Rendering omics profiles in functional network space with NEArender

Ashwini Jeggari and Andrey Alexeyenko

Contents

1 Package installation **2**

2 Overview **2**

3 A small offline example **6**

4 Species and identifier namespaces **6**

5 Preparing and using altered gene sets **8**

5.1 Continuous profiles in matrix form 8

5.2 Mutation and other event matrices 9

5.3 Importing precompiled AGSs 9

6 Preparing functional gene sets **11**

7 Preparing a global network **13**

7.1 Curated networks 13

7.2 NDEx networks 14

7.3 Custom networks 15

8 Renderers **16**

8.1 NEA 16

8.2 Over-representation analysis 20

8.3 Empirical significance with `ear()` 23

9 Other rendering methods **24**

9.1 Random walks and PageRank 24

9.2 Texture and spectral renderers 24

9.3 GSVA-family renderers 25

10 Edge scores, edge weights, and transition probabilities **25**

11 Investigating and comparing networks **26**
11.1 Degree distribution . . . 26
11.2 Second-order topology . . . 26
11.3 Exact network overlap . . . 26
11.4 Pathway-member recovery . . . 27

12 A smooth-start guide **29**

13 Reproducible analysis checklist **37**

14 References **39**

1 Package installation

Install the development release from GitHub:

```
install.packages("remotes")
remotes::install_github(
  "andreyalexeyenko/NEArender",
  build_vignettes = TRUE
)
```

NEArender.R

Alternatively, install a locally built source package:

```
install.packages(
  "NEArender_2.0.tar.gz",
  repos = NULL,
  type = "source"
)
```

NEArender.R

Load the installed package to start your analyses:

```
library(NEArender)
```

NEArender.R

2 Overview

NEArender transforms gene-level omics datasets describing biological samples into smaller pathway-by-sample matrices. Its original method is network enrichment analysis (NEA), which evaluates whether an altered gene set (AGS) has significantly more or fewer network connections to a functional gene set (FGS) than expected by chance, given the network degrees of their members. The resulting pathway-level features can be used for exploratory interpretation or as inputs to downstream statistical and machine-learning models.

The three principal inputs to `nea.render()` are:

1. **Altered gene sets (AGSs).** User-defined sets representing experimental samples, patients, perturbations, phenotypes, or hypotheses, such as data from the DepMap portal.
2. **Functional gene sets (FGSs).** Sets with a shared biological interpretation, such as pathways from Reactome, processes from the Gene Ontology, diseases from the Disease Ontology, or cell-type signatures from MSigDB.
3. **A global network (NET).** An undirected graph whose edges represent a stated form of functional or physical association between molecular nodes.

AGS members can be genes, proteins, transcripts, miRNAs, or other entities, provided that the same entities occur as network nodes. Public pathway and network resources are nevertheless predominantly gene centered. AGS, FGS, and network inputs should describe the same molecular level and identifier namespace. In this vignette, "gene" is shorthand for such a consistently identified network node, and "pathway" is sometimes shorthand for an FGS.

The package also provides renderers that do not require AGSs, a network, or both. Figure 1 summarizes how user profiles and biological resources enter the workflow.

NEArender.R

The package supports conversion among major gene, transcript, and protein identifier formats for eight organisms. Conversion occurs at input boundaries, and renderers stop when normalized inputs have incompatible species or identifier metadata. Unsupported organisms and non-biological node identifiers can still be submitted with `species = NULL`, provided that all required components use a compatible declared namespace.

Targeted assays and diagnostic panels cover restricted, often functionally biased gene universes. Ordinary enrichment results can therefore be biased. `ear()` provides empirical calibration of NEA or ORA against random AGSs drawn from the same explicitly supplied restricted universe.

Network-free alternatives include over-representation analysis (ORA) and, through the GSVA package, GSVA, ssGSEA, z-score, and PLAGE. Network-based alternatives include random walk with restart, personalized PageRank through `igraph`, network texture, and spectral analysis. All renderers return a common `nearender_result` structure.

NEArender.R

NEA and the graph-flow renderers consume AGSs, FGSs, and a network. ORA uses AGSs, FGSs, and an eligible universe but no network. Continuous-profile renderers consume a gene-by-sample matrix directly. Their interpretations also differ: enrichment methods evaluate overrepresentation of nodes or edges, whereas activation methods summarize member states within an FGS.

Renderer	Uses network	Gene-by-sample input	Approach	Statistical significance
NEA	Yes	Via AGS	Enrichment	Yes
Spectral	Yes	Direct	Activation	No
Texture	Yes	Direct	Activation	No
Random walk	Yes	Via AGS	Enrichment	No
PageRank	Yes	Via AGS	Enrichment	No
ORA	No	Via AGS	Enrichment	Yes
GSVA	No	Direct	Activation	No
z-score via GSVA	No	Direct	Activation	No
PLAGE via GSVA	No	Direct	Activation	No
ssGSEA via GSVA	No	Via fully ranked gene lists	Enrichment	No

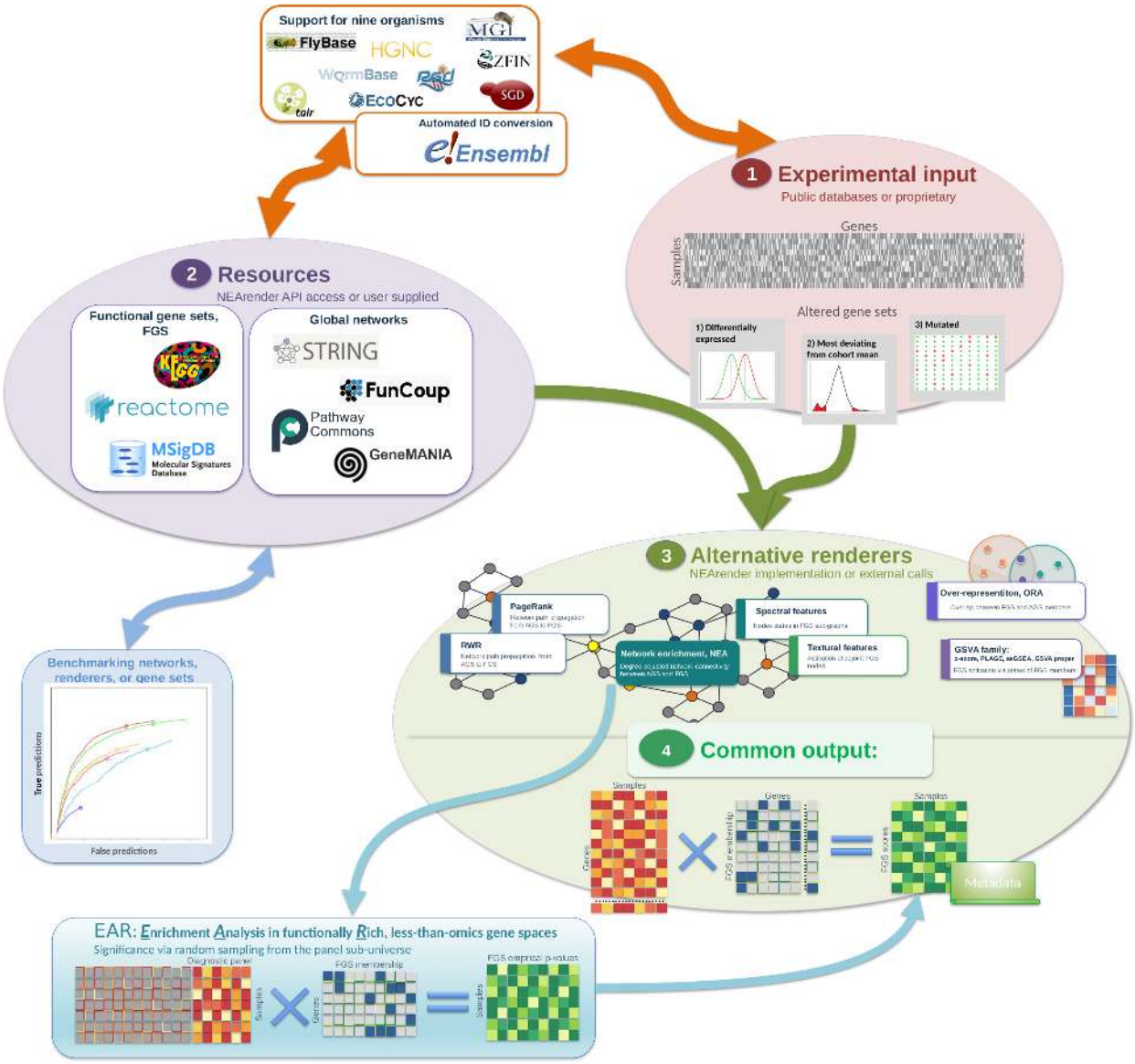

Figure 1: NEArender workflow.

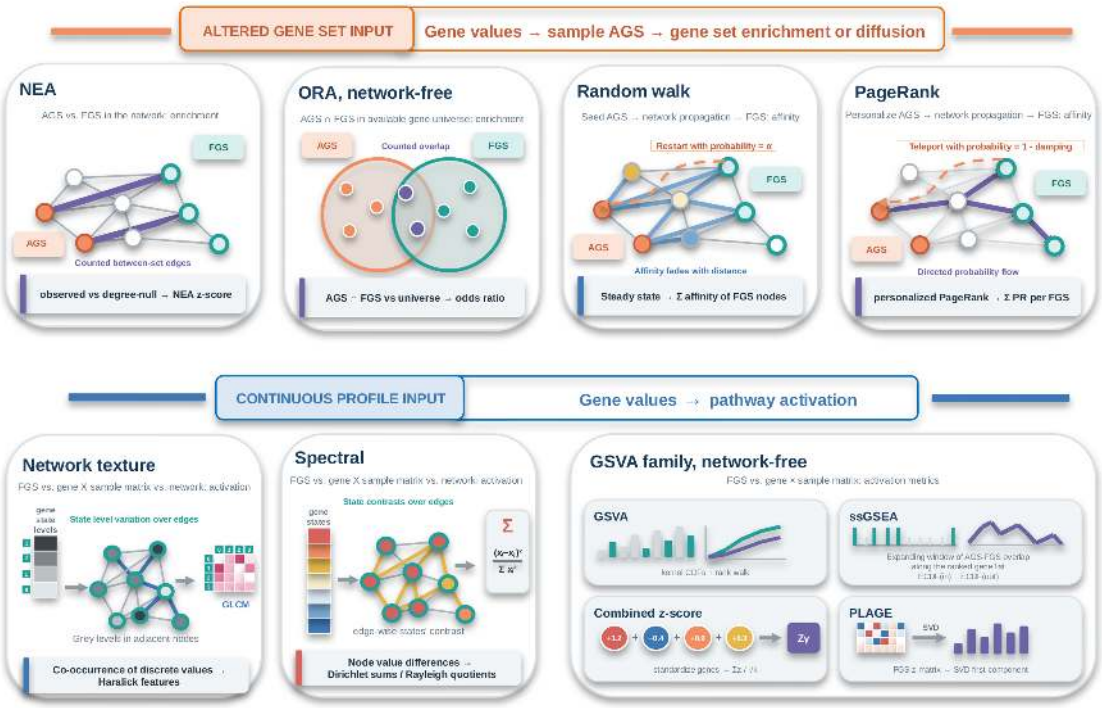

Figure 2: Alternative renderers available in NEArender.

