XTree enables memory-efficient, accurate short and long sequence alignment to millions of genomes across the tree of life

Gabriel A Al-Ghalith*[1,2], Krista A Ryon[1], James R Henriksen[1,3], David C Danko[4], Brett Farthing[5], Matteo Marengo[6], George M Church[1,7], Raquel S Peixoto[8], Chirag J Patel[1,6], Dan Knights[9], Braden T Tierney*[1,6]

1 The Two Frontiers Project, Fort Collins, CO, USA
2 Holobiome, Boston, MA, USA
3 National Renewable Energy Laboratory, Colorado State University, CO, USA
4 Biotia Inc., New York, NY, USA
5 Zymo Research, Irvine, CA, USA
6 Department of Biomedical Informatics, Harvard Medical School, Boston, MA, USA
7 Wyss Institute for Biologically Inspired Engineering, Harvard Medical School, Boston, MA, USA
8 Division of Biological and Environmental Science and Engineering (BESE), King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia
9 Biotechnology Institute, University of Minnesota, Saint Paul, MN, USA

*Corresponding authors:

Braden Tierney, braden@twofrontiers.org
Gabriel Al-Ghalith, gabe@holobiome.org

Abstract

XTree is a k-mer-based aligner enabling rapid, memory-efficient alignment of sequencing reads to whole-genome reference databases with up to millions of genomes. Here, we detail XTree's performance on short and long read sequencing data and demonstrate its high accuracy across diverse bacterial, viral, and eukaryotic genomes. Benchmarking demonstrates superior and/or comparable precision and recall over existing tools, with more scalable indexing and efficient memory mapping. We additionally provide pre-indexed databases, including (1) the Genome Taxonomy Database (versions r214-r226), (2) representative GenBank fungi and protozoan genomes and (3) the Pan-Viral-Compendium, a bespoke data resource spanning 6.6 million, quality-controlled, viral genomes.

Main Text

The alignment of Next Generation Sequencing (NGS) data to reference genomes is foundational to modern genomics. Mapping bulk sequencing reads to large databases of known genomes underpins our ability to quantify the genomic and taxonomic diversity of complex ecosystems. However, alignment algorithms designed many years ago do not necessarily: (1) computationally scale to today's large reference databases; (2) guarantee high sensitivity and specificity with continually increasing species diversity across the domains of life, nor; (3) perform well on newer sequencing methods (e.g., long read sequencing).

Here, we present XTree and its accompanying alignment databases. XTree (see *Methods*) is a k-mer-based aligner designed for memory-efficient (and virtual/out-of-core memory) parallel alignment of long or short sequencing reads to millions of genomes. Its earlier incarnations (e.g., BURST, Utree) have been used in both academia and industry over the past decade[1–11]. XTree extends upon prior k-mer based genome indexing and read assignment strategies[12,13] (Fig 1) by using: (1) memory mapping for efficient database usage by simultaneous processes; (2) motif-based sequence indexing (as opposed to using minimizers); (3) canonicalization of reference k-mers into a binary search tree for quick look ups, and; (4) "capitalist" read redistribution, which involves storing all possible read alignments and identifying the set of references that minimize all matches.

XTree additionally identifies multiple types of k-mers and reports associated coverage "types" with each: unique k-mers are any found specifically in a single genome, whereas Adamantine, or "maximally unique" k-mers are those that are a Levenshtein distance of > 1 (i.e., more than 1 base edit) away from all other references, enabling strain-specific classification. By default, XTree reports total coverage of a genome (proportion of an entire genome covered), unique

coverage (unique regions of a genome covered), and Adamantine coverage (Adamantine regions of a genome covered).

To showcase XTree's utility and guide its use, we evaluated its alignment performance on representative Genome Taxonomy Database (GTDB) genomes indexed at different k-mers (Supp Fig 1A). We used 200 synthetic metagenomes spanning species simulated at high and low relative abundances. Performance remained consistent for k ≥ 21, with k = 29 yielding the highest mean F1 (Table 1, Supp Table 1, Supp Fig 1B). XTree achieved near-perfect performance on representative (i.e., "training") genomes (k=29, mean F1 scores for high and low abundance genomesets: 0.992+/-0.006 and 0.958+/-0.014, respectively) and similarly strong results on non-representative (i.e., "testing") strains (mean F1 scores: 0.920+/-0.023 and 0.891+/-0.028), despite the fact that they were not included when the XTree database was indexed.Optimal F1 coverage thresholds varied by abundance—3% unique, 16% total for low; 9% unique, 36% total for high (Supp Fig 1C). Unique coverage (the minimum of both Adamantine and "weakly unique" (see *Methods*) coverage), minimized false positives while maximizing true positives (Fig 1B).

We compared XTree to Kraken2/Bracken and MetaPhlAn4 on the same synthetic metagenome testing set[12,14,15] (Table 1). We chose these approaches due to their frequent usage in metagenomics; specifically, we included Kraken2 because it is also a kmer-based method that is frequently used for pan-domain alignment. Constructing a 716GB Kraken2 database on the GTDB took over three days, whereas XTree completed indexing in <3 hours with the same

resources, producing a more compact 260GB database (for k=29, compression mode 2). MetaPhlAn4, a marker-gene-based approach using Bowtie2, was tested with its default database mapped to GTDB taxonomy.

Across all tests, XTree performed comparably to or outperformed other methods (Fig 1C, Table 1, Supp Table 1). Kraken2, without coverage thresholds, had high recall but low precision (mean F1 score, high abundance non-representative genomes: 0.124+/-0.009; mean F1 score, low abundance non-representative genomes: 0.0235+/-0.015); applying a coverage threshold improved classification performance (mean F1 high abundance non-representative genomes: 0.759+/-0.041; mean F1 low abundance non-representative genomes: 0.759+/0.041). XTree additionally outperformed MetaPhlan4, which had poor performance on GTDB representative strains and better performance on non-representative strains. Comparing expected versus predicted relative abundances for true positive classifications, XTree, MetaPhlan4, and Kraken2 (via Bracken) performed similarly in accurately recapitulating expected relative abundances (Supp Fig 1D).

We next analyzed how F1-optimized unique and total coverage thresholds varied across bacterial orders to guide species and strain abundance estimation in complex communities. While a threshold of <10% unique coverage was sufficient for most organisms (Fig 1D), we found that for optimal total coverage thresholds, clades with reported larger pangenomes (e.g., *Cyanobacteriales*) required different cutoffs to maximize F1 score. Thus, we propose XTree's unique coverage as a robust threshold for presence/absence detection across diverse bacterial clades and suggest its F1-optimized cutoffs offer a scalable approach to quantifying pangenome size.

