nature machine intelligence

Supplementary information

https://doi.org/10.1038/s42256-024-00908-5

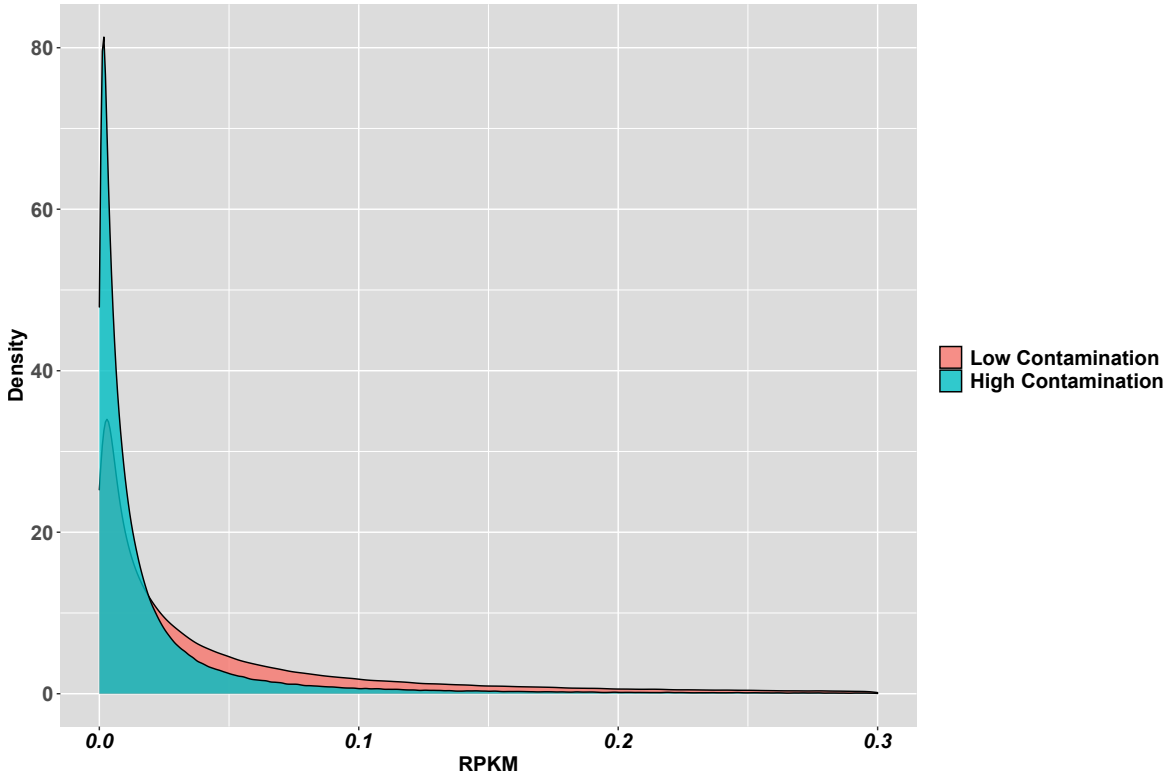
A multi-modal deep language model for contaminant removal from metagenome-assembled genomes

In the format provided by the authors and unedited

Supplementary Notes

1 Analysis of MAG abundances with both high and low levels of contamination

We randomly selected MAGs with high levels of contamination (completeness $\geq$ 50% and contamination > 10%) and randomly selected 1,000 MAGs with low levels of contamination (completeness $\geq$ 50% and contamination $\leq$ 10%) from the human fecal dataset that were binned with MetaBAT2. MAG abundance estimation was conducted by mapping reads to MAG contigs. The reads per kilobase per million mapped read (RPKM) for these MAGs were calculated.



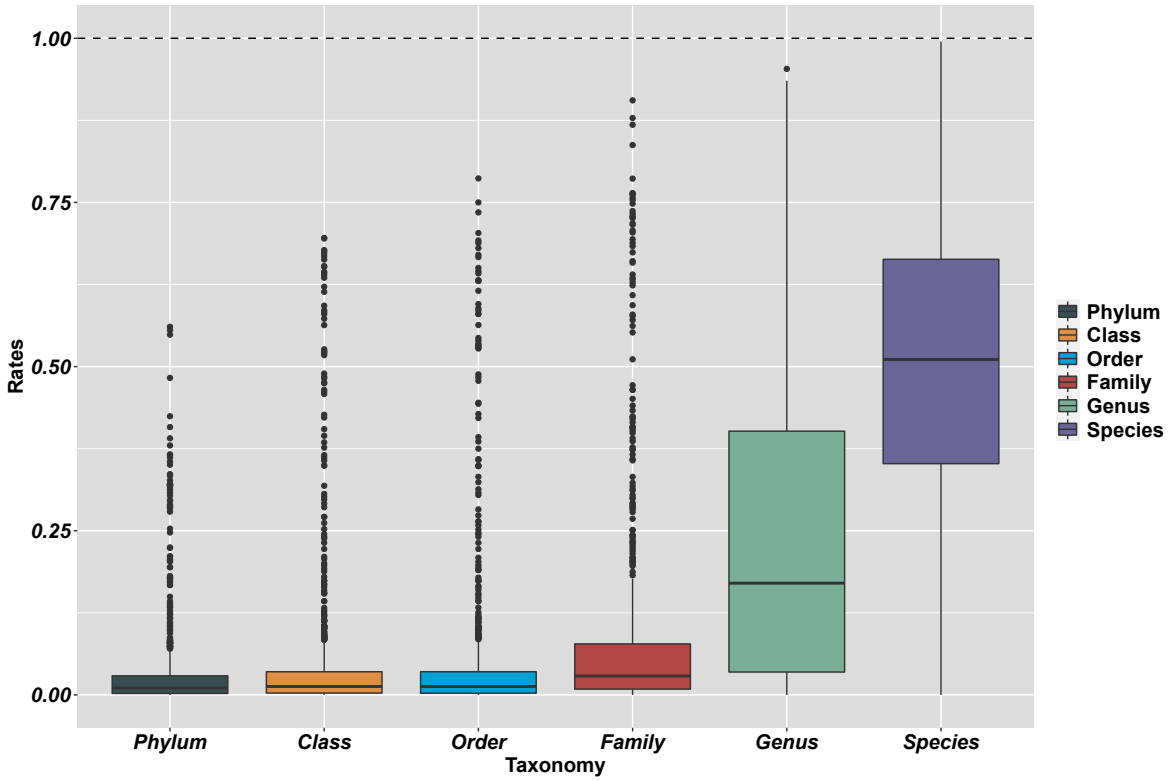
Supplementary Figure 1: Density distribution of RPKM for MAGs with high and low contamination

The density distribution of RPKM for MAGs with high and low contamination exhibited significant overlap, as depicted in **Supplementary Figure** 1. This suggests that some MAGs with high abundances might not be suitable for further analysis due to contamination, potentially resulting in the exclusion of MAGs that have significant associations with some diseases.

2 Analysis of the annotation of contigs in highly contaminated MAGs

We randomly selected 1,110 highly contaminated MAGs (completeness $\geq$ 50% and contamination $>$ 10%) from the human fecal dataset, which were assembled and binned using metaSPAdes and MetaBAT2. We utilized Kraken2 [1] with a standard database to annotate the contigs in these MAGs.