3 A small offline example

The package includes one real-world example object for documentation and testing, so the vignette does not require network access.

```
data("NEArender.test")
names(NEArender.test)
#> [1] "FGS"   "AGS"   "NET"   "mtr"   "panel"
names(NEArender.test$AGS)
#> [1] "CCLE" "LUAD"
names(NEArender.test$FGS)
#> [1] "Hallmarks"
names(NEArender.test$NET)
#> [1] "STRING.300K"
names(NEArender.test$mtr)
#> [1] "LUAD" "CCLE"
names(NEArender.test$panel)
#> [1] "Olink.Inflammation250"
```

NEArender.R

It contains top-100 expression-derived AGSs from CCLE, top-200 AGSs from TCGA LUAD samples, Hallmark gene sets, and a 300,000-edge human STRING network. All identifiers are HGNC gene symbols. The source STRING scores were used to select the highest-ranked unique edges, not as transition probabilities. The object also contains the underlying CCLE and LUAD expression matrices and a 254-protein Olink inflammation panel for offline renderer and EAR examples. The CCLE data are derived from the CC BY 4.0 DepMap 24Q4 Public release: batch-corrected log2(TPM + 1) values were standardized per gene across all 1,673 models before the ten example cell lines and their top 100 genes were selected.

The LUAD matrix is derived from the cBioPortal `luad_tcga_pan_can_atlas_2018` study's log-transformed RNA Seq V2 RSEM z-scores relative to all 510 samples. Ten samples were selected, duplicate gene-symbol rows were averaged, and the 200 highest finite z-scores per sample were retained as AGSs. The 50 Hallmark sets are from MSigDB release 6.0, licensed CC BY 4.0; gene-set names and symbols were converted to lower case, without changing memberships.

We use small AGS and FGS subsets below while retaining a realistic global network.

```
ags <- NEArender.test$AGS$CCLE[1:3]
fgs <- NEArender.test$FGS$Hallmarks[1:6]
net <- NEArender.test$NET$STRING.300K
```

NEArender.R

4 Species and identifier namespaces

Every analysis requires all inputs to use the same identifier namespace. NEArender does not infer species from identifier strings. Every user-data boundary therefore requires `species`; there is deliberately no default. The supported organisms and their principal aliases are:

```
print(
  unique(available_id_formats()[, 1:4]),
  row.names = FALSE
)
```

```
#>        species_id species                    organism tax_id
#>          hsapiens     hsa                Homo sapiens   9606
#>         mmusculus     mmu                Mus musculus  10090
#>       rnorvegicus     rno            Rattus norvegicus  10116
#>            drerio     dre                 Danio rerio   7955
#>     dmelanogaster     dme      Drosophila melanogaster   7227
#>          celegans     cel       Caenorhabditis elegans   6239
#>       scerevisiae     sce     Saccharomyces cerevisiae   4932
#>         athaliana     ath         Arabidopsis thaliana   3702
#>  ecoli_k12_mg1655     eco Escherichia coli K-12 MG1655 511145
```

NEArender.R

A species may be submitted using any value shown in the `species_id`, `species`, `organism`, or `tax_id` columns. This includes scientific and common names, package species IDs, three-letter KEGG style organism IDs, and NCBI taxonomy IDs.

```
master_id_format("9606")
#> [1] "HGNC"
master_id_format("Homo sapiens")
#> [1] "HGNC"
available_id_formats("human")
#>   species_id species     organism tax_id provider               dataset
#> 1   hsapiens     hsa Homo sapiens   9606  Ensembl hsapiens_gene_ensembl
#> 2   hsapiens     hsa Homo sapiens   9606  Ensembl hsapiens_gene_ensembl
#> 3   hsapiens     hsa Homo sapiens   9606  Ensembl hsapiens_gene_ensembl
#> 4   hsapiens     hsa Homo sapiens   9606  Ensembl hsapiens_gene_ensembl
#> 5   hsapiens     hsa Homo sapiens   9606  Ensembl hsapiens_gene_ensembl
#> 6   hsapiens     hsa Homo sapiens   9606  Ensembl hsapiens_gene_ensembl
#> 7   hsapiens     hsa Homo sapiens   9606  Ensembl hsapiens_gene_ensembl
#>     master_id             id_format   category description_attribute
#> 1 hgnc_symbol           hgnc_symbol     master           description
#> 2 hgnc_symbol       ensembl_gene_id       gene           description
#> 3 hgnc_symbol    ensembl_peptide_id    protein           description
#> 4 hgnc_symbol ensembl_transcript_id transcript           description
#> 5 hgnc_symbol               hgnc_id       gene           description
#> 6 hgnc_symbol         entrezgene_id       gene           description
#> 7 hgnc_symbol      uniprotswissprot       gene           description
```

NEArender.R

The default submitted namespace is `input_id_format = master_id_format(species)`, usually an official gene-symbol namespace. For human data this is HGNC. The declared namespace is checked during normalization, and incompatible normalized inputs stop before analysis.

Use a non-master format explicitly and supply an identifier mapping when conversion is required:

```
cache <- file.path(tempdir(), "NEArender-cache")
prepare_id_mappings("human", cache_dir = cache)
human_mappings <- load_id_mappings("human", cache_dir = cache)

ags_ensembl <- import_gene_sets(
  "altered-sets.tsv",
  gene_column = "ensembl_gene",
```

```
  set_column = "sample",
  header = TRUE,
  species = "9606",
  input_id_format = "ensembl_gene_id",
  id_mappings = human_mappings
)

normalization_info(ags_ensembl)
```

NEArender.R

`species = NULL` is an explicit escape hatch for unsupported organisms or non-biological node identifiers. In that case, `input_id_format` must still be supplied, and the user assumes responsibility for namespace compatibility.

5 Preparing and using altered gene sets

The AGS is usually the most dynamic and user-specific input. It can be prepared in either of two ways:

- from an original data matrix; or
- by importing a precompiled list representation.

5.1 Continuous profiles in matrix form

`samples2ags()` converts a numeric gene-by-sample matrix, such as a transcriptomic profile, into one AGS per sample. Columns represent samples or experimental conditions and rows represent genes. Genes absent from the network will not contribute to network-based analyses.

The available preparation rules are:

- `"top"`: subtract each gene's across-sample mean, rank the absolute unstandardized values, and retain `n_top` genes. Both positive and negative deviations can be selected.
- `"toppos"`: subtract each gene's across-sample mean and rank the signed, unstandardized values in descending order, thus selecting the upper side of the value distribution. It remains a fixed-`n_top` method and does not apply a positivity cutoff. This can suit skewed or zero-censored profiles, including some single-cell data.
- `"topnorm"`: rank absolute within-gene z-scores relative to the across-sample mean and retain `n_top` genes. Every nonempty AGS has the same requested size.
- `"bestp"`: calculate within-gene z-scores and, for each sample value, calculate the one-sided upper-tail probability of `abs(z)`. Rank genes by that probability and retain the `n_top` lowest values. The selected AGS may be enriched for informative genes, but membership does not assert a fixed significance threshold.
- `"significant"`: calculate the same one-sided z-test probabilities, adjust them across genes with `stats::p.adjust(..., method = "BH")`, and retain genes below `cutoff_q`. Both positive and negative deviations are eligible; unlike `nea.render()` q-values, there is no enrichment-only convention that assigns one direction an adjusted value of 1. Because the threshold is data driven, an AGS may be empty.
- `"toprandom"`: sample `n_top` genes from the matrix row names without using their values.

```r
set.seed(1)
profile_genes <- names(net$links)[1:200]
profiles <- matrix(
  rnorm(length(profile_genes) * 4),
  nrow = length(profile_genes),
  dimnames = list(profile_genes, paste0("sample_", 1:4))
)

prepared_ags <- samples2ags(
  profiles,
  n_top = 30,
  method = "topnorm",
  species = "human"
)
lengths(prepared_ags)
#> sample_1 sample_2 sample_3 sample_4
#>       30       30       30       30
```

NEArender.R

Fixed-size lists are straightforward and convenient for comparisons; significance-based lists preserve data-driven variation in AGS size.

Sample columns can be selected by name or index through `sample_columns`, or by a regular expression through `column_pattern`. For example, `column_pattern = "[-.]01$"` applied to TCGA barcodes selects primary tumour samples whose sample code ends in `01`.

5.2 Mutation and other event matrices

`mutations2ags()` treats every nonzero, nonmissing matrix entry as an event. Event categories and magnitudes are otherwise ignored.

```r
set.seed(2)
events <- matrix(
  rbinom(length(profile_genes) * 4, size = 1, prob = 0.03),
  nrow = length(profile_genes),
  dimnames = list(profile_genes, paste0("tumour_", 1:4))
)

mutation_ags <- mutations2ags(events, species = "human")
lengths(mutation_ags)
#> tumour_1 tumour_2 tumour_3 tumour_4
#>       10        8        8        8
```

NEArender.R

The same `sample_columns` and `column_pattern` arguments select event-matrix columns.

5.3 Importing precompiled AGSs

`import_gene_sets()` accepts a file, URL, connection, matrix, or data frame. In `layout = "grouped"`, each row associates `gene_column` with `set_column`, and each distinct set label becomes one list element. Such a table can be prepared externally or recreated from a list written by `save_gs_list()`.

```r
ags_table <- data.frame(
  gene = c("TP53", "KRAS", "EGFR", "PTEN"),
  sample = c("case_a", "case_a", "case_b", "case_b")
)

imported_ags <- import_gene_sets(
  ags_table,
  gene_column = "gene",
  set_column = "sample",
  species = "human"
)
imported_ags
#> $case_a
#> [1] "tp53" "kras"
#>
#> $case_b
#> [1] "egfr" "pten"
#>
#> attr(,"nearender_normalization")
#> attr(,"nearender_normalization")$normalized
#> [1] TRUE
#>
#> attr(,"nearender_normalization")$case_sensitive
#> [1] FALSE
#>
#> attr(,"nearender_normalization")$normalizer_version
#> [1] 1
#>
#> attr(,"nearender_normalization")$identifiers
#> attr(,"nearender_normalization")$identifiers$species_id
#> [1] "hsapiens"
#>
#> attr(,"nearender_normalization")$identifiers$species
#> [1] "hsa"
#>
#> attr(,"nearender_normalization")$identifiers$organism
#> [1] "Homo sapiens"
#>
#> attr(,"nearender_normalization")$identifiers$tax_id
#> [1] "9606"
#>
#> attr(,"nearender_normalization")$identifiers$master_id_format
#> [1] "HGNC"
#>
#> attr(,"nearender_normalization")$identifiers$master_id
#> [1] "hgnc_symbol"
#>
#> attr(,"nearender_normalization")$identifiers$input_id_format
#> [1] "hgnc_symbol"
#>
#> attr(,"nearender_normalization")$identifiers$output_id_format
#> [1] "hgnc_symbol"
#>
```

```
#> attr(,"nearender_normalization")$identifiers$mapping
#> NULL
```

NEArender.R

Text files are tab-delimited by default. In an R string, the default separator is written as `"\t"`: the backslash and `t` encode one tab character rather than two literal backslashes. Text-reader arguments such as `comment_char` are ignored when `source` is already a matrix or data frame.

6 Preparing functional gene sets

AGSs and FGSs share the same named-list representation, so custom FGS tables use `import_gene_sets()` in the same way. `layout = "single_set"` combines a gene column into one set named by `set_name`; `layout = "single_gene"` makes each distinct gene its own single-member FGS. Single-gene FGSs allow NEA to evaluate edge enrichment around individual genes. ORA has no analogous use, because it evaluates gene overlap rather than network edges.

