.csv' file in order to generate performance metrics including Matthew's Correlation Coefficient, sensitivity, specificity, accuracy, precision, FDR, and F1 score. Input '.csv' files are converted into binary vectors representing the presence or absence of annotated regions across the length of the underlying DNA sequence. Predictions are evaluated against the reference to compute true positives, false positives, false negatives, and true negatives at each nucleotide

position. These counts are then used to calculate performance metrics, which are written to an output '.txt' file.

- Note that counts for performance metrics depend on the scale of benchmarking invoked. With comprehensive benchmarking in Stage 3a, any annotated test base pair, regardless of class, will be counted as TP if the base pair is a TE in the reference annotation. With selective benchmarking in Stage 3b, if a TE is classified differently in a test annotation than in the reference annotation, the count will be either a FP or FN. This is because of the behavior of filter_csv.R which natively generates class specific CSV files filtered on the annotation classification column (V5) separately for the reference and test annotations. As an illustration, consider the base pairs of a misclassified reference LTR that is labeled as a LINE element in the test annotation. The base pairs will be present in the filtered reference LTR-specific CSV and not in the complementary filtered test CSV, and the opposite is true for LINEs. The base pairs will be counted as FNs in LTR comparisons and as FPs in LINE comparisons.

6. **view_stats.py**

This Python script is called in rules comprehensive_statistics_radar_plot and selective_statistics_radar_plot. It generates a radar plot to visualize the performance metrics contained in a '.txt' file output by generate_stats.py. The input file is parsed to extract performance metrics for one or more test annotations. These values are plotted on a polar coordinate system to enable direct comparison across test annotations, with each test annotation represented as a distinct line. The resulting figure is labeled using a provided sequence name and saved as a '.pdf' file.

7. **calculate_overlap_csvs.py**

This Python script invoked in rules comprehensive_nest_csvs and selective_nest_csvs identifies and classifies overlapping TEs within a single annotation '.csv' file. TE intervals are converted into a positional count vector across the length of the underlying sequence, where values greater than or equal to 2 indicate overlapping TEs. Each TE is then evaluated to determine whether it overlaps with others and the proportion of its length involved in overlap. Based on this proportion, TEs are classified as fully overlapping (minor nests) or partially overlapping (major nests), and the results are written to separate output '.csv' files for nested host elements and nested elements.

8. **plot_nesting.py**

This Python script is called in rules comprehensive_nesting_plot and selective_nesting_plot. It evaluates and visualizes how well predicted annotations from multiple annotation pipelines cover reference host and nested TEs as well as TEs not involved in nesting. For each test annotation '.csv' file, TE intervals are converted into positional vectors and used to calculate the proportion of each reference TE covered by the predictions. Coverage values are grouped into predefined percentage bins (0-20, 20-40, 40-60, 60-80, and 80-100), and the percentage of reference TEs falling into each bin is calculated for both host and nested TEs. These distributions are then visualized as grouped bar plots for each annotation pipeline and saved as a '.pdf' file.

9. **plot_covered.py**

This Python script invoked in rules comprehensive_coverage_analysis and selective_coverage_analysis evaluates, and for comprehensive benchmarking, visualizes how well predicted annotations in a test annotation '.csv' file cover reference TEs. For each reference TE, intervals are converted into positional vectors and used to calculate the proportion of each element covered by the test annotation. Elements with coverage greater than 80% are classified

as “covered,” while all remaining elements are classified as “uncovered.” The final figure is a nested pie chart showing both overall TE class composition in the underlying DNA sequence and the proportion of covered versus uncovered TEs within each class.

10. plot_covered_third_layer.py

This Python script is supplementary and included in the TE_Bench repository at https://github.com/hkania/TE_Bench/blob/main/scripts/plot_covered_third_layer.py. It extends the functionality of plot_covered.py by adding a third, outermost layer to the nested pie chart returned by plot_covered.py to quantify classification consistency between covered reference TEs and the test TEs that cover them. Specifically, for every reference TE that is covered by the test annotation, if any of the TEs in the test annotation ‘.csv’ file that overlap it are classified as the same type, then the reference TE is counted as “same.” If not, it is counted as “diff.”