We identified the taxon with the highest number of contigs as the core taxon, while the remaining taxa were considered contaminated at each taxonomic rank (from phylum to species). The contamination rate at each taxonomic rank was assessed by calculating the rate of contigs belonging to contaminated taxa to the total number of annotated contigs.



Supplementary Figure 2: Estimated contamination rates in six taxonomic ranks (n = 1,100). All the box plots depict the median (horizontal line inside box), 25th and 75th percentiles (box), and 25th or 75th percentiles $\pm$ 1.5 $\times$ interquartile range (whiskers). The other points are outliers.

Supplementary Figure 2 clarifies that contamination mostly did not occur at higher taxonomic ranks such as phylum, class, and order but became more concentrated at lower taxonomic ranks, including family, genus, and species.

3 Averaged precision and recall for MAGpurify, MDMcleaner, and Deepurify on SIM_1

We assessed the precision and recall for three MAG decontamination methods MAGpurify, MDMcleaner, and Deepurify, using the SIM_1 simulated testing set. Contaminated contigs were classified as positive labels and core contigs as negative labels. We calculated balanced precision and recall to account for the label imbalance in the simulated MAG, where core and contaminated contigs had unequal counts; this involved assigning higher weights to contaminated contigs, adjusted based on the rate between the number of core contigs and contaminated contigs within a simulated MAG. **Supplementary Table** 1 summarizes the precision and recall values for each MAG decontamination method.

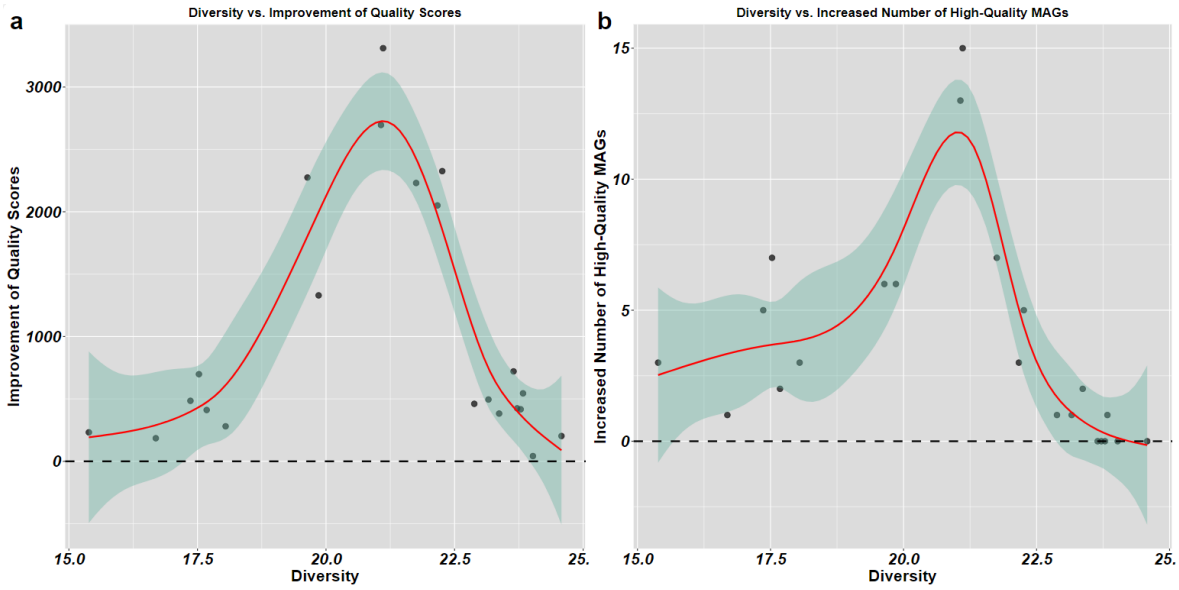
	Contamination Rate: 5%						Contamination Rate: 10%					
	MAGpurify		MDMcleaner		Deepurify		MAGpurify		MDMcleaner		Deepurify	
$LCA(c,t)$	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
Kingdom	0.921	0.592	0.991	0.988	0.996	0.996	0.980	0.577	0.998	0.993	0.997	0.997
Phylum	0.877	0.312	0.997	0.992	0.990	0.989	0.961	0.344	0.997	0.984	0.991	0.991
Class	0.895	0.256	0.998	0.936	0.981	0.980	0.890	0.271	0.997	0.961	0.978	0.977
Order	0.891	0.158	0.978	0.700	0.940	0.931	0.961	0.220	0.998	0.710	0.971	0.967
Family	0.865	0.094	0.918	0.480	0.953	0.946	0.916	0.154	0.853	0.293	0.931	0.913
Genus	0.781	0.113	0.875	0.216	0.897	0.871	0.871	0.082	0.879	0.174	0.897	0.869
	Contamination Rate: 15%						Contamination Rate: 20%					
	MAGpurify		MDMcleaner		Deepurify		MAGpurify		MDMcleaner		Deepurify	
$LCA(c,t)$	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
Kingdom	0.976	0.424	0.999	0.995	0.996	0.996	0.958	0.385	0.996	0.996	0.994	0.994
Phylum	0.952	0.221	0.992	0.974	0.990	0.989	0.906	0.294	0.991	0.987	0.990	0.990
Class	0.881	0.226	0.998	0.952	0.973	0.970	0.955	0.206	0.998	0.960	0.961	0.958
Order	0.926	0.138	0.976	0.779	0.962	0.958	0.947	0.186	0.977	0.792	0.965	0.961
Family	0.879	0.163	0.839	0.328	0.928	0.922	0.960	0.107	0.879	0.482	0.929	0.917
Genus	0.851	0.050	0.773	0.118	0.866	0.825	0.907	0.083	0.880	0.138	0.852	0.822

Supplementary Table 1: Averaged precision and recall for MAGpurify, MDMcleaner, and Deepurify on SIM_1 simulated testing set.

The table shows that Deepurify consistently achieves significantly higher precision and recall than MAGpurify and MDMcleaner, especially when contamination occurs at the family, genus, and species taxonomic ranks. These results highlight Deepurify's effectiveness in removing a substantial portion of contaminated contigs while preserving clean contigs.

4 Deepurify's performance and community complexity

We used Nonpareil to quantify the community complexity of the real-word metagenomic sequencing datasets, including 7 soil samples, 3 plant samples, 3 freshwater samples, 3 ocean samples (1021520, 1021523, 1021526), six human fecal samples (C0001, C0004, HC0006, HK0023, K0031, K0032). We plotted the community complexity (diversity) of each sample against the improvement of quality scores and the increased number of high-quality MAGs using Deepurify_Iter in **Supplementary Figure** 3.



Supplementary Figure 3: (**a**). Diversity of each sample and improvement of MAG quality scores by Deepurify_Iter. The red curve was generated using the generalized additive model (GAM) on the improvement of quality scores for MAGs and diversity of samples. (**b**). Diversity of each sample and the increased number of high-quality MAGs by Deepurify_Iter. The red curve was generated using the GAM on the increased number of high-quality MAGs and diversity of samples. The error bands represent the 95% confidence interval for the GAM curve.