A full single-gene FGS collection can be created from network nodes with:

```
gene_fgs <- import_gene_sets(
  data.frame(gene = names(net$links)),
  gene_column = "gene",
  layout = "single_gene",
  species = "human"
)
```

NEArender.R

Curated pathway collections are discoverable through the offline database catalog.

```
pathway_catalog <- available_databases(
  type = "pathways",
  species = "human"
)
pathway_catalog[, c(
  "id", "provider", "collection", "version",
  "expected_gene_sets", "requires_acceptance"
)]
#>                              id provider      collection       version
#> 1            reactome_pathways Reactome            <NA>            97
#> 2               msigdb_h_human   MSigDB               H        2026.1
#> 3     msigdb_cp_biocarta_human   MSigDB    CP:BIOCARTA        2026.1
#> 4  msigdb_cp_kegg_medicus_human   MSigDB CP:KEGG_MEDICUS        2026.1
#> 5          msigdb_cp_pid_human   MSigDB          CP:PID        2026.1
#> 6  msigdb_cp_wikipathways_human   MSigDB CP:WIKIPATHWAYS        2026.1
#> 7   msigdb_cp_kegg_legacy_human   MSigDB  CP:KEGG_LEGACY        2026.1
#> 8               msigdb_c6_human   MSigDB              C6        2026.1
#> 9               msigdb_c7_human   MSigDB              C7        2026.1
#> 10              msigdb_c8_human   MSigDB              C8        2026.1
#> 11         kegg_pathways_human     KEGG         PATHWAY user-supplied
#>    expected_gene_sets requires_acceptance
#> 1                  NA               FALSE
#> 2                  50                TRUE
```

```
#> 3                292             TRUE
#> 4                658             TRUE
#> 5                196             TRUE
#> 6                925             TRUE
#> 7                186             TRUE
#> 8                189             TRUE
#> 9               5219             TRUE
#> 10               866             TRUE
#> 11                NA             TRUE
```

NEArender.R

Downloading is always explicit. Analytical functions never download data. For example, Reactome can be cached and loaded with:

```
cache <- file.path(tempdir(), "NEArender-cache")
download_geneset_database("reactome_pathways", cache_dir = cache)
reactome_fgs <- load_geneset_database("reactome_pathways", cache_dir = cache)
manifest(reactome_fgs)
```

NEArender.R

Selected human and mouse MSigDB collections are handled similarly after the user reviews and accepts the MSigDB terms:

```
cache <- file.path(tempdir(), "NEArender-cache")
download_geneset_database(
  "msigdb_h_human",
  cache_dir = cache,
  accept_license = TRUE
)
hallmark_resource <- load_geneset_database("msigdb_h_human", cache_dir = cache)
```

NEArender.R

Reactome is obtained independently from its current release; NEArender does not use the older Reactome snapshot inside MSigDB.

Licensed KEGG subscription files are neither downloaded nor redistributed by NEArender. However, a custom, user-supplied table can be imported directly. Column 2 gene symbols are preferred by default because the Ensembl cross-references in column 1 can lag current identifiers.

```
rat_kegg <- import_kegg_pathways(
  "/path/to/KEGG.ALL.rno",
  species = "rno",
  gene_column = "symbol",
  kegg_release = "downloaded 2026-09-01"
)
```

NEArender.R

Column 1 remains available through `gene_column = "ensembl"`, but requires a current `id_mappings` object.

7 Preparing a global network

7.1 Curated networks

List network resources for the intended species first:

```
network_catalog <- available_databases(
  type = "network",
  species = "9606"
)
network_catalog[, c(
  "id", "provider", "version", "identifiers",
  "source_score_column", "score_semantics", "requires_acceptance"
)]
#>                         id        provider    version
#> 1 genemania_human_combined       GeneMANIA 2021-04-19
#> 2    intact_pathwaycommons Pathway Commons         14
#> 3      pathwaycommons_hgnc Pathway Commons         14
#> 4           string12_human          STRING       12.0
#> 5           funcoup6_human         FunCoup        6.0
#>                   identifiers   source_score_column
#> 1          UniProt accessions                Weight
#> 2           HGNC gene symbols                  <NA>
#> 3           HGNC gene symbols INTERACTION_PUBMED_ID
#> 4 Ensembl peptide identifiers        combined_score
#> 5          UniProt accessions             6:FBS_max
#>                                                                    score_semantics
#> 1                                                GeneMANIA combined-network edge score
#> 2                                                        unscored physical interaction
#> 3 derived count of distinct PubMed identifiers plus 0.5; maximum across duplicate edges
#> 4                                      STRING combined edge confidence score (0--1000)
#> 5                                                   maximum final Bayesian edge score
#>   requires_acceptance
#> 1                TRUE
#> 2               FALSE
#> 3                TRUE
#> 4               FALSE
#> 5               FALSE
```

NEArender.R

Then download a selected fixed release and load the verified cached artifact. Large STRING and FunCoup files require either an edge-score threshold or a highest-ranked edge count during loading. Their native protein identifiers should be converted with an explicitly prepared mapping before they are combined with HGNC AGSs or FGSs.

```
cache <- file.path(tempdir(), "NEArender-cache")
prepare_id_mappings("human", cache_dir = cache)
human_mappings <- load_id_mappings("human", cache_dir = cache)

download_network_database("string12_human", cache_dir = cache)
string_net <- load_network_database(
  "string12_human",
  cache_dir = cache,
```

```
  n_top = 300000,
  id_mappings = human_mappings
)
manifest(string_net)
```

NEArender.R

7.2 NDEx networks

NDEx resources are discovered from a live catalog rather than added to the fixed-release catalog above. Search results are ordinary data frames, so a web service can paginate and serialize them without depending on an R-specific network object. Inspect a selected resource before downloading its native snapshot.

```
ndex_catalog <- search_ndex_networks(
  "SIGNOR",
  visibility = "PUBLIC",
  offset = 0,
  limit = 20
)
selected <- ndex_catalog[1, , drop = FALSE]
ndex_network_summary(selected)

cache <- file.path(tempdir(), "NEArender-cache")
download_ndex_network(selected, format = "cx2", cache_dir = cache,
  max_bytes = 1e9)
ndex_net <- load_ndex_network(
  selected,
  species = "human",
  cache_dir = cache,
  node_key = "name"
)
manifest(ndex_net)
```

NEArender.R

Loading never contacts NDEx. `node_key` can instead select `"represents"`, `"id"`, or another scalar CX node attribute. Set `score_column` only when a specific scalar numeric edge attribute has defensible score semantics. The raw CX/CX2 snapshot remains available for applications that need direction, styles, or attributes outside NEArender's analytical projection. A website should pass its own cache directory and enforce server allow-lists, tenant isolation, job limits, and secret handling in its service adapter.

Very large, dense networks can be computationally expensive and can weaken enrichment specificity by adding low-confidence edges. When choosing among network releases or sizes, compare practical member-recovery performance with `nea_member_recovery_benchmark()` rather than assuming that the largest graph is best. A one-million-edge selection is a useful starting point for STRING and FunCoup, but the appropriate threshold remains analysis dependent.

Pathway Commons aggregates interaction evidence from multiple curated resources and records supporting publications for most edges. NEArender uses an edge weight equal to the number of distinct supporting publications plus a 0.5 pseudocount. Consequently, a curated relation without a linked publication remains in the network with weight 0.5. After downloading `pathwaycommons_hgnc`, use `min_score = 1.5` to require at least one cited publication or `min_score = 2.5` to require at least two.

```
cache <- file.path(tempdir(), "NEArender-cache")
download_network_database(
  "pathwaycommons_hgnc",
  cache_dir = cache,
  accept_license = TRUE
)
pwc_resource <- load_network_database(
  "pathwaycommons_hgnc",
  cache_dir = cache,
  min_score = 1.5
)
```

NEArender.R

GeneMANIA and the Pathway Commons aggregate require explicit terms acceptance. The catalog exposes `terms_url` so the applicable terms can be reviewed before setting `accept_license = TRUE`.

Writing acquisition functions require an explicit cache root. Examples and tests use a temporary cache:

```
file.path(tempdir(), "NEArender-cache")
#> [1] "/tmp/RtmpP9lODv/NEArender-cache"
```

NEArender.R

Supply the same `cache_dir` to the corresponding offline load call.

7.3 Custom networks

A custom network can be submitted as an undirected edge list with two endpoint columns. Duplicate and reciprocal edges are collapsed, and self-links are removed. The importer warns with counts and examples when it removes self-links or encounters duplicate representations of the same edge with different edge scores. A scored duplicate is collapsed to one edge carrying the highest submitted edge score.

```
edge_table <- data.frame(
  gene_a = c("TP53", "MDM2", "EGFR", "EGFR"),
  gene_b = c("MDM2", "TP53", "GRB2", "EGFR"),
  confidence = c(0.8, 0.9, 0.7, 1.0)
)

custom_net <- import_network(
  edge_table,
  from_column = "gene_a",
  to_column = "gene_b",
  score_column = "confidence",
  species = "human",
  verbose = FALSE
)
custom_net
#> IGRAPH bc4c534 UNW- 4 2 --
#> + attr: Ntotal (g/n), nearender_case_sensitive (g/l), name (v/c),
#> | weight (e/n)
#> + edges from bc4c534 (vertex names):
#> [1] mdm2--tp53 egfr--grb2
```

The default output is an undirected `igraph`. Sparse and named adjacency-list formats are also available through `output_network_format`, and `convert_network()` converts among supported representations. Sparse networks keep binary topology in `adjacency` and imported edge weights in a separate `weights` matrix, so sparse NEA is unaffected by the presence of edge weights.

8 Renderers

Every renderer returns a `nearender_result` with six top-level components: `scores`, `score_info`, `statistics`, `inputs`, `diagnostics`, and `parameters`. The master matrix in `scores` is renderer-specific and is described by `score_info`; `statistics` contains the remaining analytical layers. For example:

- NEA reports `z`, `n.actual`, `n.expected`, `chi`, `p`, and `q`;
- ORA reports `odds_ratio`, `n.actual`, `p`, and `q`; and
- EAR adds `p.ear`, `q.ear`, `p.trivial`, and `q.trivial`.