11. create_new_csv_id_ana.py

This Python script is called in rule comprehensive_garlic_perc_identity uses the ‘.inserts’ file generated by GARLIC during DNA sequence simulation to calculate the percent identity of each TE in a reference annotation ‘.csv’ file relative to the sequence it was modeled after. The resulting percent identity values are merged with the corresponding entries in the reference annotation ‘.csv’ file based on shared end positions, and a new column containing percent identity is appended to the output file.

12. plot_identity_vs_coverage.py

This Python script in rule comprehensive_garlic_plot_perc_identity evaluates how well a test annotation ‘.csv’ covers individual reference TEs from a reference annotation ‘.csv’ file and relates this coverage to percent identity values provided in the reference file. The test annotation ‘.csv’ file is first converted into a binary vector representing the presence or absence of annotated

regions across the length of the underlying DNA sequence. For each reference TE, the script computes percent coverage by overlapping its genomic coordinates with the test coverage vector and retrieves its class and percent identity from the reference file. Reference TEs are then grouped by class (optionally including user-specified subclasses), and percent identity versus percent coverage is plotted for each class as a scatter plot of individual reference elements, with a linear regression line and Pearson correlation coefficient calculated per class. The final output is a multi-panel '.pdf' figure showing identity-coverage relationships across reference TEs.

13. bpoverlap.awk

This awk script called in rule redundancy_summaries scans test annotation '.csv' files and identifies overlaps between rows based on genomic position and category. It reports both partial and highly redundant (≥95% bp) overlaps, summarizes the total number of overlapping base pairs between category pairs, and tracks the relationship type of each overlap (partial overlap, redundancy, or containment/nesting). It produces three outputs: a '.tsv' file containing all pairs that are partial overlaps, a '.tsv' file containing all pairs that are highly redundant, and a summary '.txt' file that outlines the redundancy in the annotation. Each '.tsv' file includes columns detailing the pair classification (ie. LINE vs LTR), amount of overlapping base pairs, fraction of overlap, and jaccard score where values closer to 1 occupy more of the same genomic space relative to their total combined size and are thus more redundant, and finally the exact base pair positions of the two TEs. The '.txt' file includes the total number of overlapping base pairs in the annotation, overlap totals grouped by category pair (ie. LINE vs LTR) and category pair with relationship type (ie. LINE vs LTR partial), and counts of unique rows participating in overlaps. LINE vs LTR is the same as LTR vs LINE.

14. redundancy_heatmap.py

This python script called in rule redundancy_heatmaps compares genomic interval annotations between a reference dataset and a test dataset, then visualizes overlap relationships between annotation classes as heatmaps. It is designed to assess annotation agreement, misclassification, redundancy, and fragmentation. The script first loads interval coordinates grouped by class (column V5) from '.csv' files. For each class, it computes the total nonredundant genomic coverage by merging overlapping intervals. It then calculates pairwise overlap between classes using interval intersection lengths. For every class pair, the overlap score is calculated as: score(A,B)= min(coverage(A),coverage(B)) / overlap length(A,B). This normalization measures how much the smaller class is represented within the larger one. redundancy_heatmap.py produces a visualization with two heatmaps. The lefthand heatmap compares reference annotations against test annotations and is used to evaluate annotation accuracy and class agreement where a strong diagonal signal indicates good classification consistency and an off-diagonal signal suggests possible misclassification or class confusion in the test annotation. The righthand heatmap compares the test annotation against itself and is used to identify redundancy, fragmentation, or overlap between TE classes within the test annotation. A high off-diagonal overlap suggests classes may annotate the same genomic regions and strong self-overlap can indicate fragmented or internally redundant annotations.