To assess the impact of database memory mapping on efficiency, we compared XTree and Kraken2 alignments across 15 samples with varying parallelism (Fig 1E). XTree's default

memory mapping minimized RAM increases as parallel tasks rose from 1 to 3, requiring 260GB for the database and 40–50GB per alignment. In contrast, Kraken2's median RAM usage doubled with additional tasks running the default settings. XTree completed runs in about one-third of Kraken2's runtime while using, maximally, 40% of its RAM (381GB for 3 tasks). Notably, XTree's disk-backed memory mapping enables alignment on lower-memory systems using SSDs, albeit with a speed tradeoff. We additionally note that, with specific parameters and utilizing RAM disk storage, Kraken2 can be made more efficient, however, this is not its default settings, as opposed to XTree, which will automatically optimize memory usage for a given system.

We then evaluated XTree on long read sequencing data from multiple platforms, comparing PacBio's Hi-Fi reads, PromethION data, and GridION data. In lieu of generating synthetic metagenomes, we used real data on microbial mock communities, a subset of which were used in a previous aligner benchmarking study (Fig 1F-G)[16]. Precision and recall were high on average (overall mean precision = 0.972, overall mean recall = 0.883), especially for Hi-Fi data, but we did find that the log-distributed samples and those on older Nanopore technologies suffered poorer performance. XTree did perform well compared to the published benchmarks on the same datasets[17], and we therefore claim that it is a viable tool for long read sequencing, especially for modern chemistries with low error rates.

To demonstrate XTree's ability to work with viral genomes, we developed the Pan-Viral Compendium (PVC) by integrating nine published datasets spanning human and environmental viromes comprising 2,851,990 non-redundant genomes (Supp Table 2, see *Methods*). We provide various subsets by genome quality (e.g., complete/high genomes only) as raw FASTA files and indexed XTree databases at multiple k-mer lengths. Additional PVC metadata includes genome composition (DNA vs. RNA viruses), geNomad and GenBank taxonomies, 90% identity

genome clustering, non-redundant gene catalogs, and Pfam/KEGG HMM-based protein annotations (Supp Fig 1F-G).[18–21]

XTree databases indexed at k-mer lengths >17 performed well for viral synthetic metagenomes (Fig 2A), with unique and total coverage cutoffs yielding similar F1 values (Supp Fig 1E). A database indexed with k=21 performed the best (as opposed to k=29 for bacteria). We also assessed performance on a challenging set of non-representative PVC genomes, which included fragmented sequences likely to introduce noise. Despite this, classification remained strong at higher taxonomic levels (e.g., Family, Class, Order, Phylum, Fig 2B) using a 1% unique coverage cutoff. As Kraken2 could not feasibly index the full PVC, we used its standard database for comparison at broader taxonomic levels. While not a direct benchmark, the recall differences suggest the PVC captures viral diversity absent from conventional reference databases. We additionally observed concordance between expected and observed abundance for both representative and non-representative genomes (Fig 2C).

Finally, we additionally indexed and completed a brief evaluation of multiple Eukaryotic alignment databases. Specifically, these included the full set of dereplicated, representative, complete and draft fungal (n = 6,317) and protozoan (n = 1,054) genomes from GenBank as of May and April 2024, respectively. We evaluated the ability of XTree to accurately classify the presence and abundance of genomes from these datasets in ten synthetic metagenomes and found the results to be satisfactory (Fig 2D-E). Mean precision and recall for fungal genomes were 0.96+/- 0.09 and +0.86+/-0.27, respectively. These values were 0.94+/-0.13 and 0.95+/-0.05 for protozoan genomes.

In summary, in this manuscript, we describe XTree, an aligner built for robust and flexible whole-genome-based classification across diverse use-cases. The codebase, as well as download links for all databases and synthetic data in this study, are fully available at

https://github.com/two-frontiers-project/2FP-XTree. We hope the scientific community finds these to be impactful tools.

Figures and Tables

Figure 1: Description, optimization, and evaluation of XTree on short and long read, bacterial metagenomes. A) Overview of the XTree algorithm. XTree utilizes k-mer-based indexing with motif-based compression, canonicalization via binary search trees, and "capitalist" read redistribution for optimal read assignment, as well as multiple options for Lowest Common Ancestor (LCA) analysis. B) The utility of total vs. unique coverage thresholds on minimizing false positives. Each point represents a different genome in a sample, and red lines indicate optimized coverage cutoffs for high and low-abundance genomes. C) Comparison of XTree, Kraken2, and MetaPhlAn4 on classification against GTDB non-representative genomes. D) Optimizing XTree coverage cutoffs across discrete clades E) Memory usage over time for processing 15 gut metagenomes for XTree and Kraken2 across different levels of parallelism. F) XTree's performance on long read sequencing of defined communities. G) Observed vs. expected abundance of bacterial genomes for long read sequencing.

Figure 2: XTree's performance on viral and eukaryotic genomes. A) The utility of total vs. unique coverage thresholds on minimizing false positives when aligning to different indexed versions of the Pan Viral Compendium. Each point represents a different genome in a sample, and red lines indicate coverage cutoffs that maximize F1 score. The databases used were indexed with a k-mer of 21. C = Complete genomes, H = High-quality, M = Medium-quality, L = Low quality. B) Comparing the performance of Kraken2 and XTree on classifying viral genomes. C) The expected versus observed relative abundances generated by XTree on representative (e.g., training) and non-representative (e.g., testing) viral genomes. D) The performance of XTree on GenBank fungal and protozoal genomes. E) The expected versus observed relative abundances generated by XTree on GenBank fungal and protozoan genomes.