8.1 NEA

Network enrichment analysis (NEA) evaluates whether an AGS has more or fewer network edges to an FGS than expected by chance from the node degrees of both sets. The expectations use the total network edge count and the cumulative degrees of the two sets. Edges confined only within the AGS or only within the FGS do not contribute. The analysis is symmetric in AGS and FGS: swapping the two inputs transposes, but does not otherwise change, the score matrices.

```
nea_result <- nea.render(
  ags = ags,
  fgs = fgs,
  net = net,
  n_cores = 1,
  verbose = FALSE
)
nea_result
#> $scores
#> $scores$z
#>                                       msto211h       c4ii       f36p
#> hallmark_tnfa_signaling_via_nfkb    17.7199504 -1.3362160  0.1922665
#> hallmark_hypoxia                     3.9327817 -2.8688929  4.6675221
#> hallmark_cholesterol_homeostasis    -0.2372566  1.4720187  1.3449713
#> hallmark_mitotic_spindle            -6.4010938 -3.6997761 -3.0279074
#> hallmark_wnt_beta_catenin_signaling -1.7226434 -0.1287116 -1.0756123
#> hallmark_tgf_beta_signaling          0.9270816 -0.5734389  0.1504882
#>
#>
#> $score_info
#>   score_id renderer   metric input_variant     mode parameters citation
#> 1        z      nea signed_z       default standard
#>
#> $statistics
#> $statistics$z
#>                                       msto211h       c4ii       f36p
#> hallmark_tnfa_signaling_via_nfkb    17.7199504 -1.3362160  0.1922665
```

```
#> hallmark_hypoxia                      3.9327817 -2.8688929  4.6675221
#> hallmark_cholesterol_homeostasis     -0.2372566  1.4720187  1.3449713
#> hallmark_mitotic_spindle             -6.4010938 -3.6997761 -3.0279074
#> hallmark_wnt_beta_catenin_signaling -1.7226434 -0.1287116 -1.0756123
#> hallmark_tgf_beta_signaling           0.9270816 -0.5734389  0.1504882
#>
#> $statistics$cumulativeDegrees
#> $statistics$cumulativeDegrees$AGS
#> msto211h     c4ii     f36p
#>     3353     2332     2896
#>
#> $statistics$cumulativeDegrees$FGS
#>    hallmark_tnfa_signaling_via_nfkb                   hallmark_hypoxia
#>                               13559                              10671
#>    hallmark_cholesterol_homeostasis            hallmark_mitotic_spindle
#>                                4141                              11513
#> hallmark_wnt_beta_catenin_signaling         hallmark_tgf_beta_signaling
#>                                5035                               5026
#>
#>
#> $statistics$n.actual
#>                                  msto211h c4ii f36p
#> hallmark_tnfa_signaling_via_nfkb      230   43   67
#> hallmark_hypoxia                       90   23   85
#> hallmark_cholesterol_homeostasis       22   22   26
#> hallmark_mitotic_spindle               13   20   33
#> hallmark_wnt_beta_catenin_signaling    19   19   19
#> hallmark_tgf_beta_signaling            33   17   25
#>
#> $statistics$n.expected
#>                                  msto211h     c4ii     f36p
#> hallmark_tnfa_signaling_via_nfkb 75.77221 52.69931 65.44477
#> hallmark_hypoxia                 59.63311 41.47462 51.50536
#> hallmark_cholesterol_homeostasis 23.14129 16.09469 19.98723
#> hallmark_mitotic_spindle         64.33848 44.74719 55.56941
#> hallmark_wnt_beta_catenin_signaling 28.13726 19.56937 24.30227
#> hallmark_tgf_beta_signaling      28.08696 19.53439 24.25883
#>
#> $statistics$chi
#>                                      msto211h        c4ii       f36p
#> hallmark_tnfa_signaling_via_nfkb    313.99664162  1.78547313  0.0369664
#> hallmark_hypoxia                     15.46677219  8.23054621 21.7857627
#> hallmark_cholesterol_homeostasis      0.05629071  2.16683911  1.8089479
#> hallmark_mitotic_spindle             40.97400182 13.68834319  9.1682233
#> hallmark_wnt_beta_catenin_signaling   2.96750025  0.01656669  1.1569419
#> hallmark_tgf_beta_signaling           0.85948022  0.32883213  0.0226467
#>
#> $statistics$p
#>                                         msto211h         c4ii         f36p
#> hallmark_tnfa_signaling_via_nfkb    2.941717e-70 0.1814786912 8.475335e-01
#> hallmark_hypoxia                    8.396847e-05 0.0041191131 3.048539e-06
#> hallmark_cholesterol_homeostasis    8.124577e-01 0.1410158226 1.786345e-01
#> hallmark_mitotic_spindle            1.542678e-10 0.0002157898 2.462535e-03
```

```
#> hallmark_wnt_beta_catenin_signaling 8.495303e-02 0.8975858261 2.821007e-01
#> hallmark_tgf_beta_signaling         3.538842e-01 0.5663475808 8.803795e-01
#>
#> $statistics$q
#>                                        msto211h      c4ii         f36p
#> hallmark_tnfa_signaling_via_nfkb    5.295091e-69 1.0000000 8.975858e-01
#> hallmark_hypoxia                    3.778581e-04 1.0000000 1.829124e-05
#> hallmark_cholesterol_homeostasis    1.000000e+00 0.2820316 2.969651e-01
#> hallmark_mitotic_spindle            1.000000e+00 1.0000000 1.000000e+00
#> hallmark_wnt_beta_catenin_signaling 1.000000e+00 1.0000000 1.000000e+00
#> hallmark_tgf_beta_signaling         4.899935e-01 1.0000000 8.975858e-01
#>
#>
#> $inputs
#> $inputs$ags_names
#> [1] "msto211h" "c4ii"     "f36p"
#>
#> $inputs$fgs_names
#> [1] "hallmark_tnfa_signaling_via_nfkb"    "hallmark_hypoxia"
#> [3] "hallmark_cholesterol_homeostasis"    "hallmark_mitotic_spindle"
#> [5] "hallmark_wnt_beta_catenin_signaling" "hallmark_tgf_beta_signaling"
#>
#> $inputs$ags_sizes
#> msto211h     c4ii     f36p
#>      100      100      100
#>
#> $inputs$fgs_sizes
#>    hallmark_tnfa_signaling_via_nfkb                    hallmark_hypoxia
#>                                 200                                 200
#>    hallmark_cholesterol_homeostasis            hallmark_mitotic_spindle
#>                                  74                                 200
#> hallmark_wnt_beta_catenin_signaling         hallmark_tgf_beta_signaling
#>                                  42                                  54
#>
#> $inputs$network
#> $inputs$network$name
#> [1] NA
#>
#> $inputs$network$n_nodes
#> [1] 17017
#>
#> $inputs$network$n_edges
#> [1] 300000
#>
#>
#>
#> $diagnostics
#> $diagnostics$ags_original_sizes
#> msto211h     c4ii     f36p
#>      100      100      100
#>
#> $diagnostics$ags_network_sizes
#> msto211h     c4ii     f36p
```

```
#>       89       84       91
#>
#> $diagnostics$fgs_original_sizes
#>    hallmark_tnfa_signaling_via_nfkb                  hallmark_hypoxia
#>                                 200                               200
#>    hallmark_cholesterol_homeostasis            hallmark_mitotic_spindle
#>                                  74                               200
#> hallmark_wnt_beta_catenin_signaling         hallmark_tgf_beta_signaling
#>                                  42                                54
#>
#> $diagnostics$fgs_network_sizes
#>    hallmark_tnfa_signaling_via_nfkb                  hallmark_hypoxia
#>                                 191                               188
#>    hallmark_cholesterol_homeostasis            hallmark_mitotic_spindle
#>                                  72                               197
#> hallmark_wnt_beta_catenin_signaling         hallmark_tgf_beta_signaling
#>                                  42                                54
#>
#>
#> $parameters
#> $parameters$case_sensitive
#> [1] FALSE
#>
#> $parameters$na_replace
#> [1] 0
#>
#> $parameters$members
#> [1] FALSE
#>
#> $parameters$n_cores
#> [1] 1
#>
#>
#> attr(,"class")
#> [1] "nearender_result" "list"
```

NEArender.R

The principal result matrices are:

- `n.actual` (also available as `n`): observed AGS-FGS edges;
- `n.expected`: expected edges under the degree-based null model;
- `chi`: the nonnegative chi-squared statistic;
- `z`: the signed enrichment statistic; and
- `p`: two-sided p-values reconstructed from the signed z-score; and
- `q`: FDR-adjusted probability values, computed for enrichment by convention.

`n` and `n.actual` are identical. The chi-squared statistic itself has no sign; compare `n.actual` with `n.expected`, or inspect `z`, to distinguish enrichment from depletion. BH adjustment is applied across the two-sided p-values and depleted pairs are assigned `q = 1`, preserving the package's enrichment-only q-value convention.

```r
set.heat(
  score = nea_result$scores$z,
  title = "NEA z-scores",
  Log.score = FALSE
)
```

NEArender.R

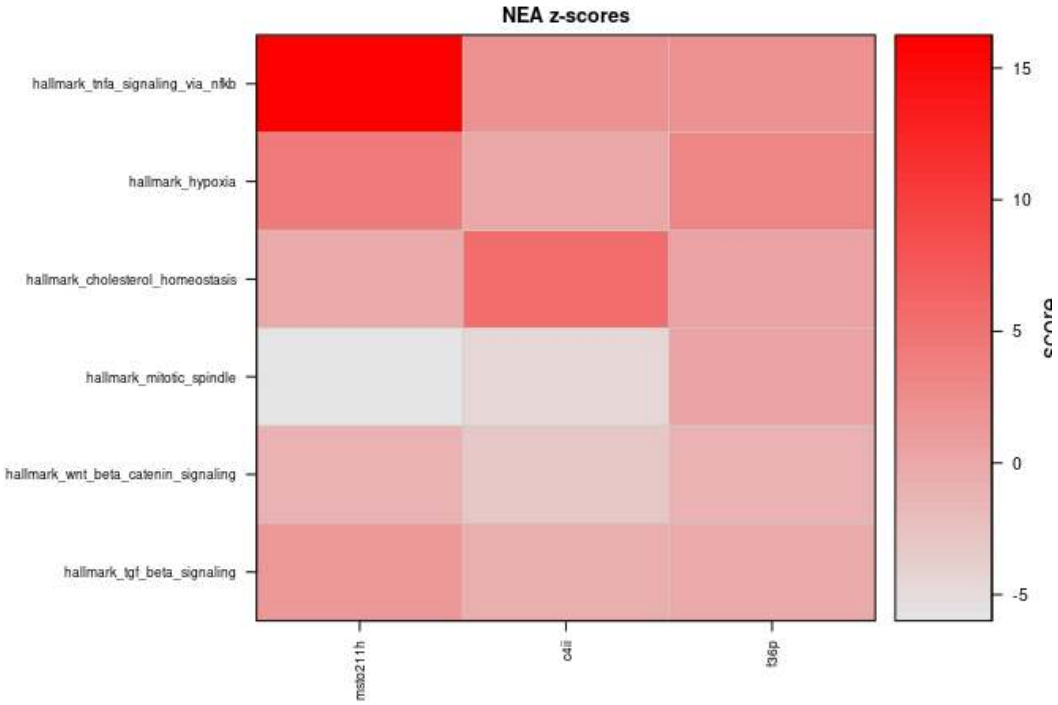

Figure 3: Matrix of NEA z-scores for the bundled "CCLE samples vs. hallmarks" example.

NEArender.R

When `members = TRUE`, the result also identifies network edges contributing to each AGS-FGS result. Member formatting can run in parallel through `n_cores`. The core sparse-matrix calculations are already vectorized and remain in one process, avoiding duplication of large sparse matrices among workers.

