Supplementary Method S4: Metagenomic binning

Binning was applied independently to all three assembly targets described in Supplementary Method S2: (1) per-sample hybrid SPAdes assemblies (56 samples), (2) the filtered Round 1 MEGAHIT co-assembly, and (3) the Round 2 rare-contig co-assembly. For each target, three binning tools were run in parallel — MetaBAT2, SemiBin2, and VAMB — and their outputs were integrated using DAS_Tool to produce a non-redundant consensus bin set per assembly target. A post-processing script then renamed each DAS_Tool-selected bin with a tool-of-origin prefix to preserve provenance through dereplication.

---

1. Coverage depth calculation

MetaBAT2 and VAMB (Round 1) require per-contig depth tables computed across all samples simultaneously. These were generated using `jgi_summarize_bam_contig_depths` (from the MetaBAT2 package), which takes all coordinate-sorted BAM files and outputs a tab-separated depth matrix. Depth files were computed as part of the read-mapping workflow (see Supplementary Method S3):

Assembly target	Depth file	BAM input directory
Round 1 filtered co-assembly	`filt.asm.depth.txt`	`bam_output_Illumina_filt_coass/`
Round 2 rare co-assembly	`rare.asm.depth.txt`	`bam_output_Illumina_rare_coass/`
Per-sample hybrid SPAdes	per-sample `depth.txt`	`Illumina_hybrid_bam_output/`, `bam_output_nanopore/`

For per-sample hybrid assemblies, `jgi_summarize_bam_contig_depths` was given both the Illumina and Nanopore BAMs for that sample simultaneously, so that long-read coverage also informed the depth profile.

---

2. MetaBAT2

MetaBAT2 (Kang et al., 2019) was run on all three assembly targets using the per-contig depth tables computed in Section 1 above.

Round 1 co-assembly (28 CPUs, 1640 GB, fat_q; script: `metabat2_coassembly.sh`): `jgi_summarize_bam_contig_depths` was run across all 122 Illumina BAMs to produce `filt.asm.depth.txt`, and MetaBAT2 was then run with default parameters on the filtered assembly, with bins written to `filtered_co_assembly_metabat_bins/`.

Round 2 rare co-assembly (28 CPUs, 840 GB, fat_q; script: `0_M2_06_metabat_rare_coass.sh`): MetaBAT2 was run using the pre-computed `rare.asm.depth.txt` depth file, with `--minContig 1500` and `--unbinned` (to retain a file of unbinned contigs for diagnostic purposes). Bins were written to `rare_co_assembly_metabat_bins/`.

Per-sample hybrid assemblies (SLURM array, 8 CPUs, 48 GB per sample; script: `metabat_binning_hybrid_slurm.sh`): For each sample, depth was computed from both the Illumina and Nanopore BAMs for that sample, and MetaBAT2 was run with default parameters on that sample's hybrid SPAdes contigs. Bins were written to `metabat_hybrid_output2/${SAMPLE}/`.

---

3. SemiBin2

SemiBin2 (Pan et al., 2023) uses a self-supervised neural network trained on coverage and sequence composition to cluster contigs into bins. Because SemiBin2's `multi_easy_bin` mode requires the assembly FASTA to carry a sample identifier in the contig headers (in `SAMPLE:contig` format), co-assemblies were reformatted with a `C:` prefix on all contig names prior to mapping (see Supplementary Method S3, sections 2b and 3).

Round 1 co-assembly (24 CPUs, 500 GB, fat_q; script: `03_semibin2_with_prefixes.slurm.sh`): `SemiBin2 multi_easy_bin` was run with the prefixed co-assembly (`pref_coassembly/assembly.prefixed.fa`) and all bowtie2-aligned per-sample BAMs from `mapped_prefixed/`. Minimum contig length was set to 2,500 bp (`-m 2500`). No habitat model was specified; SemiBin2 used its generic global model. Output was written to `semibin2_out_coassembly/semibin2_run/`.

Round 2 rare co-assembly (24 CPUs, 900 GB, fat_q; script: `0_M2_10_semibin2_rare_coassembly.sh`): `SemiBin2 multi_easy_bin` was run on the prefixed Round 2 assembly (`rare_pref_coassembly/rare.assembly.prefixed.f`) and all bowtie2-aligned BAMs from `mapped_rare_prefixed/`, with `-m 2500` and no habitat model. Output was written to `semibin2_out_rare_coassembly/semibin2_run/`.

Per-sample hybrid assemblies (SLURM array, 24 CPUs, 900 GB per sample; script: `semibin2_hybrid_single.slurm.sh`) For each sample, `SemiBin2 single_easy_bin` was run. The Illumina and Nanopore BAMs were merged with `samtools merge` before being passed as a single BAM. Contigs shorter than 2,500 bp were pre-filtered from the assembly with `seqkit seq -m 2500` to reduce memory requirements. The `--sequencing-type long_read` flag was set to select the appropriate embedding strategy for long-read and hybrid data. Output bins were written to `semibin2_out_hybrid_single_v4_last3/${SAMPLE}/semibin2_run_min2500/`.

---

4. VAMB

VAMB v5 (Sieber et al., 2018; Nissen et al., 2021) is a variational autoencoder that encodes both sequence composition and abundance co-variation across samples to cluster contigs. The `bin default` subcommand was used throughout.