	XTree	**Kraken2**	**Metaphlan4**
Classification approach	Exact multi-mapping k-mer classification and pseudoalignment on whole genomes/genes	K-mer–based classification using LCA on minimizers	Bowtie2 on marker genes
Intended input data type	Short/long reads, genes, genomes	Short reads	Short reads
Intended domains for alignment	Bacteria, Eukaryotes, Viruses	Bacteria, Eukaryotes, Viruses	Bacteria
Primary database structure	Prefix-hashed left binary search tree	Hash map	Bowtie2 index
Database size on disk	260GB (GTDB r214, comp=2, k =29)	716GB (GTDB r214)	23G (default DB)
RAM usage for 3 parallel processes, (default memory settings), classifying against GTDB	399GB	1014GB	N/A (very low)
GTDB alignment results, representative, high abundance genomes	Mean F1: 0.992 +/- 0.006	Mean F1: 0.977+/0.012	Mean F1: 0.418+/-0.067
GTDB alignment results, non-representative, high abundance genomes	Mean F1: 0.920 +/- 0.023	Mean F1: 0.759+/-0.041	Mean F1: 0.407+/0.067
GTDB alignment results, representative, low abundance genomes	Mean F1: 0.958 +/- 0.014	Mean F1: 0.739+/0.042	Mean F1: 0.832+/-0.033
GTDB alignment results, non-representative, low abundance genomes	Mean F1: 0.891 +/- 0.028	Mean F1: 0.759+/0.041	Mean F1: 0.807+/-0.035

Long read alignment results	Mean F1: 0.916+/-0.122	N/A	N/A
Viral alignment results, PVC representative genomes	Mean F1: 0.989+/-0.00812	N/A	N/A
Viral alignment results, PVC non-representative genomes	Mean F1: 0.743+/-0.0358	N/A	N/A
Viral alignment results, taxonomies (all ranks, PVC for XTree and default viral database for Kraken2)	Mean F1: 0.951 +/- 0.0541	Mean F1: 0.347+/-0.140	N/A
Fungal alignment results	Mean F1: 0.884+/-0.207	N/A	N/A
Protozoan alignment results	Mean F1: 0.946+/-0.0905	N/A	N/A

Table 1: Comparing the algorithms and performance of XTree, MetaPhlan4, and Kraken2. All outputs are reported for the default settings for MetaPhlAn4 and 5% coverage cutoffs for XTree (referring to unique coverage) and Kraken2, respectively. The XTree values report the k=29 index. Additional details on aligner performance in Supplementary Table 1. GTDB = Genome Taxonomy Database.

Supplement

Supplemental Figure 1: Additional information on experimental design and alignment statistics. A) Synthetic metagenome and database generation process. Green boxes were training sets/databases; orange is the “testing sets” of genomes XTree had not been trained on. B) Performance of XTree with different kmer sizes. We recommend tailoring kmer choice to the domain of life analyzed and keeping kmers above 17. C) Precision-Recall curves showing F1-optimized unique/total coverage values for bacterial genomes indexed at k = 29. D) Expected vs. observed abundance across aligners. E) K-mer optimization for viral genomes. F) Gene catalog summary statistics for the Pan-Viral-Compendium.

Supplementary Table 1: Additional aligner comparison and performance statistics to provide more context for Table 1.

Supplementary Table 2: List of databases found in the Pan Viral Compendium.

Methods

XTree databases and indexing

XTree is a k-mer-based aligner. It matches reference sub-sequences of length k – determined during the database build step – to kmers found in input sequences. These input sequences can be in FASTQ and FASTA form, paired or single-ended, and of any length. The general process for the XTree algorithm, both database construction and alignment, is described in Figure 1.

Databases can be constructed of any set of contigs in FASTA format. We recommend high-quality genomes that are dereplicated at a level between 90% and 99% nucleotide identity (e.g., 95% and 99% for bacterial and viral genomes, respectively). We do not recommend constructing databases from multiple domains of life, as the optimal k-mer length or dereplication cutoff may vary depending on genome size and complexity. Genome dereplication is important prior to database generation as it enables the identification of species or strain-specific k-mers, which in turn inform a user's ability to design coverage cutoffs.

Databases are built using a motif-based search, with the motifs selected varying depending on the compression level (i.e., number of kmers) selected. At varying compression levels, different motifs are used to determine which kmers are preserved in the index (sampling stride); the motifs in question were chosen to capture a wide range of genomic sequences while reducing the redundancy of stored information (i.e., minimizing tightly overlapping k-mers while retaining comparability of the same k-mers regardless of context). For compression level 0, all kmers of a given length are stored, resulting in larger database sizes. For compression level 1, all kmers preceded by an A are stored. For compression level 2, all kmers following AT, AC, or AG 2-mers are stored. This results in a compression level 2 database size being similar to the input fasta file used to generate it.

XTree currently stores all k-mers that meet these motif-based criteria, and it additionally can track k-mers that appear in a single reference as being "unique" and/or "Adamantine." Unique kmers are those found in a single genome. Adamantine kmers are a subset of these; each Adamantine kmer is more than 1 base pair edit (Levenshtein distance of 2 or greater) from all other kmers present in other genomes; this makes Adamantine k-mers a more ironclad indicator of that specific reference, rather than a simple off-by-one sequencing error/small mutation/noise flip. We propose them to be useful for tracking strains when you have multiple similar strains, as sequencing noise can often flip a k-mer to appear like it's from another similar genome. We do not recommend using Adamantine k-mers in cases where strain-level specificity is not necessary.

During alignment, query k-mers are canonicalized into a binary format consisting of a binary prefix (for use as initial perfect hash index) and a suffix, which is looked up within a binary tree stored at that prefix. We use a flattened binary tree due to its efficiency of data structure, with lower RAM usage than a full hash map given its ability to assemble full kmers on the fly using only their stored suffices combined with the tree structure. Effectively, lookup consists of an initial perfect hash lookup of the binary prefix, which provides the index range within the flattened tree array containing suffix-keyed leaves, within which the (first) suffix match is identified by left-first binary search (LBS) within that range.

The completed lookup provides an index into all matching genomes and associated metadata (e.g., source genome identifier, the source genome's taxonomy, etc). If present in the database structure, a secondary "Adamantine" tree is also searched by default for each unique k-mer to determine whether the k-mer is "weakly unique" (e.g., found within 1 Levenshtein distance of a

k-mer in another genome); if a unique k-mer is not specified as weakly unique, it is Adamantine (strongly unique). The adamantine tree is generated by default on database construction, though, as mentioned above, for analyses not reliant on strain-level differentiation, it can be omitted during the database build step (or not utilized during alignment even if present in the database), enabling downstream efficiency gains during alignment.

If a taxonomy map is provided in the database BUILD process, two forms of taxonomy output are available in ALIGN mode using the --tax-out flag. Using the flag will default to using Lowest Common Ancestor (LCA) taxonomic assignment by performing a root-to-leaf interpolation among each query's set of identified matches, adhering to the user-specified confidence (--confidence) level at each level of the hierarchy; a simple leaf-only voting scheme will be used instead if --shallow-lca is specified[22]. Alternatively, the Capitalist read redistribution algorithm will be used for both taxonomy assignment and reference re-assignment if --redistribute is specified, similar to its use in BURST but using an expectation maximization step to approximate BURST's graph-pruning method for simplicity[1]. When Capitalist is enabled with --tax-out, taxonomy will be reported at the leaf node of the winning taxonomic label (highest specificity).