8.2 Over-representation analysis

`ora.render()` performs Fisher's exact test for every AGS-FGS pair. It is useful when direct gene overlap is the relevant signal and no network is required. The result includes overlap counts, odds ratios, probabilities, and adjusted probabilities. As for NEA, `n` and `n.actual` are identical fields.

```r
ora_result <- ora.render(
  ags = ags,
  fgs = fgs,
  universe = names(net$links),
  n_cores = 1,
  verbose = FALSE
)
```

```
ora_result
#> $scores
#> $scores$odds_ratio
#>                                    msto211h     c4ii      f36p
#> hallmark_tnfa_signaling_via_nfkb    7.7657373 0.000000 0.0000000
#> hallmark_hypoxia                    0.0000000 1.078907 0.9945933
#> hallmark_cholesterol_homeostasis    0.0000000 0.000000 0.0000000
#> hallmark_mitotic_spindle            0.9700852 0.000000 0.0000000
#> hallmark_wnt_beta_catenin_signaling 0.0000000 0.000000 0.0000000
#> hallmark_tgf_beta_signaling         0.0000000 0.000000 0.0000000
#>
#>
#> $score_info
#>     score_id renderer     metric input_variant     mode parameters citation
#> 1 odds_ratio      ora odds_ratio       default standard
#>
#> $statistics
#> $statistics$odds_ratio
#>                                    msto211h     c4ii      f36p
#> hallmark_tnfa_signaling_via_nfkb    7.7657373 0.000000 0.0000000
#> hallmark_hypoxia                    0.0000000 1.078907 0.9945933
#> hallmark_cholesterol_homeostasis    0.0000000 0.000000 0.0000000
#> hallmark_mitotic_spindle            0.9700852 0.000000 0.0000000
#> hallmark_wnt_beta_catenin_signaling 0.0000000 0.000000 0.0000000
#> hallmark_tgf_beta_signaling         0.0000000 0.000000 0.0000000
#>
#> $statistics$n.actual
#>                                    msto211h c4ii f36p
#> hallmark_tnfa_signaling_via_nfkb          7    0    0
#> hallmark_hypoxia                          0    1    1
#> hallmark_cholesterol_homeostasis          0    0    0
#> hallmark_mitotic_spindle                  1    0    0
#> hallmark_wnt_beta_catenin_signaling       0    0    0
#> hallmark_tgf_beta_signaling               0    0    0
#>
#> $statistics$p
#>                                       msto211h   c4ii      f36p
#> hallmark_tnfa_signaling_via_nfkb    6.372785e-05 1.0000 0.6291997
#> hallmark_hypoxia                    1.000000e+00 0.6076 1.0000000
#> hallmark_cholesterol_homeostasis    1.000000e+00 1.0000 1.0000000
#> hallmark_mitotic_spindle            1.000000e+00 1.0000 0.6296643
#> hallmark_wnt_beta_catenin_signaling 1.000000e+00 1.0000 1.0000000
#> hallmark_tgf_beta_signaling         1.000000e+00 1.0000 1.0000000
#>
#> $statistics$q
#>                                       msto211h c4ii f36p
#> hallmark_tnfa_signaling_via_nfkb    0.001147101    1    1
#> hallmark_hypoxia                    1.000000000    1    1
#> hallmark_cholesterol_homeostasis    1.000000000    1    1
#> hallmark_mitotic_spindle            1.000000000    1    1
#> hallmark_wnt_beta_catenin_signaling 1.000000000    1    1
#> hallmark_tgf_beta_signaling         1.000000000    1    1
#>
```

```
#>
#> $inputs
#> $inputs$ags_names
#> [1] "msto211h" "c4ii"     "f36p"
#>
#> $inputs$fgs_names
#> [1] "hallmark_tnfa_signaling_via_nfkb"    "hallmark_hypoxia"
#> [3] "hallmark_cholesterol_homeostasis"    "hallmark_mitotic_spindle"
#> [5] "hallmark_wnt_beta_catenin_signaling" "hallmark_tgf_beta_signaling"
#>
#> $inputs$ags_sizes
#> msto211h     c4ii     f36p
#>       89       84       91
#>
#> $inputs$fgs_sizes
#>    hallmark_tnfa_signaling_via_nfkb                   hallmark_hypoxia
#>                                 191                                188
#>    hallmark_cholesterol_homeostasis            hallmark_mitotic_spindle
#>                                  72                                197
#> hallmark_wnt_beta_catenin_signaling         hallmark_tgf_beta_signaling
#>                                  42                                 54
#>
#> $inputs$universe_size
#> [1] 17017
#>
#>
#> $diagnostics
#> $diagnostics$universe_size
#> [1] 17017
#>
#> $diagnostics$explicit_universe
#> [1] TRUE
#>
#> $diagnostics$ags_sizes
#> msto211h     c4ii     f36p
#>       89       84       91
#>
#> $diagnostics$fgs_sizes
#>    hallmark_tnfa_signaling_via_nfkb                   hallmark_hypoxia
#>                                 191                                188
#>    hallmark_cholesterol_homeostasis            hallmark_mitotic_spindle
#>                                  72                                197
#> hallmark_wnt_beta_catenin_signaling         hallmark_tgf_beta_signaling
#>                                  42                                 54
#>
#>
#> $parameters
#> $parameters$case_sensitive
#> [1] FALSE
#>
#> $parameters$n_cores
#> [1] 1
#>
```

```
#> $parameters$explicit_universe
#> [1] TRUE
#>
#>
#> attr(,"class")
#> [1] "nearender_result" "list"
```

NEArender.R

ORA rows are independent and can be distributed across workers with `n_cores > 1`. Use NEA when indirect functional relations represented by the network should contribute; use ORA when only literal set intersection should count.

8.3 Empirical significance with `ear()`

`ear()` estimates empirical significance against a restricted or otherwise biased gene universe. It repeatedly samples random AGSs of the same size as each submitted AGS and compares their NEA or ORA statistic with the observed one. This is valuable when the assayed or selectable genes are not a random sample of all genes.

```
panel <- NEArender.test$panel$Olink.Inflammation250
panel_ags <- samples2ags(
  panel,
  method = "topnorm",
  n_top = 20,
  species = "human"
)

ear_result <- ear(
  ags = panel_ags,
  fgs = NEArender.test$FGS$Hallmarks,
  net = NEArender.test$NET$STRING.300K,
  universe = rownames(panel),
  method = "nea",
  n_samples = 1000,
  n_cores = 4,
  seed = 1,
  verbose = FALSE
)
```

NEArender.R

Supply `universe` explicitly to describe the genes that could have entered an AGS. For NEA, genes in this submitted universe remain eligible even when they are absent from the network; those genes simply contribute no network edges. Use `network_nodes_only = TRUE` when controls should instead be restricted to network nodes (in which case every observed AGS member must also be a network node). The empirical probability uses a finite-sampling correction, `(n_exceed + 1) / (n_samples + 1)`, so it is never reported as zero. The returned `n_exceed` records how many random statistics met or exceeded the observed statistic. `ear()` forwards `n_cores` to both renderers: ORA can use it to distribute rows, while NEA receives it even though EAR does not request the parallel member-formatting step. Constant-size AGSs are especially efficient because a common collection of random sets can be reused across samples.

9 Other rendering methods

9.1 Random walks and PageRank

Random-walk renderers propagate information through paths of arbitrary length, whereas NEA uses direct network edges. In `rwr.render()`, a larger `restart` probability keeps propagation closer to the starting genes and emphasizes shorter paths. In `pagerank.render()`, a larger `damping` value does the opposite: it follows network edges for longer before teleporting. Their scores are relative propagation measures, not calibrated p- or q-values.

```
rwr_result <- rwr.render(
  ags = ags,
  fgs = fgs,
  net = net,
  restart = 0.75,
  verbose = FALSE
)

pagerank_result <- pagerank.render(
  ags = ags,
  fgs = fgs,
  net = net,
  damping = 0.85,
  verbose = FALSE
)
pagerank_result$scores
```

NEArender.R

9.2 Texture and spectral renderers

These methods combine a numeric gene-by-sample matrix with FGS definitions. Texture rendering summarizes the spatial arrangement of discretized node states around a pathway in the network, whereas spectral rendering measures graph contrasts.

```
texture_result <- texture.render(
  fgs = NEArender.test$FGS$Hallmarks,
  mtr = NEArender.test$mtr$CCLE,
  net = NEArender.test$NET$STRING.300K,
  features = c("asm", "con", "cor", "idm"),
  scope = "within-pathway",
  n_cores = 4
)

spectral_result <- spectral.render(
  fgs = NEArender.test$FGS$Hallmarks,
  mtr = NEArender.test$mtr$CCLE,
  net = NEArender.test$NET$STRING.300K,
  features = c("rlq", "dir"),
  scope = "boundary-expanded",
  n_cores = 4
)
```

NEArender.R

`texture.render()` supports 15 Haralick features: angular second moment or energy (`asm`), contrast (`con`), correlation (`cor`), difference entropy (`den`), difference variance (`dva`), entropy (`ent`), information measures of correlation 1 and 2 (`f12`, `f13`), inverse difference moment or homogeneity (`idm`), cluster prominence (`pro`), sum average (`sav`), sum entropy (`sen`), cluster shade (`sha`), sum variance (`sva`), and variance (`var`). The default requests all 15 explicitly. `haralick.net()` is the lower-level function for a single FGS and an already constructed gray-level co-occurrence matrix.

The texture definitions follow Haralick, Shanmugam, and Dinstein (1973). Kobayashi et al. (2015) illustrate their use for biological material, and Alexeyenko (2026) describes their use for pathway-level network representations.

The spectral features are Dirichlet sums and Rayleigh quotients. Dirichlet sums summarize contrasts between states of adjacent network nodes within a pathway; Rayleigh quotients additionally normalize those contrasts by the individual node states. A separate pathway-by-sample matrix is returned for every feature. `scope = "boundary-expanded"` is typically about ten times slower than `scope = "within-pathway"`; submit alternative scope analyses as separate calls so their cost and results remain explicit.

`spectral_eigenpairs()` exposes the corresponding structural generalized eigenproblem without requiring a sample matrix:

```
pathway_spectra <- spectral_eigenpairs(
  fgs = NEArender.test$FGS$Hallmarks,
  net = NEArender.test$NET$STRING.300K,
  scope = "within-pathway",
  n_cores = 4
)
```

NEArender.R

Here `n_cores` distributes pathways across R workers. The linked BLAS/LAPACK library used by `eigen()` may independently use all available cores, so actual CPU use can exceed `n_cores`; BLAS/LAPACK versions can also cause small numerical differences.

NEA member formatting, ORA rows, PageRank personalization vectors, texture pathways, and spectral features can use `n_cores > 1`. Sparse NEA and RWR retain their vectorized sparse-matrix implementations.

9.3 GSVA-family renderers

`gsva.render()` provides GSVA, ssGSEA, z-score, and PLAGE through the Bioconductor GSVA package. These methods do not use a network.

```
gsva_result <- gsva.render(
  fgs = NEArender.test$FGS$Hallmarks,
  mtr = NEArender.test$mtr$CCLE,
  method = "gsva"
)
```

NEArender.R

10 Edge scores, edge weights, and transition probabilities

These terms describe different stages of using a weighted network:

Term	Meaning in NEArender
Edge score	A provider-specific attribute read from or derived for a source resource, such as STRING confidence or a Pathway Commons PubMed-count-plus-0.5 value.
Edge weight	The numeric value assigned to an imported edge after selecting and, if desired, transforming the source edge score.
Transition probability	The row-normalized probability used internally by a random walk to move from one node to a neighbor.

The catalog's `source_score_column` documents which source column is used for a curated network. Curated downloads deliberately avoid exposing every provider-specific alternative or score combination as package parameters. For a custom file, inspect its provider documentation and header, select the appropriate column with `score_column` in `import_network()`, and record any transformation used to turn the source edge score into an edge weight. Random walk code then converts nonnegative edge weights to transition probabilities; a transition probability is not an additional database attribute.