5 Sequence augmentation

We proposed several simple data augmentation strategies for the target sequences.

- Insertions: We randomly generated a nucleotide sequence by concatenating A, T, C, and G with equal probabilities. The length of this generated sequence was in the range of 20% - 40% of the target sequence's length, following a uniform distribution. The insertion of the generated sequence occurred as follows: it was either not inserted or inserted at either the beginning or end of the target sequence with corresponding probabilities of 0.5, 0.25, and 0.25, respectively.
- Mutations: Each base pair in the target sequence had a 0.01 probability of being replaced by one of the other three nucleotides, with equal probabilities for each nucleotide.

6 Advantage of GTDB's taxonomic lineage for training Deepurify

We trained Deepurify on representative microbial genomes using both NCBI and GTDB taxonomic lineages. Our findings suggested that utilizing GTDB's taxonomic lineage could significantly improve Deepurify's performance on simulated data (SIM_1, Wilcoxon Signed-Rank Sum test, two-sided, p-value = 1.192e-07, 95 percent confidence interval = [0.023, 0.041], effect size statistic = 0.303). We further designed a strategy to improve the performance of Deepurify by selecting the sequences with low similarity from the representative genomes. It included two steps: 1. split the representative

genomes into sequences of length 8,192 bps with 512 bps overlap; 2. Apply MMSeqs2 to group these sequences and only select the representative sequences from each cluster (SIM_1, Wilcoxon Signed-Rank Sum test, two-sided, p-value = 1.192e-07, 95 percent confidence interval = [0.089, 0.144], effect size statistic = 1.29). **Supplementary Table 2** presents the averaged balanced macro F1-scores for Deepurify trained under different conditions: with NCBI's taxonomic lineages, GTDB's taxonomic lineages, and GTDB's lineages using MMseqs2 clustering.

	Contamination Rate: 5%			Contamination Rate: 10%		
$LCA(c,t)$	NCBI	GTDB	GTDB & MMseqs2	NCBI	GTDB	GTDB & MMseqs2
Kingdom	0.969	0.970	0.996	0.976	0.983	0.997
Phylum	0.927	0.944	0.991	0.929	0.944	0.989
Class	0.880	0.909	0.980	0.858	0.911	0.977
Order	0.806	0.834	0.930	0.832	0.880	0.967
Family	0.750	0.820	0.945	0.740	0.798	0.910
Genus	0.683	0.722	0.865	0.657	0.725	0.863
	Contamination Rate: 15%			Contamination Rate: 20%		
$LCA(c,t)$	NCBI	GTDB	GTDB & MMseqs2	NCBI	GTDB	GTDB & MMseqs2
Kingdom	0.971	0.981	0.996	0.972	0.972	0.994
Phylum	0.929	0.944	0.989	0.928	0.948	0.991
Class	0.862	0.880	0.970	0.867	0.889	0.957
Order	0.812	0.865	0.957	0.842	0.872	0.960
Family	0.757	0.803	0.917	0.726	0.779	0.915
Genus	0.625	0.661	0.813	0.627	0.647	0.814

Supplementary Table 2: Averaged balanced macro F1-score across various contamination rates, along with the LCA of core and contaminated genomes at different taxonomic ranks on SIM1. The training sets were constructed based on NCBI's taxonomy, GTDB's taxonomy, and GTDB's taxonomy followed by MMseqs2 clustering.

7 Sequence embedding method

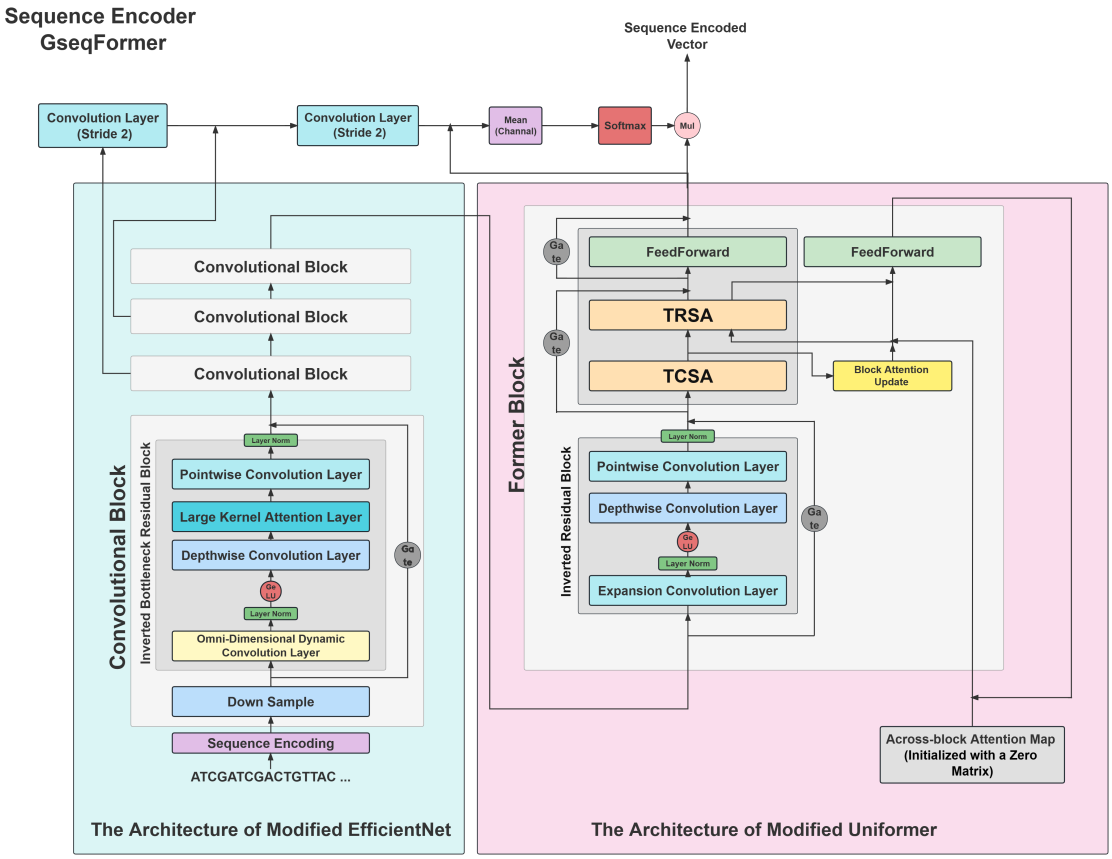
We performed sequence embedding using both one-hot embedding and k-mer embeddings and then merged them to form a unified matrix. In the one-hot embedding procedure, we included two special tokens: "X" to indicate padding applied at the sequence's end to ensure consistent input length within a mini-batch, and "N" to represent any unidentified characters. Consequently, the one-hot embedding generated a matrix sized $L \times 6$ for each sequence, with L representing the sequence length.