During the alignment process, a given database is mapped to memory, enabling multiple XTree processes on the same machine to simultaneously use a single copy of the loaded database in dynamic shared memory backed by a single location on physical disk, reducing overall memory footprint, enabling "warm starts" for subsequent runs, and allowing for simultaneous instances to run. Small databases can alternatively be copied to CPU-local memory (--copymem), providing each process greater locality of reference; on some architectures (such as multi-socket systems for which NUMA controls are not enabled) or for some very small databases that can partially fit within L3 cache, this may be preferred. We note that as CPU-local memory sizes increase, XTree's performance will increase for proportionally larger databases. Notably, due to memory mapping, XTree does not strictly require the entire database to fit within RAM at one time. Users with limited memory (as low as <= 64GB) but with fast direct NVME SSD storage may run XTree without issue but incur a speed penalty approximately proportional to both the difference between the size of the database and the free RAM, and the speed differential between the SSD random read performance and system RAM bandwidth.

XTree alignment, read assignment, and memory efficiency

Upon ingestion of reads, XTree begins its k-mer search in the same manner as it approaches database indexing. All k-mers for a given read with the appropriate motifs (depending on the reference database's compression level) are stored in memory. These k-mers are matched to the canonicalized reference database and the appropriate leaves of the binary search tree are searched. Each potential match to a given reference genome is stored.

By default, XTree is run in "Capitalist Redistribution" (i.e., winner takes all) mode, where the space of all possible k-mer matches are searched and the set of assignments that minimize the total number of references assigned is selected. Capitalist is a global read redistribution algorithm, as opposed to a per-read greedy assignment method. Its purpose is to resolve multi-mapping reads across the entire dataset simultaneously, as opposed to doing so for every query at once. It leverages the fact that XTree records all possible reference hits for every k-mer in every read, producing a large, multi-mapped space describing which reads could feasibly derive from which sources. Capitalist performs an expectation-maximization procedure over this space; in each iteration, reads are provisionally assigned to the reference that is most strongly supported by other reads across the entire dataset. As assignments aggregate, the relative support for competing references shifts accordingly. This process continues until convergence on a given reference, or until an iteration threshold is reached, producing a one-to-one

assignment in which a read is assigned to a single reference. The phrase “winner takes all” refers to this globally considered outcome, as opposed to a per-read or per-reference approach. Therefore, capitalist redistribution is approximating the solution to a global set-cover-like problem without explicitly solving an NP-complete formulation, using dataset-wide competition among references to converge on a parsimonious explanation across all reads, jointly.

This behavior is distinct from other approaches (e.g., fast read mappers or Kraken2), which rely on per-read best-hit (or stored LCA-based) assignment modes. Other read redistribution techniques (e.g. Bracken) use post hoc abundance redistribution without access to information on each read's specific multi-mappings, and do not explicitly leverage the set of multi-mappings across all queries in a dataset for making assignment decisions. If Capitalist Redistribution mode is disabled, the reference genome with the highest number of k-mer matches is simply assigned (with "first stored reference" as a tie breaker).

If XTree is run with the --tax-out command (without Capitalist Redistribution), XTree will independently compute the LCA among all reference genomes matched by the query. Importantly, LCA is not stored per reference k-mer based on a pre-computed database; it is computed among the specific matches for each query during alignment. Specifically, the LCA taxonomic assignment of a read is computed after the k-mer lookup stage by traversing the taxonomic subtree formed by all references with matching k-mers in the read; if --tax-out is run with Capitalist Redistribution, the taxonomic call of a read is also determined based on the full-dataset-aware read assignment on the taxonomic string provided; in tandem with the reference-level capitalist redistribution, this enables multiple independent granularities of assignment for a single query, leveraging the different scales of evidence available at different set granularities.

XTree outputs and coverage thresholding

In addition to the timing and alignment statistics printed to stdout by default, XTree generates up to four output files depending on user specification: tax, cov, ref, and perq. The tax file, as previously described, contains taxonomic information in a simple tab-delimited format of key-value association (reference, followed by semicolon-delimited hierarchical string). To use "--tax-out" mode, the taxonomy map must be supplied during database construction (we plan for future versions to allow supplying an arbitrary taxonomy at runtime). The taxonomy file may contain up to two hierarchies (e.g., two different taxonomic schemas or granularities, or a taxonomy and a GO hierarchy if genes are used as references instead of genomes), both of which will independently be used for alignment summarization in the tax output file. The cov file contains a complete set of total, unique, and Adamantine breadth-of-coverage and depth-of-coverage (i.e., pileup) statistics for each genome or reference sequence, separated into total k-mer coverage, unique, and adamantine. Since specific coordinates of alignments are not stored, these values are approximated via kmer set coverage and multiplicity of coverage per reference of different kmer types (e.g., total adamantine coverage, total unique coverage, total coverage).

Depth of coverage can be interpreted as read pileup, which can be used to generate relative abundance estimates. The ref file contains a simple table of references matched, and the number of queries assigned to each.. Finally, the perq out file will report the potential assignments of every read, enabling users to further understand their alignment results (e.g., filtering multi-mapping reads). Note that each output type is only computed if it is requested by the associated --out flag, to avoid unnecessary bookkeeping.

By default, XTree does not threshold based on coverage in order to determine genome presence and absence. While this step is critical for determining meaningful compositional analysis downstream, the decision is left up to the user, as the selected threshold will vary

based on sampleset and tolerance to noise. Based on the analysis in this manuscript, we recommend a 1% to 10% threshold derived from the minimum of total, unique, and Adamantine coverage. Recall that, for example, 1% adamantine coverage does not correspond to covering 1% of the genome; it instead indicates 1% of coverage of the ultra-unique (adamantine) regions of the genome. We provide scripts for assisting with coverage thresholding in the GitHub repository.

XTree *pseudocode*

Below is pseudocode for the XTree algorithm:

Inputs:

1. Reference genomes G = {g_1 … g_N}
2. Query reads Q = {q_1 … q_M}
3. k-mer length K (BUILD mode)
4. Prefix length PL (BUILD mode)
5. Compression level C (BUILD mode)
6. Optional taxonomy mapping T (BUILD mode)

<u>BUILD:</u>

Construct lookup data structure PrefixBins with length = all possible prefixes

1. For each reference genome g_i in G:
    a. Stream through g_i and extract canonical k-mers, defined as the lexicographically smaller of the forward k-mer and its reverse complement.
    b. Apply motif-based subsampling according to compression level C.
    c. For each retained k-mer, split it into a prefix pfx of length PL and a suffix sfx.
    d. Append the pair (sfx, i) to the bin PrefixBins[pfx][] corresponding to prefix pfx.