11 Investigating and comparing networks

11.1 Degree distribution

`node_degrees()` plots the number of nodes at each degree on log-log axes and returns the negative slope of a fitted straight line. A roughly linear plot is a quick indication of scale-free-like structure, not proof of a power-law distribution. Formal claims require appropriate estimation, selection of the lower cutoff, and goodness-of-fit comparisons such as those described by Clauset, Shalizi, and Newman (2009).

```
degree_exponent <- node_degrees(net)
```

NEArender.R

11.2 Second-order topology

`topology_2nd()` displays a two-dimensional kernel density for the degrees at opposite endpoints of network edges. This makes hub-hub and hub-low-degree regions visible. Maslov and Sneppen (2002) attributed patterns in their biological networks by comparing them with degree-preserving randomized networks. `topology_2nd()` itself is descriptive and does not perform that null model comparison, so the same attribution should not be made from its plot alone.

```
topology_2nd(net, n = 30)
```

NEArender.R

11.3 Exact network overlap

`network_overlap()` compares exact vertex and undirected edge sets. Low exact overlap can arise because resources cover different evidence types, omit real relations, or contain false-positive edges; it is not by itself a ranking of network quality. For more than five networks, the default switches from Venn diagrams to UpSet-style displays.

```
overlap <- network_overlap(
  list(
    PathwayCommons = pwc_net,
    STRING = string_net,
    GeneMANIA = genemania_net
  )
)
overlap$edges$table
```

NEArender.R

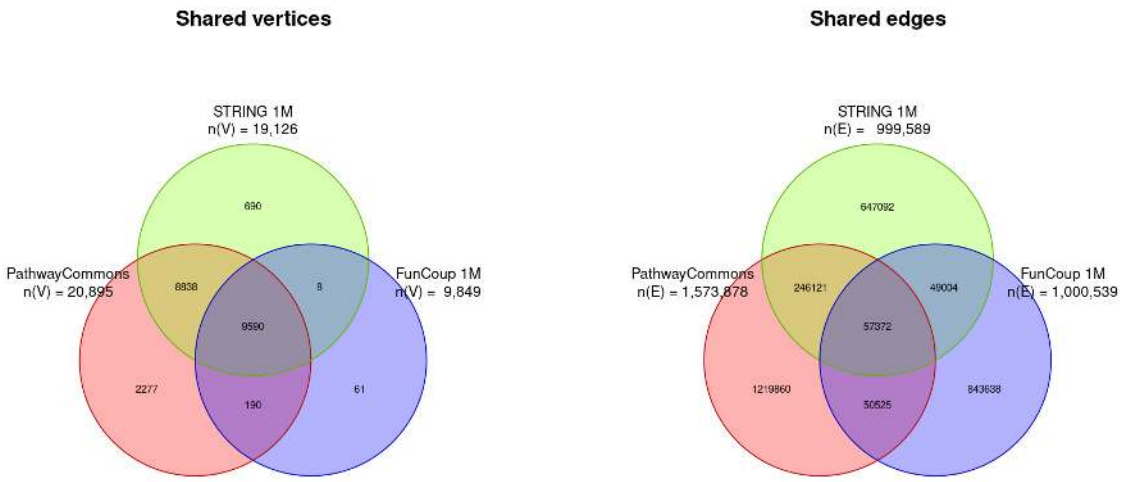

Figure 4: Shared vertex and edge overlaps between three networks.

NEArender.R

11.4 Pathway-member recovery

`nea_member_recovery_benchmark()` tests whether a network connects a known pathway member to the other members of its set. Degree-matched non-member network nodes provide negative controls by default. The returned prediction object records cumulative true-positive, false-positive, true-negative, and false-negative counts over score cutoffs. This evaluates functional recovery rather than literal overlap among source databases.

```
prediction <- nea_member_recovery_benchmark(
  net = net,
  gs = fgs,
  match_by_degree = TRUE,
  show_below_p = 0.05,
  mark_p = 0.01,
  mark_fdr = 0.1,
  seed = 1,
  verbose = FALSE
```

```
)
roc_curve <- roc(prediction, mode = "roc")
roc(prediction, mode = "prc")
roc_curve[[1]]$odds_ratio
```

NEArender.R

`universal_member_recovery_benchmark()` provides a complementary held-out design for NEA, RWR, and PageRank. It splits each pathway into training seeds and held-out members, scores network nodes against the training set, and compares held-out members with matched negative controls.

`roc()` returns normalized specificity and sensitivity in ROC mode. Its `odds_ratio` column follows Merid, Goranskaya, and Alexeyenko (2014): `sensitivity / (1 - specificity)` at each cutoff. When `mark_fdr` is not `NULL`, the ratio at that FDR cutoff is also shown next to the network name in the legend. Both sensitivity and the false-positive rate must be non-zero; otherwise OR is `NA`. With equally sized positive and negative gene sets, a higher OR at a fixed `mark_fdr` indicates a lower false-positive rate for a given sensitivity. This is not the cross-product odds ratio from a contingency table. Precision-recall mode uses recall and precision and is often informative when positives are rare.

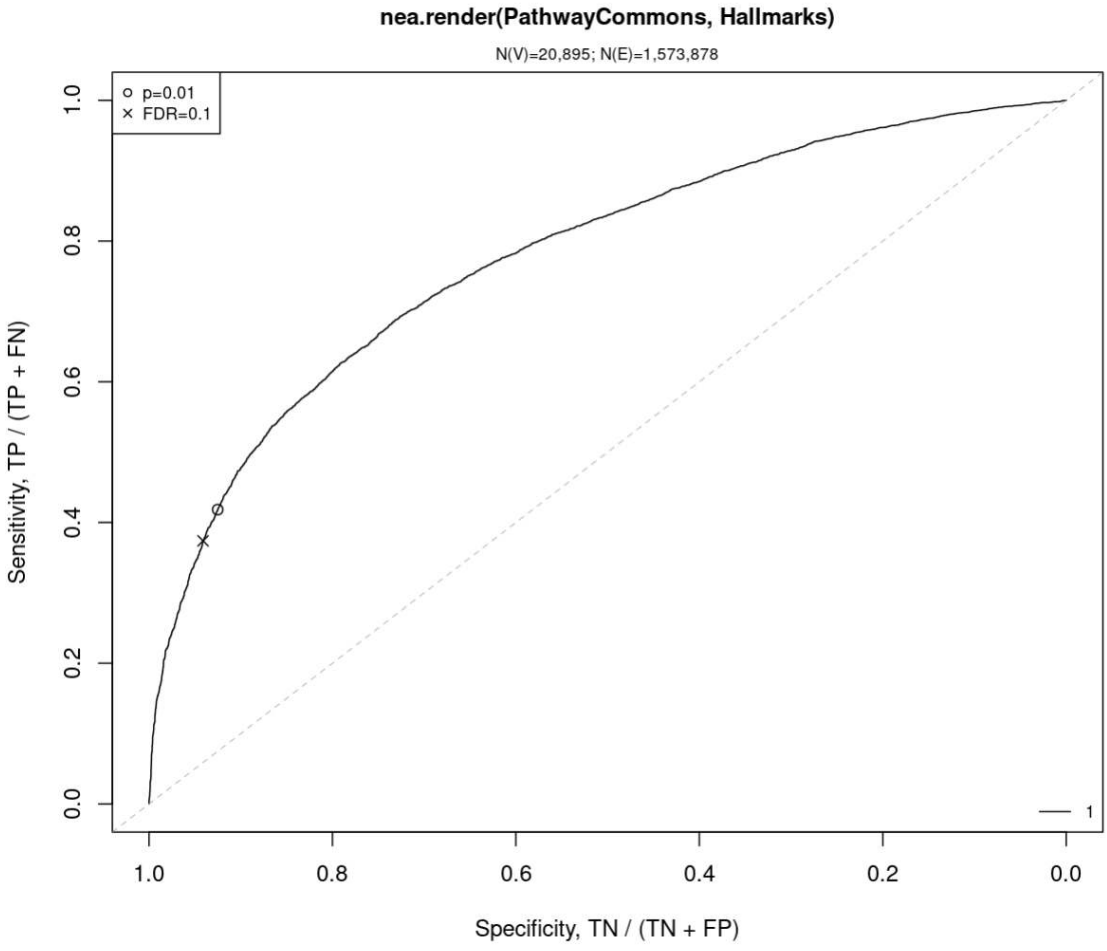

Figure 5: Hallmark pathway-member recovery benchmark.

NEArender.R

12 A smooth-start guide

This section joins the individual operations above into one end-to-end workflow. The download and full-network examples are deliberately not run during vignette construction.

Start with the bundled data and select the package species code:

```
data("NEArender.test", package = "NEArender")
spec <- "hsa"

ags_collections <- NEArender.test$AGS
fgs_collections <- NEArender.test$FGS
example_networks <- NEArender.test$NET
```

NEArender.R

Prepare identifier mappings, inspect the offline catalog, and explicitly download accepted resources:

```
cache <- file.path(tempdir(), "NEArender-cache")
prepare_id_mappings(spec, cache_dir = cache)
mappings <- load_id_mappings(spec, cache_dir = cache)

catalog <- list(
  networks = available_databases("network", species = spec),
  gene_sets = available_databases("gene_sets", species = spec)
)

for (id in catalog$networks$id) {
  download_network_database(
    id,
    cache_dir = cache,
    force = TRUE,
    accept_license = TRUE
  )
}

for (id in catalog$gene_sets$id) {
  if (!grepl("^kegg_pathways", id)) {
    download_geneset_database(
      id,
      cache_dir = cache,
      force = TRUE,
      accept_license = TRUE
    )
  }
}
```

NEArender.R

Downloaded artifacts remain in the verified cache. Load gene-set collections in a common namespace; licensed KEGG tables can be imported separately from a local subscription file.

```
fgs_resources <- list()
for (id in catalog$gene_sets$id) {
```

```
  if (!grepl("^kegg_pathways", id)) {
    fgs_resources[[id]] <- load_geneset_database(id)
  }
}

fgs_resources$KEGG_medicus <- import_kegg_pathways(
  "/path/to/KEGG_medicus.v2024",
  species = spec,
  gene_column = "symbol",
  kegg_release = "KEGG_medicus 2024"
)

fgs_resources$CCLE <- ags_collections$CCLE
fgs_resources$LUAD <- ags_collections$LUAD
```

NEArender.R

For scored resources such as STRING and GeneMANIA it is useful to compare a one-million-edge high-confidence network with a much larger version. Pathway Commons is retained at its available size.

```
network_ids <- c(
  "genemania_human_combined",
  "string12_human",
  "pathwaycommons_hgnc"
)
networks <- list()

for (id in network_ids) {
  sizes <- if (
    grepl("^pathwaycommons_", id)
  ) {
    list(Full = NULL)
  } else {
    list(`1M` = 1e6, `10M` = 1e7)
  }
  for (size_name in names(sizes)) {
    network_name <- paste(id, size_name, sep = ".")
    networks[[network_name]] <- load_network_database(
      id,
      n_top = sizes[[size_name]],
      id_mappings = mappings,
      map_to_symbols = TRUE,
      ambiguous = "drop",
      unmapped = "drop"
    )
  }
}
```

NEArender.R

A union can combine complementary resources. Provider edge scores have different meanings, so the example deliberately drops them:

```
genemania_pathwaycommons <- merge_networks(
  list(
    GeneMANIA = reduce_network(
      networks$genemania_human_combined.10M,
      select = list(n_top = 5e6)
    ),
    PathwayCommons = networks$pathwaycommons_hgnc.Full
  ),
  preserve_scores = FALSE,
  output_network_format = "igraph"
)
```

NEArender.R

Degree and second-order topology plots reveal complementary aspects of network structure. The reviewed analysis is shown in Figure 5.

```
par(mfrow = c(2, 2))
for (network_name in names(networks)) {
  node_degrees(networks[[network_name]], main = network_name)
  topology_2nd(networks[[network_name]], main = network_name)
}
```

NEArender.R

NEArender.R

Exact-overlap panels show that databases can share surprisingly few vertices or edges in absolute terms. Small apparent unique sets can arise after synonymous identifiers are collapsed.

```
network_groups <- list(
  curated_and_integrated = c(
    "pathwaycommons_hgnc.Full",
    "genemania_human_combined.1M",
    "string12_human.1M"
  ),
  integrated_1M = c(
    "genemania_human_combined.1M",
    "string12_human.1M"
  ),
  integrated_10M = c(
    "genemania_human_combined.10M",
    "string12_human.10M"
  )
)

par(mfrow = c(2, 2))
for (group_name in names(network_groups)) {
  network_overlap(
    networks[network_groups[[group_name]]],
    plot_type = "venn"
  )
}
```