We have integrated k-mer embedding into our sequence embedding approach. First, we created a look-up table where each row represents a unique k-mer token and its corresponding ID. This table included two special tokens: the [PAD] token for padding sequences of varying lengths and [UNK] to signify any unidentified k-mer tokens within the sequences. Next, we split the sequence into k-mer tokens and converted them to IDs using the look-up table. Finally, a trainable matrix within the

embedding layer mapped the k-mer token ID to a dense vector with $d_m(k\text{-}mer)$ dimensions. In this study, we used both 3-mer and 4-mer token embeddings to represent the input sequences, with the dimensions set at $d_m(3\text{-}mer) = 16$ and $d_m(4\text{-}mer) = 32$, respectively.

We combined the one-hot, 3-mer, and 4-mer embeddings of the sequence with those of its reverse complement using the same embedding methods. This fusion resulted in a unified matrix that effectively represents the input sequence. The unified matrix has a fixed dimension of $L \times 108$, with L representing the sequence length. This approach provides a comprehensive and enriched representation of the input sequences, enhancing the robustness and effectiveness of our model.

8 Architecture of modified Uniformer



Supplementary Figure 4: The GseqFormer architecture comprises two essential components. The segment shown on the left utilizes a modified EfficientNet architecture to compress sequence length, while that on the right uses a modified UniFormer architecture with four distinct attention mechanisms designed to capture intricate taxonomic patterns within complex DNA sequences.

We replaced the dynamic position embedding (DPE) blocks in UniFormer with the inverted residual blocks (IRB) designed from MobileNet V2 [2]. The DPE layer has a limited number of parameters and is less effective at handling complex sequences on its own. Therefore, we substituted it with the IRB, which has more trainable parameters to enhance the model's capability to process complex DNA

sequences.

The multi-head relation aggregator blocks in UniFormer have been replaced with tensor column-wise and row-wise gated self-attention modules (TCSA, **Supplementary Figure** 5 and TRSA, **Supplementary Figure** 6), modified from the Evoformer architecture. TRSA and TCSA are designed to improve the model's capacity to capture taxonomic patterns within complex DNA sequences by integrating four distinct attention mechanisms for their input tensor (ψ), which represents the sequence embedding. The four attention mechanisms are as follows: (1) **Embedding attention**: This mechanism evaluates the attention scores assigned to different embedding approaches for each nucleotide in the sequence. (2) **Nucleotide attention**: This mechanism captures interactions between nucleotides in a sequence, similar to the self-attention mechanism in the Transformer model. (3) **Across-block attention**: This mechanism links nucleotide interaction knowledge learned from different Former blocks, enabling the model to capture nucleotide interrelationships across these blocks. (4) **Spatial attention**: This mechanism extracts local spatial contexts from the attention map generated by nucleotide attention.

Both TCSA and TRSA have an input sequence embedding tensor $\psi \in \mathbf{R}^{L\times C}$ with two dimensions: sequence length (L), and channel (C). Initially, we partitioned the channel dimension C into three equal segments in these two modules, resulting in a three-dimensional tensor denoted as ψ_d, with dimensions $[3, L, d_k]$, where d_k is the value obtained by dividing C by 3. These dimensions correspond to the number of embedding methods, the sequence length, and the channel dimension for the corresponding embedding method. This reshaped configuration allows ψ to have an extra dimension to simultaneously represent three distinct embedding methods (one-hot, 3-mer, and 4-mer; refer to **Supplementary Note** 7) for a given sequence enabling the application of embedding attention to this dimension.

In the TCSA module, embedding attention was applied to ψ_d, focusing specifically on its first dimension, which represents the number of embedding approaches for a sequence. We denoted the dense vector at the l-th token of ψ_d as $\upsilon_e \in \mathbf{R}^{3\times d_k}$. The embedding attention map $\rho_e \in \mathbf{R}^{3\times 3}$ can be calculated as follows:

$$\rho_e = \frac{Q_e K_e^T}{\sqrt{d_k}} \quad where \quad Q_e = \upsilon_e W_e^Q, K_e = \upsilon_e W_e^K \tag{1}$$

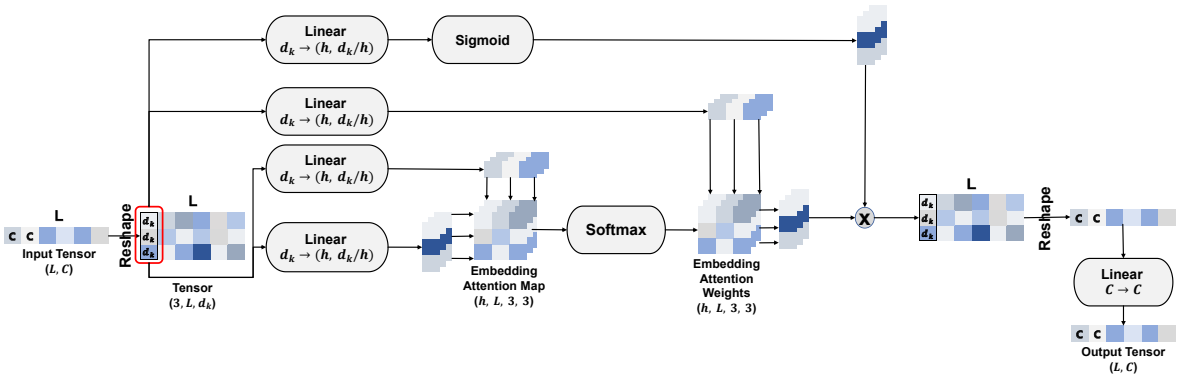
where $W_e^Q \in \mathbf{R}^{d_k\times d_k/h}$, $W_e^K \in \mathbf{R}^{d_k\times d_k/h}$, and h is the number of heads. Embedding attention effectively captures interactions among different embedding approaches within the sequence and dynamically assigns attention scores to these methods for each token in ψ_d.

In the TRSA module, nucleotide attention was applied to ψ_d, focusing specifically on its second

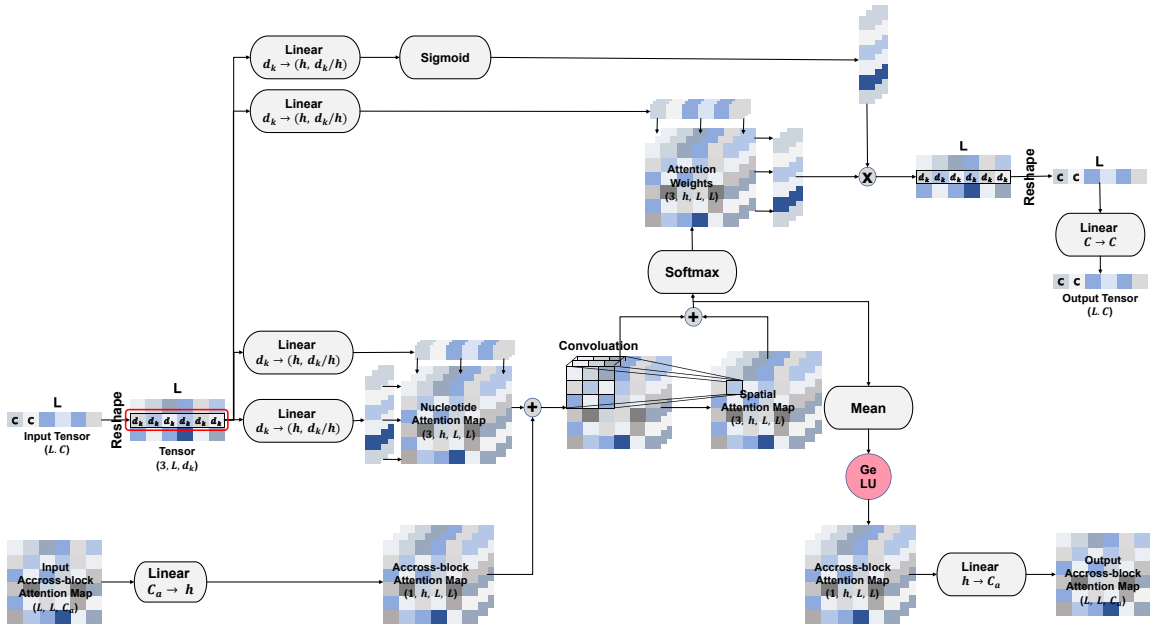
dimension, which represents the sequence length. We denoted the dense vector at the e-th embedding technique of ψ_d as $v_n \in \mathbf{R}^{L \times d_k}$. The nucleotide attention map $\rho_n \in \mathbf{R}^{L \times L}$ can be calculated as follows:

$$\rho_n = \frac{Q_n K_n^T}{\sqrt{d_k}}, \quad where \quad Q_n = v_n W_n^Q, K_n = v_n W_n^K$$

where $W_n^Q \in \mathbf{R}^{d_k \times d_k/h}$ and $W_n^K \in \mathbf{R}^{d_k \times d_k/h}$. This attention mechanism effectively captures interactions among nucleotides within the sequence and assigns dynamic attention scores to each token in ψ_d.



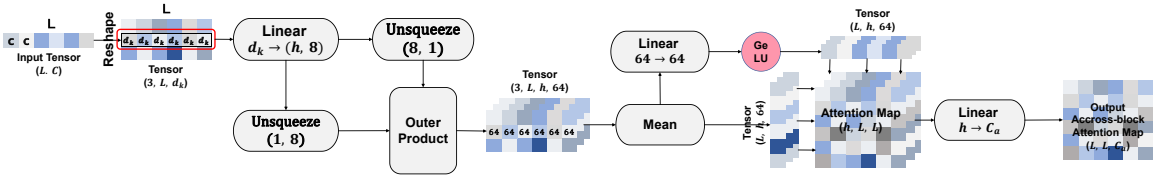
Supplementary Figure 5: Tensor Column-wise gated Self-Attention (TCSA). Dimensions: L: sequence length, C: channels, d_k: channels after reshape, h: heads.



Supplementary Figure 6: Tensor Row-wise gated Self-Attention (TRSA). Dimensions: L: sequence length, C: channels for tokens, d_k: channels for tokens after reshape, C_a: channels for across-block attention map, h: heads.

We incorporated the across-block attention mechanism into the TRSA module to facilitate con-

nections between the learned knowledge of nucleotide interactions across different blocks. Initially, the across-attention map ($\rho_a \in \mathbf{R}^{L\times L\times C_a}$) was set as a zero matrix. Then, ρ_a was updated (added) through the block attention update (BAU) module (**Supplementary Figure** 7) based on the TCSA output. Subsequently, the updated ρ_a was fed into TRSA. After a linear transformation, it was added to ρ_n to serve as an additional nucleotide interaction attention map derived from previous blocks. Further refinement of ρ_a was achieved through a subsequent feedforward module. The implementation of the across-block attention mechanism facilitates the sharing and propagation of nucleotide interaction knowledge learned within individual blocks across different parts of the model.



Supplementary Figure 7: Block Attention Update (BAU) module. Dimensions: L: sequence length, C: channels for tokens, d_k: channels for tokens after reshape, C_a: channels for across-block attention map, h: heads.

Transformer-based models are known for their ability to capture the global context of a sequence but they may struggle with capturing its local context [3]. To address this limitation, we extended the nucleotide attention map and across-block attention map by incorporating spatial attention. We treated the sequence as an image and applied a convolutional layer to extract local spatial context from $\rho_n + Linear(\rho_a)$, resulting in a spatial attention map denoted as ρ_s. In the TRSA module, ρ_s was incorporated into $\rho_n + Linear(\rho_a)$ as a local context attention map to improve the model's ability to capture and comprehend the local context within a given sequence.

9 Architecture of modified EfficientNet

We made three changes to the EfficientNet architecture[4]. First, we substituted the expansion convolutional layers present in inverted bottleneck residual blocks (IBRB) with omni-dimensional dynamic convolution [5], which has been shown to achieve higher accuracy than static convolutions [5]. Second, we replaced the squeeze and excitation blocks [6] in IBRB with the large kernel attention layers [7] to calculate token-wise and channel-wise attention scores for sequence embeddings. Third, we incorporated DeepNorm [8] to prevent gradient vanishing during model training.

10 Strategy for generating negative taxonomic lineages during training

We denoted $T_{k_i} = [\leqslant t_{k_i}]$ as the prefix of taxonomic lineage T_i before k_i rank, where $k_i \in \{phylum, class, order, family, genus, species\}$. During training, k_i was randomly selected, and we treated T_{k_i} as the

positive taxonomic lineage for sequence s_i.

We denote the negative taxonomic lineage as T_{k_j}. During contrastive training, we select 199 negative taxonomic lineages from the taxonomic tree. The half-number of negative taxonomic lineages have three characteristics: 1. $k_i = k_j$, 2. $[< t_{k_i}] = [< t_{k_j}]$, 3. $t_{k_i} \neq t_{k_j}$. The remaining half of these negative lineages are randomly drawn from the taxonomic tree, ensuring they are distinct with T_{k_i}. This generation method is pivotal in Deepurify's contrastive training process, where T_{k_j} with the three specified characteristics act as hard negative samples, increasing the contrastive training difficulty and enhancing model performance [9].

11 Solving the imbalanced phyla classes in GS_c and GS_p

In our data processing pipeline, we incorporated an oversampling strategy to mitigate the issue of imbalanced phyla observed in GS_c and GS_p. This imbalance problem was caused by the varying number of species present across different phyla. The oversampling approach involved duplicating genomes within a phylum until the total number of genomes from that phylum reached a predefined threshold, specifically a minimum of 500 for GS_c and 20 for GS_p. This strategy was only applied to the phylum containing fewer than 500 and 20 species, respectively.

Furthermore, we adopted the focal loss [10] in the training stage to address the issue of label imbalance in the other taxonomic ranks. This additional measure enhanced our model's robustness to imbalanced data distributions.

12 Hyper-parameter setting for training

We applied the following hyper-parameters for training Deepurify, similar to the training configuration of UniFormer. Specifically, we set the stochastic depth rate [11] at 0.01 for the EfficientNet and set the dropout rate [12] to 0.08. We set the weight decay, learning rate, batch size, and number of negative lineages for contrastive learning as $1e^{-5}$, $1e^{-6}$, 10, and 199, respectively. Training Deepurify involved utilizing the AdamW optimizer [13] in conjunction with a cosine learning rate schedule [14], spanning a training period of 96 epochs. The initial 4 epochs were used for linear warm-up.