For each prefix bin:

- Sort all stored (sfx, i) pairs lexicographically by sfx, then by reference index i.

If adamantine enabled, construct the adamantine disqualifier tree:

1. Initialize a new prefix-bin data structure WeakBins with length = all possible prefixes.
2. For each reference genome g_i in G:
    a. Extract to identify unique k-mers by examining multiplicity of suffixes within each prefix bin in PrefixBins.
    b. For each unique k-mer, determine weakly unique status
        i. Generate all possible singly-mutated k-mers, k_mut
        ii. Look up each k_mut in PrefixBins. Consider each returned tuple (sfx,j).
        iii. If tuple exists and i != j, add to WeakBins[pfx] (it is "weakly unique").

Store the database as:

- A prefix offset array mapping each prefix to a contiguous suffix range.
- A flat, suffix-sorted array of (suffix, reference index) pairs.
- Reference metadata and optional taxonomy mappings.
- Optionally, the adamantine disqualifier tree, by flattening WeakBins.

<u>ALIGN:</u>

For each query read q_j in Q:

1. Extract canonical k-mers and split each into (pfx, sfx).
    a. For each (pfx, sfx):
        i. Locate the suffix range corresponding to pfx.
        ii. Left-first binary-search for sfx within that range.
        iii. Enumerate all reference indices sharing that suffix.
        iv. Aggregate k-mer hit counts per reference for q_j.
2. If taxonomy output is requested, derive taxonomic labels for q_j directly from the set of possible reference hits using lowest-common-ancestor interpolation with a confidence threshold.
    a. If shallow LCA selected, use voting scheme among leaves
3. Assign q_j to exactly one reference using one of the following modes:
    a. BEST: select the reference with the highest k-mer count for q_j.
    b. CAPITALIST: defer assignment for global redistribution.

If coverage output is requested:

1. Tally matched k-mers per reference across all reads.
2. Compute total, unique, and Adamantine coverage statistics.
    a. If --forage is specified, consider all references with at least 1 k-mer match
    b. If --half-forage is specified, consider the set of k-mers mapping to the highest supported reference and all others within 50% of the k-mer support
    c. If neither is specified, consider the set of k-mers mapping to the highest supported reference only
3. If CAPITALIST redistribution is enabled:
    a. Iteratively reassign reads to references using a global expectation–maximization-like procedure until convergence.
    b. Output final one-to-one read-to-reference assignments.

Outputs:

- Reference assignments, optional taxonomy summaries, and optional coverage statistics.

Construction of XTree bacterial reference databases

We constructed XTree databases on the representative genomes provided by the Genome Taxonomy Database (GTDB) Release 214. We indexed these with XTree's BUILD function at k = 17, 21, 24, and 29.

Construction of and benchmarking on synthetic bacterial metagenomes

To evaluate the performance of XTree, we constructed synthetic metagenomes. We assessed XTree's performance on both high- and low-abundance genomes across multiple k-mer sizes and taxonomic groups. The workflow for synthetic metagenome construction is detailed in Supplementary Figure 1A.

We then generated four sets of 50 synthetic metagenomes, each comprising between 75 and 100 bacterial genomes sampled randomly from the GTDB. The four datasets created were at high abundance (0.1x coverage to 10x coverage) and from GTDB representative species, low abundance (0.001x coverage to 1x coverage) from GTDB representative species, high abundance from GTDB non-representative species, and low abundance from GTDB non-representative species. Within each of the 200 metagenomes, every genome was assigned an abundance (i.e., depth of coverage) following random sampling from a uniform distribution, with a minimal coverage threshold enforced to ensure at least one read per genome.

To generate synthetic Illumina sequencing reads for each metagenome, we used DWGSIM with an error rate of 0.001, a paired-end read length of 150 bp. Each synthetic metagenome was created by concatenating reads from all genomes in a given set. The synthetic reads were subsequently aligned to a corresponding XTree reference database. Alignments were run with forage mode (--doforage) and capitalist redistribution (default, Fig 1) enabled[1]. For this and all other synthetic metagenome evaluation tests, expected relative abundance was equal to the specified DWGSIM coverage for a given genome, normalized to sum to one. Observed relative abundance was the similarly normalized depth of coverage (pileup) output by XTree.

Evaluation of runtime and memory usage

We selected and downloaded fifteen samples at random from a major gut microbiome study with publicly available metagenomic data[23]. Tracking memory usage and runtime, we ran the sequencing reads through XTree and Kraken2 with the same parameters as used in the other benchmarking studies to the GTDB-indexed databases (again using the k = 29 database in the case of XTree).

Benchmarking on bacterial long read data

We benchmarked XTree on long read data across a variety of platforms. We first downloaded publicly available datasets used in prior benchmarking studies (https://github.com/LomanLab/mockcommunity as well as SRR11606871 and SRR13128014). The samples contained sequencing data on the (1) Zymo Research Mock Community Standard (comprising samples with both even and log distributed abundances), (2) the Zymo Research Mock Gut Community, and (3) the ATCC MSA-1003 Mock Community. These data were PacBio Hi-Fi sequencing as well as sequencing from Oxford Nanopore PromethION and GridION devices. The specific chemistries and flow cells are available on the above GitHub repository. We additionally were provided with PacBio Hi-Fi sequencing by Zymo Research on their even abundance Mock Community. After downloading, we quality controlled samples with Shi7 (parameters 500 8 FLOOR 4 ASS_QUALITY 10)[24]. We then ran XTree on this data with the default parameters and filtered for 10% unique coverage (again computed using the minimum of Adamantine and unique coverage).