NEArender.R

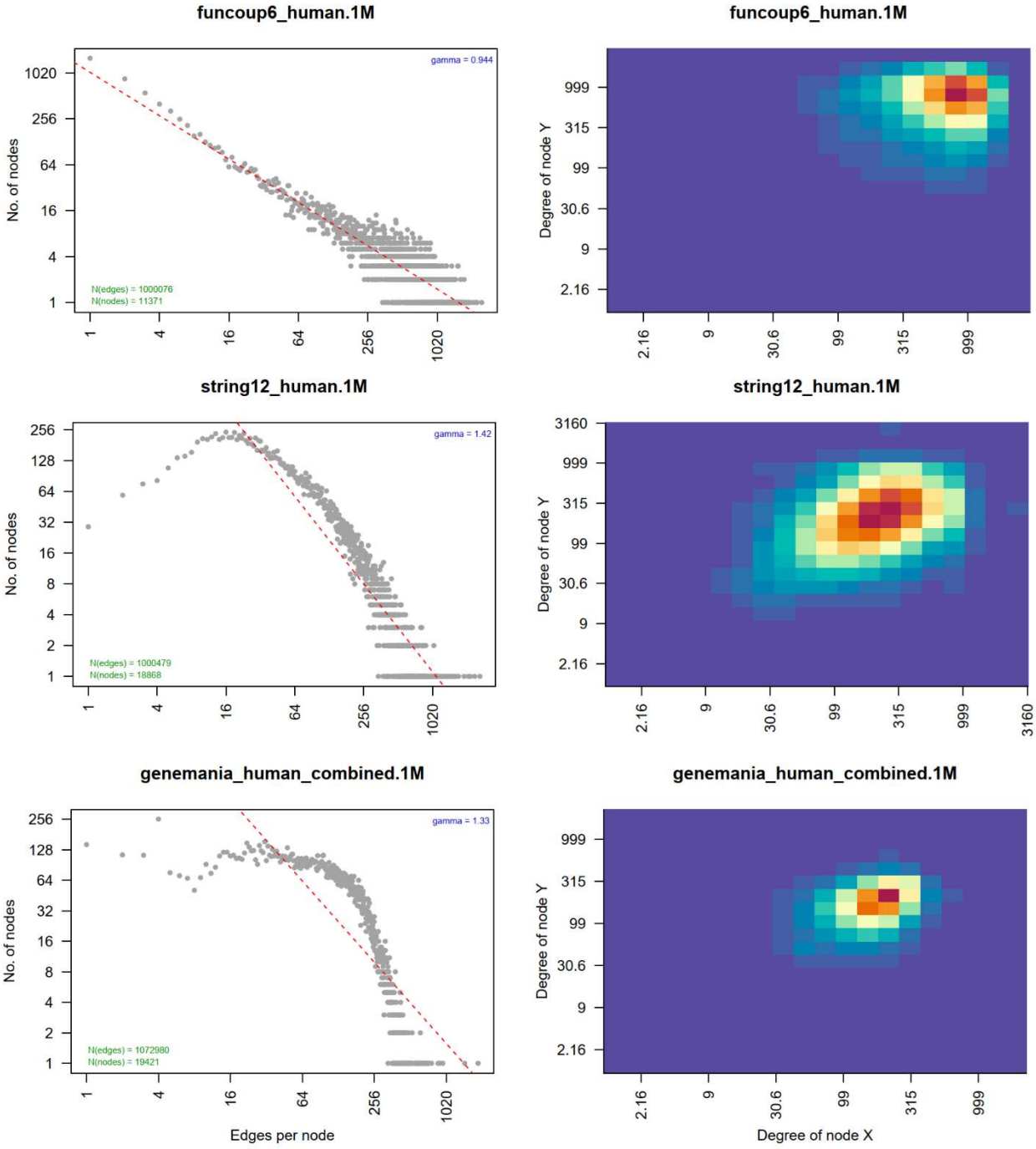

Figure 6: Degree distributions and second-order topology for three equal-size networks.

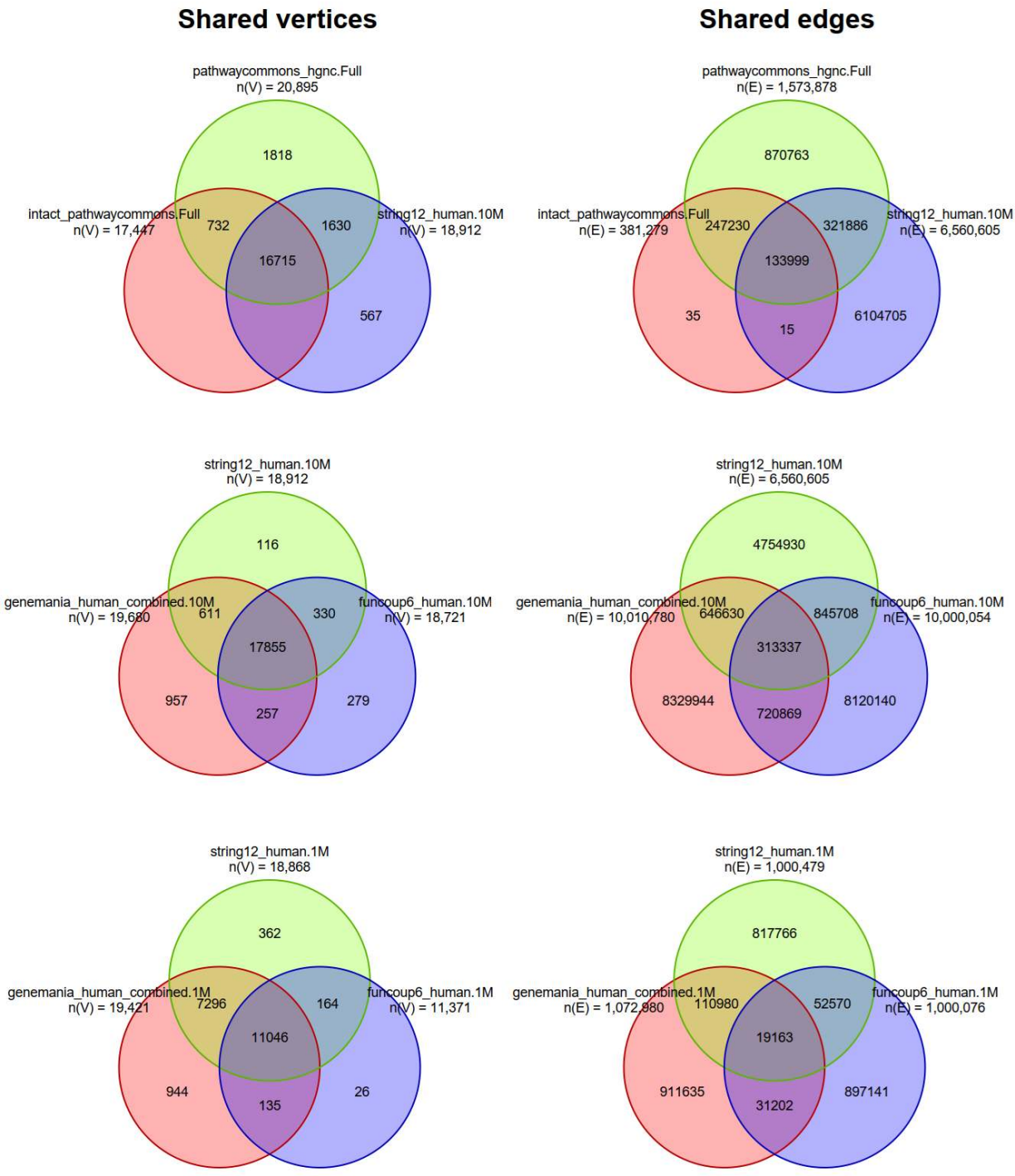

Figure 7: Network overlap panels for curated and data-integration resources.

NEArender.R

Low overlap does not by itself imply low analytical value. Member-recovery benchmarks compare networks and renderers by their ability to recover held-out functional-set members. The dedicated NEA benchmark and the universal held-out benchmark can be collected in a common nested list:

```
renderer_args <- list(
  nea = list(),
  rwr = list(restart = 0.50, normalise = "laplacian"),
  pagerank = list(damping = 0.40)
)
benchmarks <- list()

for (method in names(renderer_args)) {
  for (network_name in names(networks)) {
    network <- networks[[network_name]]
    for (collection_name in c("LUAD", "CCLE", "msigdb_h_human")) {
      sets <- fgs_resources[[collection_name]]
      if (method == "nea") {
        benchmarks$nea[[network_name]][[collection_name]] <-
          nea_member_recovery_benchmark(
            net = network,
            gs = sets,
            match_by_degree = TRUE,
            mark_p = 0.01,
            mark_fdr = 0.1,
            seed = 1
          )
      }
      benchmarks[[method]][[network_name]][[collection_name]] <-
        universal_member_recovery_benchmark(
          net = network,
          gs = sets,
          method = method,
          renderer_args = renderer_args[[method]],
          train_fraction = 2 / 3,
          min_size = 9L,
          match_by_degree = TRUE,
          seed = 1
        )
    }
  }
}
```

NEArender.R

Regroup the nested results before plotting to compare methods within a network (Figure 7) or networks within a method (Figure 8). The reviewed results favored NEA in this member-recovery task, although the random-walk parameters had not been optimized. GeneMANIA and STRING outperformed FunCoup in the analyzed settings, while all three large data-integration resources outperformed the curated resources. These observations apply to the stated releases, sizes, gene-set collections, and parameters; they are not universal database rankings.