The AWK overlap script and this heatmap script provide complementary views of redundancy:

- The AWK script operates at the individual interval level:
    - identifies exact overlapping elements
    - classifies overlaps as partial, redundant, or nested
    - quantifies overlapping base pairs

    - counts unique overlapping elements
    - useful for detecting local annotation redundancy and duplicated features
- The heatmap script operates at the class-wide coverage level:
    - summarizes overlap patterns across entire annotation categories
    - identifies systematic redundancy between TE families/classes
    - reveals broad patterns of fragmentation or class confusion
    - useful for understanding global annotation structure

Together, the two analyses allow redundancy to be examined at multiple scales:

- interval-level redundancy between specific elements
- category-level redundancy between annotation classes

For example:

- a strong off-diagonal signal in the TEST × TEST heatmap may indicate two TE classes frequently annotate the same regions
- the AWK overlap outputs can then identify the exact intervals responsible for those overlaps and determine whether they are partial overlaps, nested annotations, or near-duplicate redundant calls.

TE_Bench Supplemental Methods

Simulations

Simulated sequences for *Homo sapiens*, *Drosophila melanogaster*, *Saccharomyces cerevisiae*, *Myotis lucifugus*, and *Arabidopsis thaliana* were generated using GARLIC (version 1.4), updated

to match features of GARLIC (version 1.5). Specifically, createModel.pl was updated to assume 0-based coordinates with the remainder of the repository left intact. For each species, createModel.pl was run with default parameters and input files downloaded from the UCSC Browser (7) and affiliated databases (Additional file 3). Then, createFakeSequence.pl was run for each model to generate 1-10 sequences of 100Mb each. Seqkit (version 2.6.1) was run on each output FASTA to split sequences for downstream analysis.

Mapping Strategies

D. melanogaster reference genome

Release 6 of the *D. melanogaster* reference genome FASTA and annotation files (accession GCF_000001215.4) were pulled from NCBI datasets using NCBI-datasets (v15.29.0) (8).

Illumina ReadsFastq files from Bioproject PRJEB51956 Biosample SAMEA13793103 run ERR9466181 were pulled from the NCBI SRA using the SRA-Toolkit (version 3.0.0) (9) with flags --skip-technical --read-filter pass --dumpbase --split-3 --clip and --gzip.

PacBio HiFi Reads

Fastq files from Bioproject PRJNA636174 Biosample SAMN15065810 run SRR11906525 were pulled from the NCBI SRA using the SRA-Toolkit (version 3.0.0) with flags --skip-technical --read-filter pass --split-files --clip and --gzip.

Trimming

Illumina reads were trimmed using trimmomatic PE (version 0.39) (10) with flags SLIDINGWINDOW:4:20 MINLEN:50 and built in NexteraPE-PE.fa:2:30:10 adaptor sequences.

Mapping

Filtered and trimmed Illumina reads were mapped using bwa mem (version 0.7.17-r1188) (11) and the -M flag. They were sorted, de-duplicated, and indexed into a final bam file using samtools view, fixmate, sort, markdup, and index (version 1.17) (12). Filtered PacBio reads were mapped using bwa mem (version 0.7.17-r1188) with the -x pacbio and -M flags. Filtered PacBio reads were also mapped using minimap2 (version 2.24) (13) with the -ax map-pb flags and pbmm2 align (version 1.14.99, installed with minimap version 2.26) with --sort -j 12 -J 4 --rg '@RG\tID:myid\tSM:SRR11906525' flags. BWA mapped reads were sorted, de-duplicated, and indexed into a final bam file using samtools view, fixmate, sort, markdup, and index (version 1.17) to return three distinct bam files, one from each PacBio mapping schema.

Consensus sequence generation

Consensus sequences from all alignments were pulled using samtools consensus (version 1.17) with the -f fasta flag. The --mode simple flag was used for PacBio bam alignments, as the PacBio reads lack ASCII quality scores.