Constructing the Pan Viral Compendium (PVC)

We downloaded assembled contigs from the databases listed in Supp Table 2 (most recent versions as of October 31st, 2023). The resultant 6,658,852 contigs were then combined and de-replicated (via greedy clustering of length-sorted contigs with AkronyMer) at a 99% identity threshold, yielding 2,851,990 non-redundant contigs. These were then processed with CheckV (default settings, V1.0.1) and separated into low, medium, high, and complete quality based on the outputs[25]. Pfam annotations were generated for each assembly with HMMER3 --notextw --noali --nobias --nseq_buffer 100000 --nhmm_buffer 1000, V3.3.2)[26]. Each contig was taxonomically annotated with GeNomad (default settings, V1.6.1)[18]. We additionally provide GenBank taxonomy for dereplicated contigs that either 1) were derived directly from the GenBank complete database (and therefore had an associated taxonomy) and/or 2) were representative contigs for a cluster of genomes containing GenBank assembly. We used the GenBank species-level annotation for the sensitivity analysis in the manuscript regarding the accuracy of high-resolution taxonomic classifications (Fig 2, Supp Fig 1, Supp Table 1). The sequences which comprise the Pan Viral Compendium (PVC), taxonomic annotations, Pfam annotations, gene catalog, and other associated metadata are available at https://figshare.com/articles/dataset/PVC_Release_V0_1/24566995?file=43154491.

PVC XTree database generation

For sets of dereplicated contigs stratified by quality (complete, high, medium, low), we generated and compared different XTree databases, varying the k-mer length used to index the genome sets, which we hypothesized would bias the dataset towards different clades based on genome features (e.g, length, repetitiveness), while suffering potential tradeoffs in alignment time (i.e., a smaller k-mer index yields a smaller database, meaning faster alignments).

Generation of and benchmarking on synthetic viral datasets

For viral benchmarking, we constructed synthetic metagenomes using the Pan Viral Compendium (PVC), a curated database of 6.6 million viral genomes. To systematically assess alignment performance across different genome qualities, we stratified the PVC into four groups: (1) Complete-High (CH) quality genomes, (2) Complete-High-Medium (CHM) quality genomes, (3) Complete-High-Medium-Low (CHML) quality genomes, and (4) Non-representative genomes (testing set). Each synthetic viral metagenome consisted of 100 to 250 randomly selected genomes per set, simulated at 1x–10x coverage using the same DWGSIM pipeline as bacterial metagenomes. Viral reads were aligned to XTree databases indexed at k = 17, 21, 24, and 29.

To evaluate classification performance, we compared XTree's predictions against known ground-truth taxonomic labels from the PVC. We computed precision, recall, and F1-score, analyzing classification accuracy across multiple taxonomic levels, including species, genus, family, order, and phylum.

Each synthetic dataset was aligned to its respective XTree database, and classification accuracy was assessed based on genome coverage thresholds. The benchmarking workflow involved five key steps: (1) random genome selection from the reference database, (2) coverage level assignment using a uniform sampling function, (3) read simulation with DWGSIM, (4) alignment with XTree using memory-mapped database indexing, and (5) performance evaluation using standard precision-recall metrics.

Generation and benchmarking of synthetic eukaryotic datasets

We downloaded the full set of complete and draft protozoan and fungal genomes from GenBank as of May and April 2024, respectively. We dereplicated each set at 95% identity and build XTree databases at compression level 2 with k=29. We created 10 synthetic metagenomes per dataset. We randomly selected between 10 and 25 genomes from each collection. We used DWGSIM with the same parameters as before to create synthetic reads for each genome at a random coverage level between 0.01 and 0.1, combining the outputs to create a metagenome. We aligned these reads back against the indexed references and estimated true and false positive rate 1% adamantine coverage.

Data Availability

All code used in this study can be accessed through the Two Frontiers Project XTree Github repository (https://github.com/two-frontiers-project/2FP-XTree). The Pan-Viral-Compendium and additional sequencing data that is not publicly available can be accessed at https://figshare.com/articles/dataset/PVC_Release_V0_1/24566995?file=43154491.

Competing Interests

GAG is an employee of Holobiome and consults for Seed Health. BF is an employee of Zymo Research, which had no role in funding this study other than providing sequencing data of one

sample. BTT and GMC have additional conflicts as founders and consultants at a variety of biotechnology companies, none of which were involved in this study.

Acknowledgements

We thank Zymo Research for providing PacBio Hi-Fi sequencing of their microbial standard. We thank Heng Li and Jim Shaw for their feedback on an earlier version of this manuscript.

References

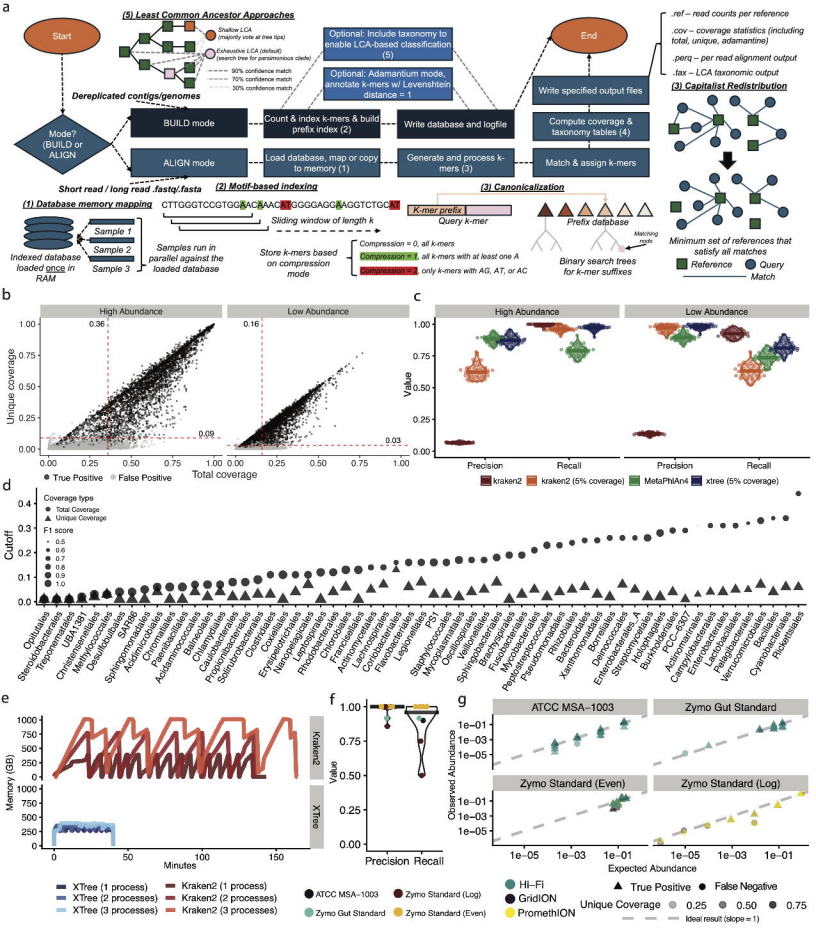
1. Al-Ghalith, G. & Knights, D. BURST enables mathematically optimal short-read alignment for big data. *Bioinformatics* (2020).
2. Priya, S. *et al.* Identification of shared and disease-specific host gene-microbiome associations across human diseases using multi-omic integration. *Nat. Microbiol.* **7**, 780–795 (2022).
3. Tierney, B. T. *et al.* Longitudinal multi-omics analysis of host microbiome architecture and immune responses during short-term spaceflight. *Nat. Microbiol.* **9**, 1661–1675 (2024).
4. Button, J. E. *et al.* Precision modulation of dysbiotic adult microbiomes with a human-milk-derived synbiotic reshapes gut microbial composition and metabolites. *Cell Host Microbe* **31**, 1523–1538.e10 (2023).
5. Tolonen, A. C. *et al.* Synthetic glycans control gut microbiome structure and mitigate colitis in mice. *Nat. Commun.* **13**, 1244 (2022).
6. Lin, C.-Y. *et al.* Longitudinal fecal microbiome and metabolite data demonstrate rapid shifts and subsequent stabilization after an abrupt dietary change in healthy adult dogs. *Anim. Microbiome* **4**, 46 (2022).
7. Johnson, K. E. *et al.* Human cytomegalovirus in breast milk is associated with milk composition and the infant gut microbiome and growth. *Nat. Commun.* **15**, 6216 (2024).
8. Vinithakumari, A. A. *et al.* Clostridioides difficile infection dysregulates brain dopamine metabolism. *Microbiol. Spectr.* **10**, e0007322 (2022).
9. Tierney, B. T. *et al.* Towards geospatially-resolved public-health surveillance via