Subsequently, we fine-tuned Deepurify to minimize the influence of homologous sequences during training. The basic parameters remain unchanged, except setting stochastic depth rate and dropout as 0. The taxonomic encoder (LSTM) parameter was fixed, and only 15 epochs were used for fine-tuning. L_{ST} was replaced with cross-entropy loss but not focal loss. To mitigate the impact of homologous sequences on the model, we computed the loss L_{ST} for a given sequence s_i only within the context of contrastive training if the absolute difference between the top-1 and top-2 predicted probabilities of lineages exceeded 0.05.

13 Determining the number of clusters with SCGs

We used Prodigal and the HMM tool to identify SCGs, resulting in a comprehensive list that matched contigs with their corresponding SCGs. We then sorted the contigs in descending order of length and assigned them to sets one by one, while recording the SCGs in each set. If we encountered duplicate SCGs within a set while adding a contig, we created a new set for that contig. This iterative process continued until all contigs were placed in different sets, resulting in multiple divisions. The number of clusters is equal to the number of divisions minus one.

14 Traversing the MAG-separated tree with DFS

To maximize the number of medium- and high-quality MAGs, Deepurify used post-order traversal to traverse the MAG-separated tree. Each node maintained a list to store its binning outcomes. The current node being traversed is referred to as V_C. $V_{H\_}$ and $V_{M\_}$ are the child nodes of V_C. $V_{H\_}$ must contain high-quality MAGs, while $V_{M\_}$ should not contain high-quality MAGs but must include medium-quality MAGs. We used **Algorithm** 1 to compare and select nodes. The comparison and selection process occurs recursively, starting from the left nodes and progressing up to the root node, resulting in a collection of nodes stored in the list of the root node.

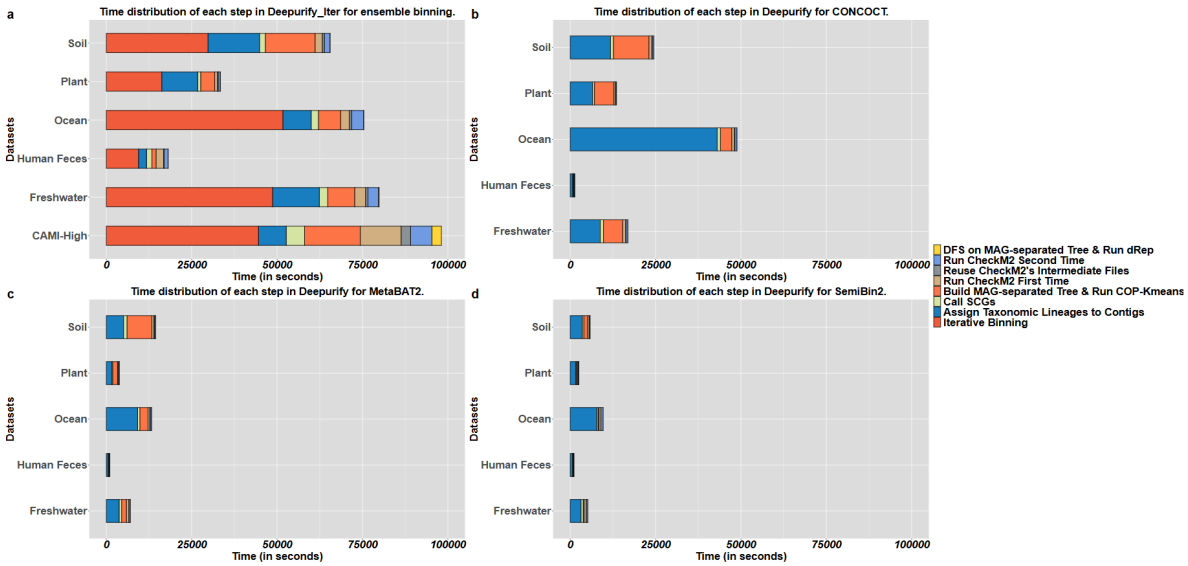
15 Running time distribution

We evaluated the running time distributions of each step in Deepurify_Iter (**Supplementary Figure 8, a**) on CAMI I High complexity datasets and real-world metagenomic sequencing data (a soil sample (SRR25158210), a plant sample (SRR14308228), an ocean sample (1102224), a Human Fecal (IBS-D) sample (K0273), a freshwater sample (SRR26420192)) and showed the running time in **Supplementary Figure 8**. The hardware used in this experiment included: 1. CPU: AMD EPYC 7742 64-Cores Processor (2 Sockets); 2. RAM: 1TB; 3. GPU: 8 x A100-40GB GPUs. We observed that CheckM2 was no longer the bottleneck of Deepurify and the running time was dominated by the contig binning step.

We further analyzed the runtime distributions of different steps in Deepurify when applied to the outputs of CONCOCT (**Supplementary Figure 8b**), MetaBAT2 (**Supplementary Figure 8c**), and SemiBin2 (**Supplementary Figure 8d**) on the same real-world metagenomic sequencing datasets. The experiments were performed using the following hardware: 1. CPU: Intel(R) Xeon(R) Silver 4114, 10-Core Processor; 2. RAM: 512GB; 3. GPU: 2 x V100-32GB GPUs. The assignment of taxonomic lineages to contigs was found to be the most time-consuming step.

Algorithm 1 The comparison and selection of nodes

Input: A $V_{M\_}$ node list : $L_M = [V_{M_1}, ..., V_{M_m}]$,
Input: A $V_{H\_}$ node list: $L_H = [V_{H_1}, ..., V_{H_h}]$,
Input: The node that DFS is currently traversing: V_C.
func: **summedQualityScores** is a function to calculate the summed quality score of MAGs in a node list; **getHighQualityNum**, **getMediumQualityNum** are the functions to get the number of high-, medium-quality MAGs from a node list.
Output: None.
function comparison(L_M, L_H, V_C):
    curNodeRes = V_c.binningResList
    h = getHighQualityNum(curNodeRes)
    m = getMediumQualityNum(curNodeRes)
    h_c = size(L_H)
    m_c = size(L_M)
    $qsSum$ = summedQualityScores(curNodeRes)
    $qsSum_c$ = summedQualityScores(L_M) + summedQualityScores(L_H)
    if $h > h_c$:
        return
    elif $h == h_c$:
        if $m > m_c$:
            return
        elif $m == m_c$:
            if $qsSum > qsSum_c$:
                return
            else:
                V_c.binningResList = $L_H + L_M$
        else:
            V_c.binningResList = $L_H + L_M$
    else:
        V_c.binningResList = $L_H + L_M$