Round 1 co-assembly (32 CPUs, 900 GB, fat_q; script: `vamb_coassembly.slurm.sh`): Coverage was provided directly via `--bamdir bam_output_Illumina_filt_coass`, letting VAMB compute its own abundance matrix. Only bins with a minimum total size of 100 kbp were written (`--minfasta 100000`). Bins were written to `vamb_coassembly_out2/`.

Round 2 rare co-assembly (32 CPUs, 900 GB, fat_q; script: `0_M2_07_vamb.rare.sh`): To avoid recomputing coverage that had already been computed for MetaBAT2, the existing `rare.asm.depth.txt` depth file was converted to VAMB's TSV abundance format via an inline Python script (stripping variance columns and converting contig depths to the `contigname + per-sample depth` format expected by `--abundance_tsv`). VAMB was then run with `-m 1000` and `--minfasta 100000`. Bins were written to `vamb_coassembly_out_rare/bins/`.

Per-sample hybrid assemblies (SLURM array, 16 CPUs, 280 GB per sample; script: `vamb_hybrid_assemblies.sh`): For each sample, Illumina and Nanopore BAMs were merged with `samtools merge`, and VAMB was run using `--bamdir` pointing to a directory containing the single merged BAM. The `-o` flag (disabling binsplitting) was set because hybrid SPAdes contig headers contain characters that would otherwise trigger spurious splits. Parameters: `-m 1000`, `--minfasta 100000`, `--seed 42`. Bins were written to `vamb_single_hybrid_out/${SAMPLE}/`.

---

5. DAS_Tool bin integration

DAS_Tool (Sieber et al., 2018) was used to select a non-redundant, high-quality consensus bin set from the three independent binners. It takes contig-to-bin membership tables for each binner, evaluates per-bin single-copy marker gene completeness and contamination, and selects the best-scoring non-overlapping bin from each cluster.

Before calling DAS_Tool, a custom `mk_c2b()` shell function (embedded in each DAS_Tool script) parsed each binner's output directory to generate the required contig-to-bin TSV tables. For SemiBin2 bins — whose contig headers carried the `C:` prefix added during assembly reformatting — the function stripped this

prefix to match the original assembly contig identifiers, which DAS_Tool uses for deduplication. This prefix stripping was done with an awk substitution (`sub(/^[^:]*:/,"",id)`) applied to each FASTA header.

Three DAS_Tool runs were performed, one per assembly target:

Assembly target	Script	Input bins	Output directory
Round 1 co-assembly	`10_dastool_coass_V2.slurm.sh`	`filtered_co_assembly_metabat_bins/`, `vamb_coassembly_out2/bins`, `semibin2_out_coassembly/semibin2_run/bins`	`dastool_out_coassembly_v2/`
Round 2 rare co-assembly	`0_M2_11_Dastool_rare.sh`	`rare_co_assembly_metabat_bins/`, `vamb_coassembly_out_rare/bins/bins`, `semibin2_out_rare_coassembly/semibin2_run/bins`	`dastool_out_rare_coassembly_v2/`
Per-sample hybrid SPAdes	`dastool_single_hybrid.sbatch.sh`	per-sample MetaBAT2, SemiBin2, VAMB bin directories	`dastool_out_single_hybrid_prefixed/`

All three runs used `--write_bins` to output the winning bin FASTA files alongside the scoring summary. Default DAS_Tool scoring and completeness thresholds were used throughout. Runs used 24 CPUs (co-assemblies) or 16 CPUs (per-sample).

---

6. Bin renaming and provenance

Because DAS_Tool inherits bin filenames directly from the winning binner, bins from different tools have different naming conventions (MetaBAT2: `bin.NNN`; SemiBin2: `C_SemiBin_NNN`; VAMB: `NNN.fna`). To allow downstream dereplication and quality filtering to trace each bin back to its binning tool, all bins were renamed with an explicit tool-of-origin prefix using the script `dastool_harvest_v2.sh`.

The script matched each filename against regular expression patterns and renamed bins as follows:

Original pattern	Renamed to	Tool
`bin.NNN`	`metabat2_NNN.fa`	MetaBAT2
`C_SemiBin_NNN`	`semibin2_NNN.fa`	SemiBin2
`NNN.fna`	`vamb_NNN.fa`	VAMB

Renamed bins were copied to `drep_ready_v2/`, and a provenance table (`drep_ready_v2/mapping.tsv`) was written recording the original filename, renamed filename, inferred tool, and source path for each bin. Bins whose filenames matched the MetaBAT2 unbinned/lowDepth patterns (`bin.unbinned`, `bin.lowDepth`) were excluded from the renamed output.

References

- Kang, D. D. et al. (2019). MetaBAT 2: an adaptive binning algorithm for robust and efficient genome reconstruction from metagenome assemblies. *PeerJ*, 7, e7359. https://doi.org/10.7717/peerj.7359
- Pan, S. et al. (2023). A deep siamese neural network improves metagenome-assembled genomes in microbiome datasets across different environments. *Nature Communications*, 13, 2326. https://doi.org/10.1038/s41467-022-29843-y
- Nissen, J. N. et al. (2021). Improved metagenome binning and assembly using deep variational autoencoders. *Nature Biotechnology*, 39, 555–560. https://doi.org/10.1038/s41587-020-00777-4
- Sieber, C. M. K. et al. (2018). Recovery of genomes from metagenomes via a dereplication, aggregation and scoring strategy. *Nature Microbiology*, 3, 836–843. https://doi.org/10.1038/s41564-018-0171-1