wastewater sequencing. *Nat. Commun.* **15**, 8386 (2024).

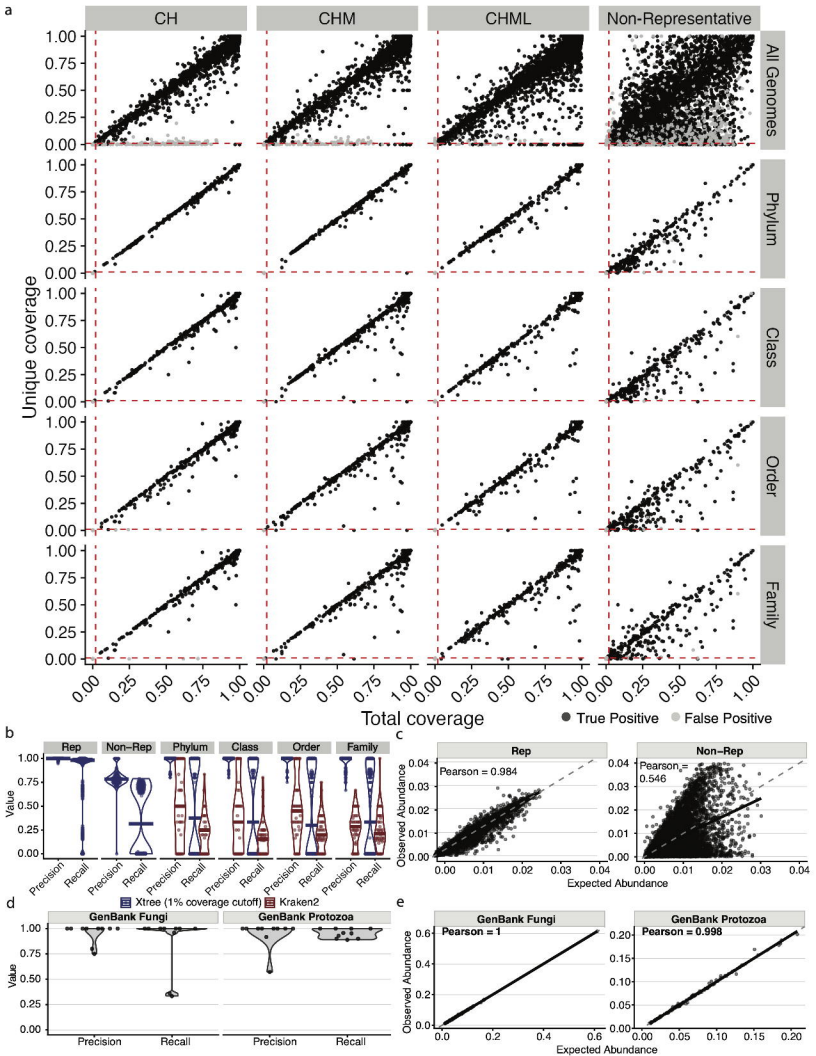
10. Napier, B. A., Van Den Elzen, C., Al-Ghalith, G. A. & Avena, C. V. Mo1894 A MULTI-SPECIES SYNBIOTIC (DS-01) ALLEVIATES CONSTIPATION AND ABDOMINAL PAIN IN IRRITABLE BOWEL SYNDROME SUBTYPE …. *Gastroenterology* (2024).
11. Napier, B. A. *et al.* Mo1894 A MULTI-SPECIES SYNBIOTIC (DS-01) ALLEVIATES CONSTIPATION AND ABDOMINAL PAIN IN IRRITABLE BOWEL SYNDROME SUBTYPE MIXED (IBS-M) SUBJECTS WHILE BOOSTING SYNBIOTIC SPECIES ASSOCIATED WITH DECREASED SYSTEMIC INFLAMMATION AND NET FORMATION. *Gastroenterology* **166**, S–1164 (2024).
12. Wood, D. E., Lu, J. & Langmead, B. Improved metagenomic analysis with Kraken 2. *Genome Biol.* **20**, 257 (2019).
13. Irber, L. *et al.* sourmash v4: A multitool to quickly search, compare, and analyze genomic and metagenomic data sets. *J. Open Source Softw.* **9**, 6830 (2024).
14. Blanco-Míguez, A. *et al.* Extending and improving metagenomic taxonomic profiling with uncharacterized species using MetaPhlAn 4. *Nat. Biotechnol.* **41**, 1633–1644 (2023).
15. Lu, J., Breitwieser, F. P., Thielen, P. & Salzberg, S. L. Bracken: estimating species abundance in metagenomics data. *PeerJ Comput. Sci.* **3**, e104 (2017).
16. Marić, J., Križanović, K., Riondet, S., Nagarajan, N. & Šikić, M. Comparative analysis of metagenomic classifiers for long-read sequencing datasets. *BMC Bioinformatics* **25**, 15 (2024).
17. Helal, A. A., Saad, B. T., Saad, M. T., Mosaad, G. S. & Aboshanab, K. M. Benchmarking long-read aligners and SV callers for structural variation detection in Oxford nanopore sequencing data. *Sci. Rep.* **14**, 6160 (2024).
18. Camargo, A. P. *et al.* Identification of mobile genetic elements with geNomad. *Nat. Biotechnol.* **42**, 1303–1312 (2024).
19. Punta, M. *et al.* The Pfam protein families database. *Nucleic Acids Res.* **40**, D290–301

(2012).