Supplementary Figure 8: (**a**). Running time distributions (in seconds) for the different steps in Deepurify_Iter (**a**) and Deepurify (**b-d**). (**a**) Running time distribution of Deepurify_Iter on metagenomic sequencing data from soil (SRR25158210), plant (SRR14308228), ocean (1102234), human feces (K0273), freshwater (SRR26420192), and CAMI I High complexity datasets. (**b-d**) Running time distributions of Deepurify applied to the results from CONCOCT (**b**), MetaBAT2 (**c**), and SemiBin2 (**d**) on five real-world metagenomic sequencing data. 'Iterative Binning' refers to running CONCOCT, MetaBAT2, and SemiBin2. 'Assign Taxonomic Lineages to Contigs' involves calculating contig embeddings and identifying their taxonomic lineages. 'Call SCGs' refers to using Prodigal and HMMER for identifying SCGs in MAGs. 'Build MAG-separated Tree & Run COP-Kmeans' includes constructing MAG-separated trees and implementing COP-Kmeans for each node within these trees. 'Run CheckM2 First Time' is the application of CheckM2 to assess MAGs' quality. 'Reuse CheckM2's Intermediate Files' entails employing intermediate files from CheckM2, such as Prodigal and DIAMOND outputs to build intermediate files for sub-MAGs. 'Run CheckM2 Second Time' involves executing CheckM2 using the intermediate files prepared previously. 'DFS on MAG-separated tree & Run dRep' includes performing DFS on MAG-separated trees to select the maximum number of high- and medium-quality MAGs and applying dRep to eliminate duplicate MAGs.

16 Ablation study for multi-modal model and MAG-separated tree

We performed two ablation studies on the CAMI High complexity dataset to investigate the necessity of using the multi-modal model (sequence embeddings) and the MAG-separated tree.

(Ablation Study 1) We adopted a simpler method to build the MAG-separated tree. Given a MAG, we predicted its core lineage using the score function designed for simulation experiments (**Figure 2b**). In the simplified MAG-separated tree, each taxonomic rank only includes one node that represents the corresponding value from the core lineage. We then selected sub-MAGs from the six nodes (from Phylum to Species) with the highest summation of quality scores ($\Sigma QS(i)$).

(Ablation Study 2) We implemented an iterative greedy approach to select subsets of contigs. Initially, we calculated the average embeddings of all contigs in a MAG, which were generated by GseqFormer (from the multi-modal model). We computed the cosine distances between this average value and each contig's embedding. The contig that was most distant from the average was removed. We then calculated the quality score based on completeness and contamination. This process was carried out repeatedly until there was no further improvement in the MAG's quality score.

We compared the performance of Deepurify_Iter and the models from Ablation studies 1 and 2 on the CAMI I High complexity dataset (**Supplementary Table 3**). This result indicates the MAG-separated tree and the sequence embeddings from the multi-modal model in Deepurify could generate more high-quality MAGs and improve MAG quality scores.

Ensemble Binning	CAMI I High		
Pass GUNC	High	Medium	QS
Before Decontamination	118	127	18849.25
Ablation Study 1	147	152	23407.19
Ablation Study 2	134	125	20350.36
Deepurify_Iter	151	166	24253.64

Supplementary Table 3: The number of high- (High) and medium-quality (Median) MAGs that passed GUNC criterion on contamination, along with the quality scores across CAMI I High datasets for ensemble baseline, two ablation experiments, and Deepurify_Iter. 'Ensemble Binning' refers to the integration of MAGs generated by CONCOCT, MetaBAT2, and SemiBin2.

17 Software versions and computational environment

Our study used MAGpurify (v2.1.2) and MDMcleaner (v0.8.7) for MAG decontamination. The development of Deepurify was developed using Python (v3.8.18) along with PyTorch (v2.0.0 + cu118). The SCGs calling was executed using Prodigal (v2.6.3) and HMMER (v3.3.2). The computation of the balanced macro F1-score was performed using Scikit-Learn (v1.2.0). The evaluation of MAG quality was carried out using CheckM2 (v1.0.1). For binning, we employed CONCOCT (v1.1.0),

MetaBAT2 (v2.15), and SemiBin2 (v2.1.0). The annotation of MAGs was executed using GTDB-Tk (v1.4.0). In this study, metaSPAdes (v3.15.0) and MegaHit (v1.2.9) were applied for assembly. We applied MMseqs2 (v14.7e284) to cluster sequences. The GitHub link for this study is 'https://github.com/ericcombiolab/Deepurify/'.

The experiments were conducted on a compute node with two AMD EPYC 7742 processors comprising 64 cores (128 threads) and 1 TB of memory. Eight NVIDIA Tesla A100-40GB GPUs were engaged to expedite the Deepurify training. During inference, the utilization was on eight Tesla A100-40GB GPUs, with two threads allocated for data feeding on each GPU. Other MAG decontamination tools were run with 256 threads.

18 Evaluating Deepurify's performance on real-world metagenomic sequencing data by processing outputs from individual binning tools

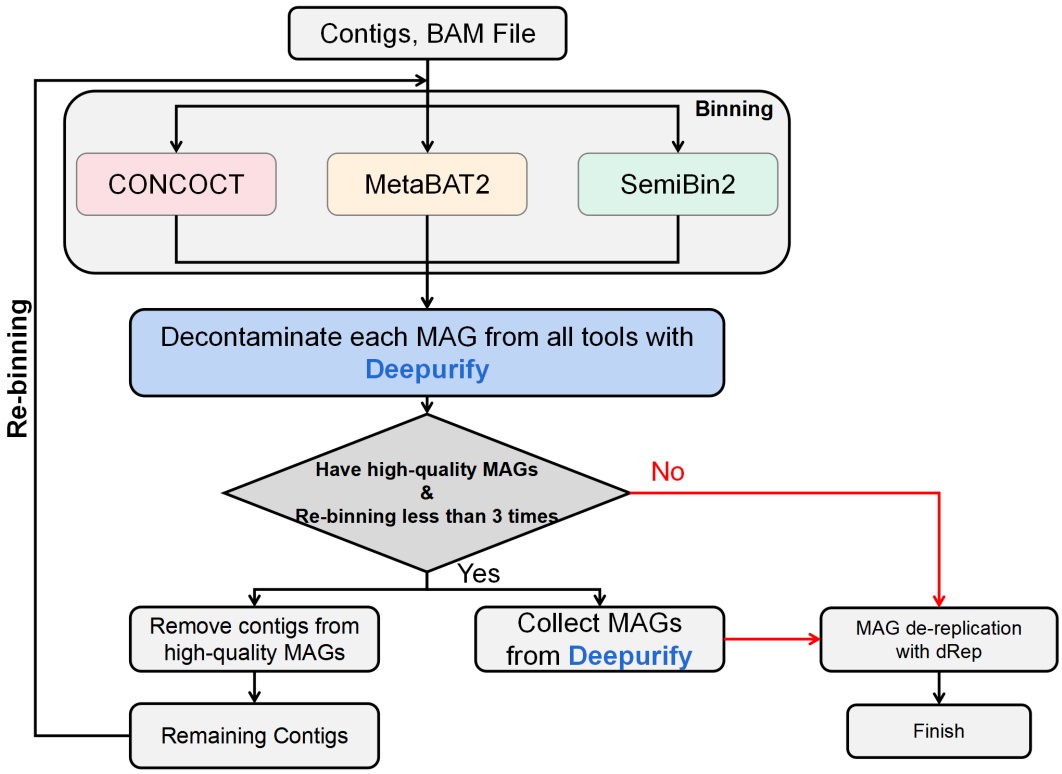
We applied Deepurify to the outputs of individual binning tools (CONCOCT, MetaBAT2, and SemiBin2) on various metagenomic sequencing datasets, including soil (7 samples), ocean (11 samples), freshwater (3 samples), plant (3 samples), and human feces (227 samples). The results (**Supplementary Table** 4) show that Deepurify increased the number of high-quality MAGs across all three binning tools, producing 3.88, 1.16, and 1.18 times more high-quality MAGs from CONCOCT, MetaBAT2, and SemiBin2, respectively.