D. melanogaster reference GFF file generation

The *D. melanogaster* reference transposable element library was pulled from Repbase (14), a database regarded as having the highest quality consensus sequences. RepeatMasker (version 4.1.6) (15) was run in sensitive mode with the input library from Repbase to mask the reference genome. The output GFF file was run through a custom script (append_class.sh) to generate the final GFF file for downstream comparison.

Transposable Element Discovery

Arrays were submitted to identify TEs within repeat-containing sequences using all three annotation pipelines. Each query sequence was submitted to RepeatModeler2 with 16 threads (version 2.0.5), EDTA with 40 threads (version 2.2.0), and Earl Grey with flags with 16 threads

(version 4.4.0, configured to include all Dfam partitions and rmblast version 2.14) for TE identification. The -LTRStruct flag was used in RepeatModeler2, and EDTA annotations were built from an iterative approach where each category of TE was identified separately. GFF files produced by EDTA and Earl Grey were used as queries for comparison to other pipelines. Generated consensus repeat sequences from RepeatModeler2 were then used as input for RepeatMasker (version 4.1.7) to build query GFF files with 16 threads and the -gff flag.

Method References

1. Caballero J, Smit AFA, Hood L, Glusman G. Realistic artificial DNA sequences as negative controls for computational genomics. Nucleic Acids Research. 2014 July 8;42(12):e99.

2. Shen W, Sipos B, Zhao L. SeqKit2: A Swiss army knife for sequence and alignment processing. iMeta. 2024;3(3):e191.

3. Camacho C, Coulouris G, Avagyan V , Ma N, Papadopoulos J, Bealer K, et al. BLAST+: architecture and applications. BMC Bioinformatics. 2009;10(1):421.

4. Flynn JM, Hubley R, Goubert C, Rosen J, Clark AG, Feschotte C, et al. RepeatModeler2 for automated genomic discovery of transposable element families. Proceedings of the National Academy of Sciences. 2020 Apr 28;117(17):9451–7.

5. Baril T, Galbraith J, Hayward A. Earl Grey: A Fully Automated User-Friendly Transposable Element Annotation and Analysis Pipeline. Molecular Biology and Evolution. 2024 Apr 1;41(4):msae068.

6. Ou S, Su W, Liao Y , Chougule K, Agda JRA, Hellinga AJ, et al. Benchmarking transposable element annotation methods for creation of a streamlined, comprehensive pipeline. Genome Biology. 2019 Dec 16;20(1):275.

7. Casper J, Speir ML, Raney BJ, Perez G, Nassar LR, Lee CM, et al. The UCSC Genome

Browser database: 2026 update. Nucleic Acids Res. 2025 Nov 18;gkaf1250.

8. O'Leary NA, Cox E, Holmes JB, Anderson WR, Falk R, Hem V , et al. Exploring and retrieving sequence and metadata for species across the tree of life with NCBI Datasets. Sci Data. 2024;11(1):732.

9. Sayers EW, Beck J, Bolton EE, Brister JR, Chan J, Connor R, et al. Database resources of the National Center for Biotechnology Information in 2025. Nucleic Acids Res. 2025;53(D1):D20–9.

10. Bolger AM, Lohse M, Usadel B. Trimmomatic: a flexible trimmer for Illumina sequence data. Bioinformatics. 2014;30(15):2114–20.

11. Li H. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. http://arxiv.org/abs/1303.3997 (2013). Accessed 17 Mar 2025.

12. Danecek P, Bonfield JK, Liddle J, Marshall J, Ohan V , Pollard MO, et al. Twelve years of SAMtools and BCFtools. GigaScience. 2021;10(2):giab008.

13. Li H. Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics. 2018;34(18):3094–100.

14. Bao W, Kojima KK, Kohany O. Repbase Update, a database of repetitive elements in eukaryotic genomes. Mobile DNA. 2015 June 2;6(1):11.

15. Smit AFA, Hubley R, Green P. RepeatMasker Open-4.0. 2013.