20. Kanehisa, M., Furumichi, M., Sato, Y., Kawashima, M. & Ishiguro-Watanabe, M. KEGG for taxonomy-based analysis of pathways and genomes. *Nucleic Acids Res.* **51**, D587–D592 (2023).
21. Federhen, S. The NCBI taxonomy database. *Nucleic Acids Res.* **40**, D136–43 (2012).
22. Farach-Colton, M. Lowest common ancestors in trees. in *Encyclopedia of Algorithms* 1169–1174 (Springer New York, New York, NY, 2016).
23. Lloyd-Price, J. *et al.* Multi-omics of the gut microbial ecosystem in inflammatory bowel diseases. *Nature* **569**, 655–662 (2019).
24. Al-Ghalith, G. A., Hillmann, B., Ang, K., Shields-Cutler, R. & Knights, D. SHI7 is a self-learning pipeline for multipurpose short-read DNA quality control. *mSystems* **3**, 10.1128/msystems. 00202–17 (2018).
25. Nayfach, S. *et al.* CheckV assesses the quality and completeness of metagenome-assembled viral genomes. *Nat. Biotechnol.* **39**, 578–585 (2021).
26. Eddy, S. R. A new generation of homology search tools based on probabilistic inference. *Genome Inform.* **23**, 205–211 (2009).



a
Start
(5) Least Common Ancestor Approaches
Shallow LCA (majority vote at tree tips)
Exhaustive LCA (default) (search tree for parsimonious clade)
90% confidence match
70% confidence match
30% confidence match
Dereplicated contigs/genomes
Mode? (BUILD or ALIGN
BUILD mode
ALIGN mode
Short read / long read .fastq/.fasta
Optional: Include taxonomy to enable LCA-based classification (5)
Optional: Adamantium mode, annotate k-mers w/ Levenshtein distance = 1
Count & index k-mers & build prefix index (2)
Load database, map or copy to memory (1)
Write database and logfile
Generate and process k-mers (3)
End
Write specified output files
Compute coverage & taxonomy tables (4)
Match & assign k-mers
.ref – read counts per reference
.cov – coverage statistics (including total, unique, adamantine)
.perq – per read alignment output
.tax – LCA taxonomic output
(3) Capitalist Redistribution
Minimum set of references that satisfy all matches
Reference
Query
Match
(1) Database memory mapping
Indexed database loaded once in RAM
Sample 1
Sample 2
Sample 3
Samples run in parallel against the loaded database
(2) Motif-based indexing
CTTGGGTCCGTGGTAACAAACATGGGGAGGAAGGTCTGCAT
Sliding window of length k
Store k-mers based on compression mode
Compression = 0, all k-mers
Compression = 1, all k-mers with at least one A
Compression = 2, only k-mers with AG, AT, or AC
(3) Canonicalization
K-mer prefix
Query k-mer
Prefix database
Matching node
Binary search trees for k-mer suffixes
b
High Abundance
Low Abundance
Unique coverage
Total coverage
0.36
0.09
0.16
0.03
True Positive
False Positive
c
High Abundance
Low Abundance
Value
Precision
Recall
kraken2
kraken2 (5% coverage)
MetaPhlAn4
xtree (5% coverage)
d
Cutoff
Coverage type
Total Coverage
Unique Coverage
F1 score
0.5
0.6
0.7
0.8
0.9
1.0
Opitutales
Steroidobacterales
Treponematales
UBA1381
Christensenellales
Methylococcales
Desulfobulbales
SAR86
Sphingomonadales
Acidimicrobiales
Chromatiales
Paenibacillales
Acidaminococcales
Balneolales
Chlamydiales
Caulobacterales
Propionibacterales
Solirubrobacterales
Cuthridiales
Coxiellales
Erysipelotrichales
Nanopelagicales
Leptospirales
Rhodobacterales
Chlorobiales
Francisellales
Actinomycetales
Lachnospirales
Coriobacteriales
Flavobacteriales
Legionellales
PS1
Staphylococcales
Mycoplasmatales
Oscillospirales
Vellonellales
Sphingobacteriales
Brachyspirales
Fusobacteriales
Mycobacteriales
Peptostreptococcales
Pseudomonadales
Rhizobiales
Bacteroidales
Xanthomonadales
Borrelales
Deinococcales
Enterobacteriales_A
Streptomycetales
Holophagales
Burkholderiales
PCC-6307
Actinomarinales
Campylobacterales
Enterobacteriales
Lactobacillales
Pelagibacterales
Verrucomicrobiales
Bacillales
Cyanobacteriales
Rickettsiales
e
Memory (GB)
Minutes
Kraken2
XTree
XTree (1 process)
XTree (2 processes)
XTree (3 processes)
Kraken2 (1 process)
Kraken2 (2 processes)
Kraken2 (3 processes)
f
Value
Precision
Recall
ATCC MSA–1003
Zymo Gut Standard
Zymo Standard (Log)
Zymo Standard (Even)
g
ATCC MSA–1003
Zymo Gut Standard
Zymo Standard (Even)
Zymo Standard (Log)
Observed Abundance
Expected Abundance
Hi–Fi
GridION
PromethION
True Positive
False Negative
Unique Coverage
0.25
0.50
0.75
Ideal result (slope = 1)


a
CH
CHM
CHML
Non-Representative
All Genomes
Phylum
Class
Order
Family
Unique coverage
Total coverage
True Positive
False Positive
b
Rep
Non−Rep
Phylum
Class
Order
Family
Value
Precision
Recall
Xtree (1% coverage cutoff)
Kraken2
c
Rep
Non−Rep
Pearson = 0.984
Pearson = 0.546
Observed Abundance
Expected Abundance
d
GenBank Fungi
GenBank Protozoa
Value
Precision
Recall
e
GenBank Fungi
GenBank Protozoa
Pearson = 1
Pearson = 0.998
Observed Abundance
Expected Abundance