To demonstrate the effectiveness of Deepurify in significantly reducing contamination in highly contaminated MAGs, we selected MAGs with contamination levels exceeding 10% from the original binning results of CONCOCT, MetaBAT2, and SemiBin2 across all five real-world metagenomic sequencing datasets (251 samples). After processing these MAGs with Deepurify, we observed a substantial reduction in contamination levels (Wilcoxon Signed-Rank Sum test, two-sided, p-value = 2.2e-16, confidence interval: [-78.1, -75.7], effect size: -1.36; **Supplementary Figure 10**).

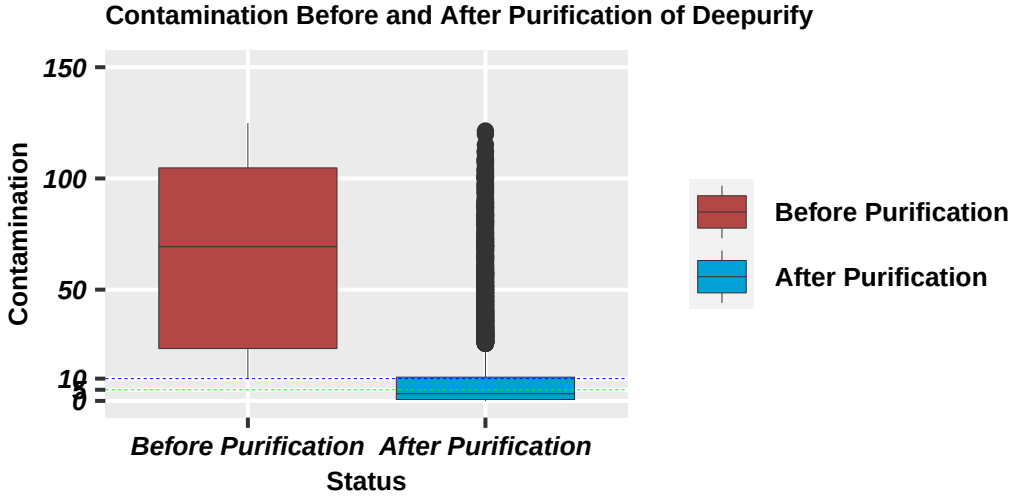
CONCOCT	Soil (7 Samples)			Ocean (11 Samples)			Plant (3 Samples)			Freshwater (3 Samples)			Human feces (227 Samples)		
Pass GUNC	High	Medium	QS (k)	High	Medium	QS (k)	High	Medium	QS (k)	High	Medium	QS (k)	High	Medium	QS (k)
Before Decontamination	3	28	1.60	13	77	4.15	1	13	0.76	13	55	4.26	800	881	125.22
Deepurify	6	95	4.44	71	453	23.6	18	127	7.46	51	225	15.17	3080	3631	498.48
MeatBAT2	Soil (7 Samples)			Ocean (11 Samples)			Plant (3 Samples)			Freshwater (3 Samples)			Human feces (227 Samples)		
Pass GUNC	High	Medium	QS (k)	High	Medium	QS (k)	High	Medium	QS (k)	High	Medium	QS (k)	High	Medium	QS (k)
Before Decontamination	14	56	4.16	71	174	17.53	28	46	5.23	57	131	13.26	2125	2858	382.60
Deepurify	19	121	7.20	88	374	28.04	32	127	9.22	65	229	18.96	2474	5125	544.18
SemiBin2	Soil (7 Samples)			Ocean (11 Samples)			Plant (3 Samples)			Freshwater (3 Samples)			Human feces (227 Samples)		
Pass GUNC	High	Medium	QS (k)	High	Medium	QS (k)	High	Medium	QS (k)	High	Medium	QS (k)	High	Medium	QS (k)
Before Decontamination	7	92	5.05	82	241	21.51	18	47	4.39	59	158	15.15	2902	3130	473.53
Deepurify	12	184	9.12	99	430	32.29	27	111	8.31	66	251	20.55	3432	5457	654.69

Supplementary Table 4: The number of high- (High) and medium-quality (Median) MAGs that passed the GUNC criterion on contamination, along with the quality scores (QS, 1k = 1,000) across five real-world datasets.

Supplementary Figures



Supplementary Figure 9: The workflow of the iterative decontamination strategy used in Deepurify_Iter.



Supplementary Figure 10: Contamination levels of MAGs before (red, n = 6,492) and after (blue, n = 10,921) applying Deepurify. The analysis included 6,492 highly contaminated MAGs from 251 real-world metagenome sequencing samples. The blue dashed line represents the contamination threshold for medium-quality MAGs, while the green dashed line indicates the threshold for high-quality MAGs. All the box plots depict the median (horizontal line inside box), 25th and 75th percentiles (box), and 25th or 75th percentiles ± 1.5 × interquartile range (whiskers). The other points are outliers.

References

[1] Wood, D. E., Lu, J. & Langmead, B. Improved metagenomic analysis with kraken 2. *Genome biology* **20**, 1–13 (2019).

[2] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. & Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4510–4520 (2018).

[3] Nguyen, P. M., Le, T., Nguyen, H. T., Tran, V. & Nguyen, M. L. Phrasetransformer: an incorporation of local context information into sequence-to-sequence semantic parsing. *Applied Intelligence* **53**, 15889–15908 (2023).

[4] Tan, M. & Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, 6105–6114 (PMLR, 2019).

[5] Li, C., Zhou, A. & Yao, A. Omni-dimensional dynamic convolution. In *International Conference on Learning Representations* (2021).

[6] Iandola, F. N. *et al.* Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360* (2016).

[7] Guo, M.-H., Lu, C.-Z., Liu, Z.-N., Cheng, M.-M. & Hu, S.-M. Visual attention network. *arXiv preprint arXiv:2202.09741* (2022).

[8] Wang, H. *et al.* Deepnet: Scaling transformers to 1,000 layers. *arXiv preprint arXiv:2203.00555* (2022).

[9] Robinson, J., Chuang, C.-Y., Sra, S. & Jegelka, S. Contrastive learning with hard negative samples. *arXiv preprint arXiv:2010.04592* (2020).

[10] Lin, T.-Y., Goyal, P., Girshick, R., He, K. & Dollár, P. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, 2980–2988 (2017).

[11] Huang, G., Sun, Y., Liu, Z., Sedra, D. & Weinberger, K. Q. Deep networks with stochastic depth. In *European conference on computer vision*, 646–661 (Springer, 2016).

[12] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* **15**, 1929–1958 (2014).

[13] Loshchilov, I. & Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).

[14] Loshchilov, I. & Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983* (2016).