NEArender.R

NEArender.R

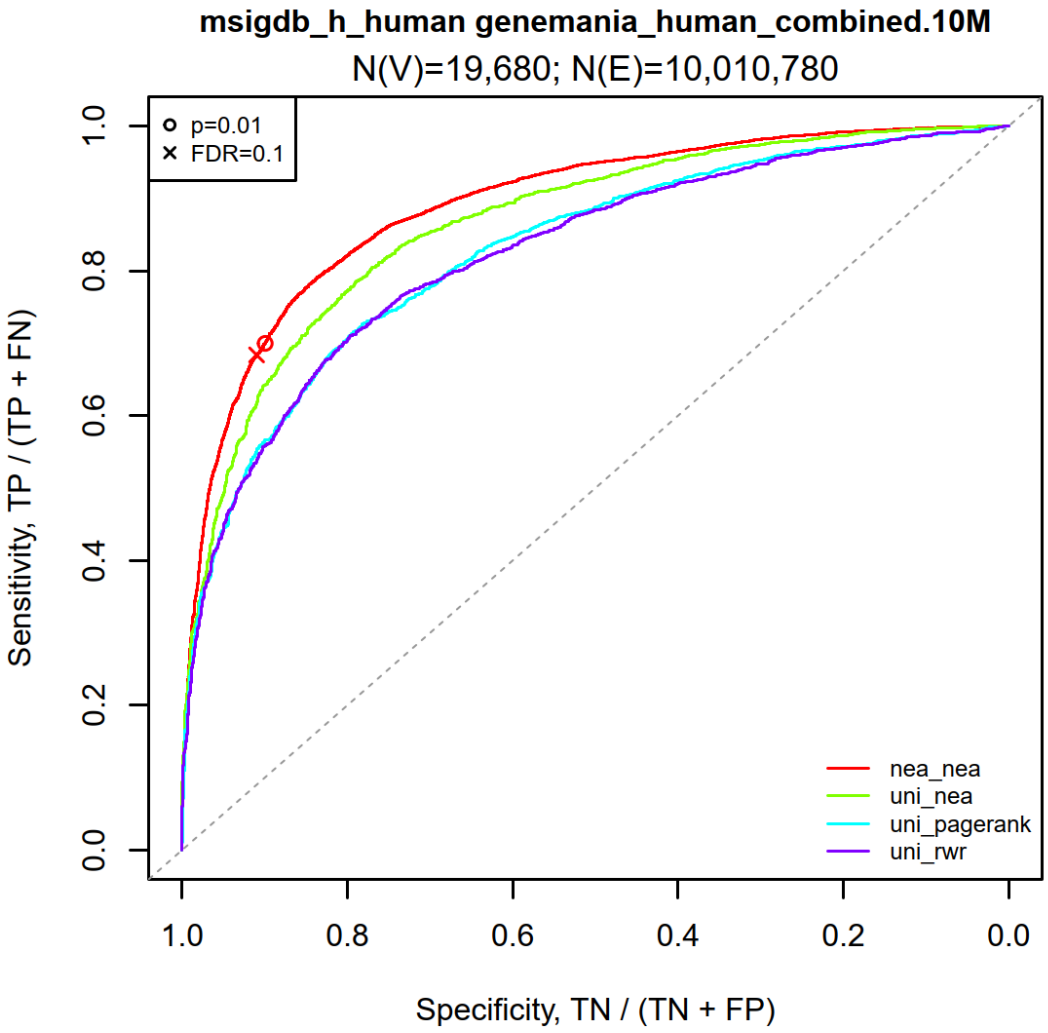

Figure 8: Pathway-member recovery benchmark of three enrichment methods.

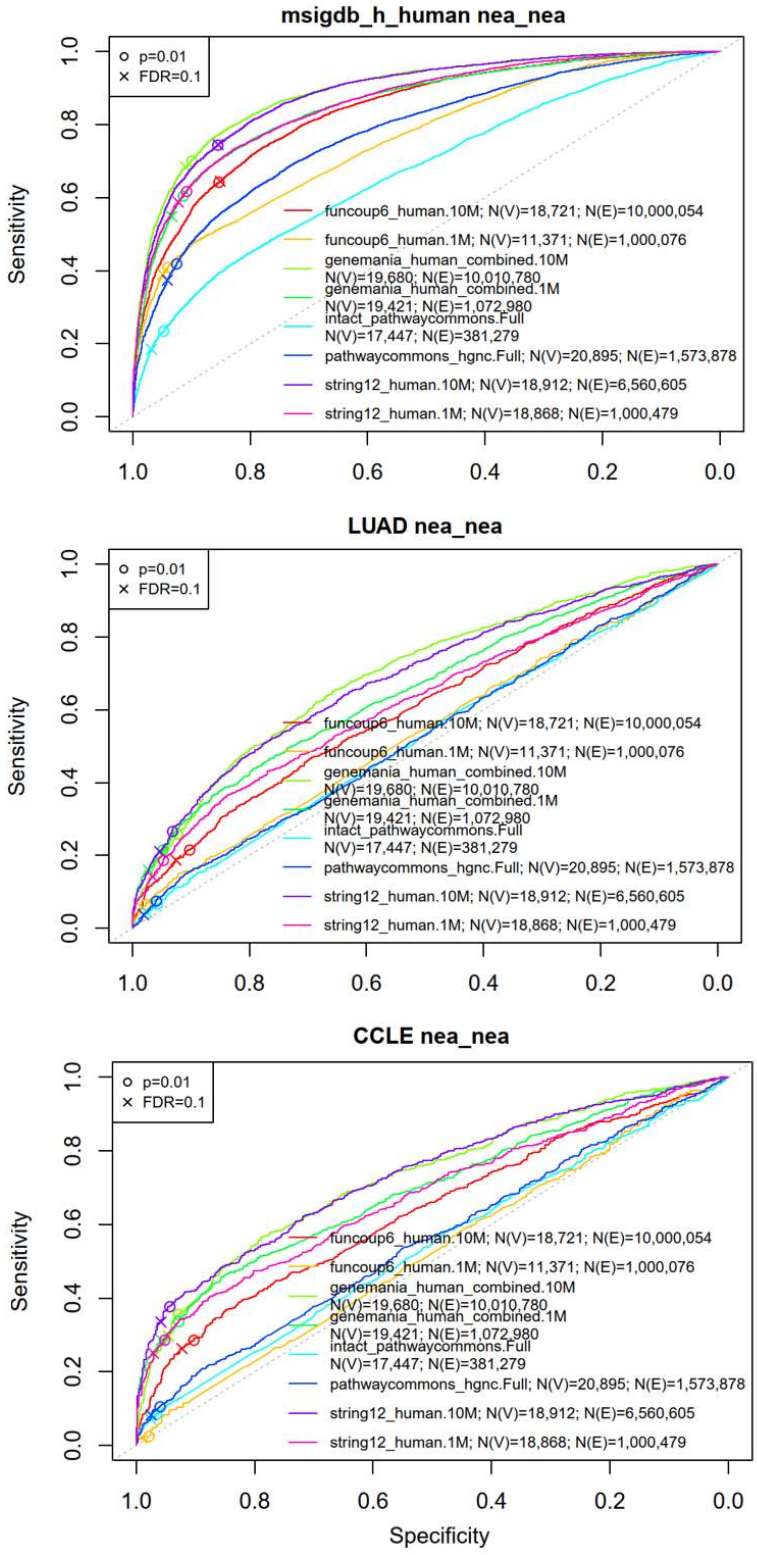

Figure 9: Pathway-member recovery compared across alternative network resources and sizes.

Point-by-point score agreement provides another view of network and renderer robustness. `run_renderer()` makes the repeated calls uniform:

```
score_results <- list()
comparison_networks <- c(
  "pathwaycommons_hgnc.Full",
  "genemania_human_combined.10M",
  "string12_human.10M"
)

for (method in names(renderer_args)) {
  for (network_name in comparison_networks) {
    result <- run_renderer(
      method = method,
      net = networks[[network_name]],
      ags = fgs_resources$msigdb_h_human,
      fgs = fgs_resources$msigdb_h_human,
      renderer_args = renderer_args[[method]]
    )
    score_results[[network_name]][[method]] <- result$scores[[1]]
  }
}
```

NEArender.R

Across network versions, direct-edge NEA preserved rank only approximately, whereas the random-walk scores were highly consistent in the high-score domain (Figure 9). Comparisons within the same network expose the extent to which the three renderers capture different topological relations (Figure 10).

NEArender.R

NEArender.R

Examples of downstream use include drug-response prediction consistent between *in vitro* and clinical settings (Franco et al. 2019), individualized cancer driver discovery (Merid et al. 2014; Petrov and Alexeyenko 2022), pathway signatures of osimertinib resistance without significant differential expression (Alexeyenko et al. 2022), EAR calibration for small proteomics panels (Brink et al. 2019), and textural-feature drug-response prediction (Alexeyenko 2026).

13 Reproducible analysis checklist

For an analysis that can be reconstructed later:

1. Record the species and submitted identifier format for every user dataset.
2. Record database keys, releases, checksums, and license acceptance reported by the downloaded resource metadata.
3. Preserve AGS preparation method and thresholds, including `n_top` or `cutoff_q`.
4. Record any custom `score_column` and transformation from source edge score to edge weight.
5. Set and record random seeds for EAR, randomized AGS preparation, and member recovery controls.
6. Save the package version and `sessionInfo()`.

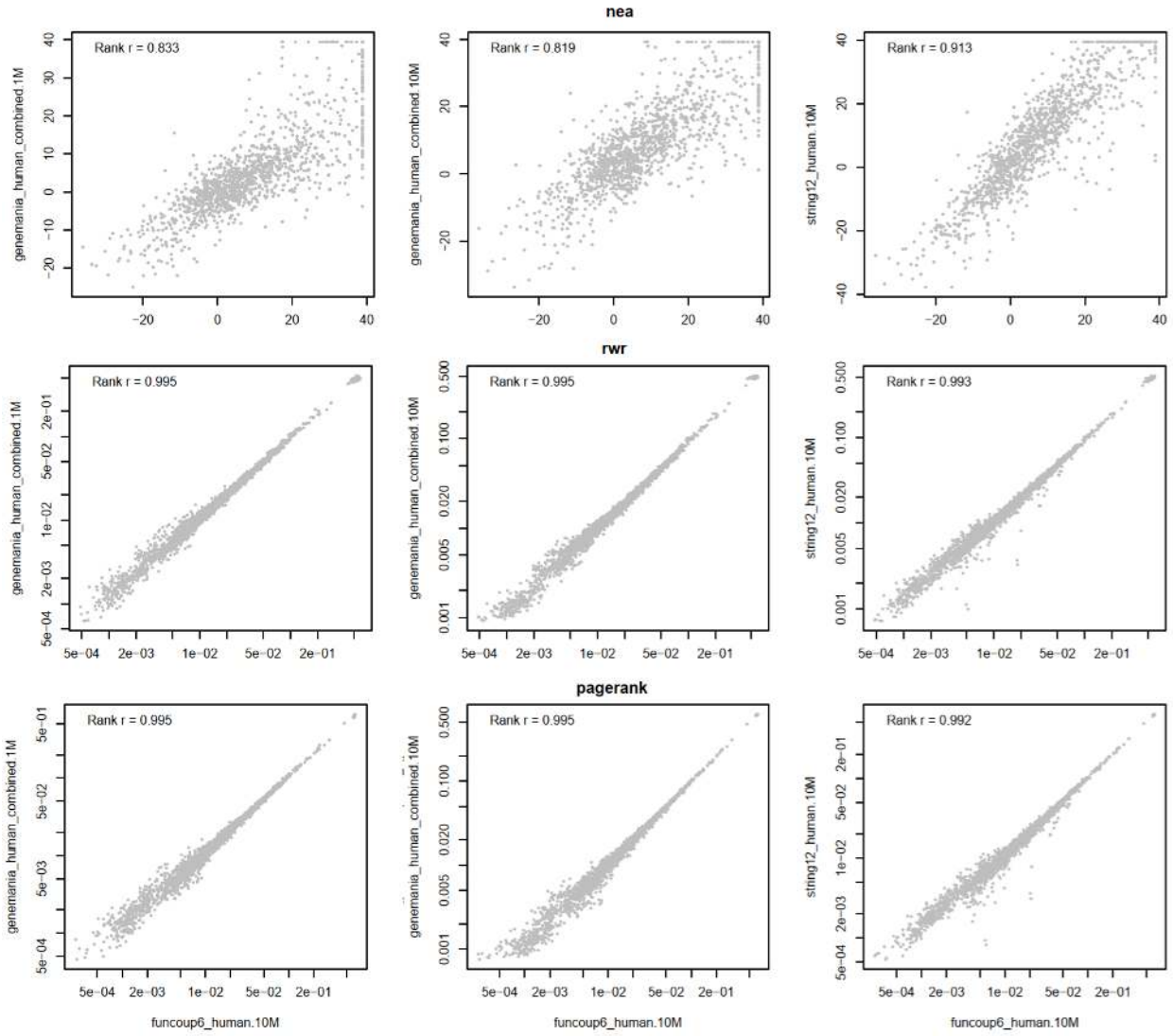

Figure 10: Point-by-point score agreement between network versions using three renderers.

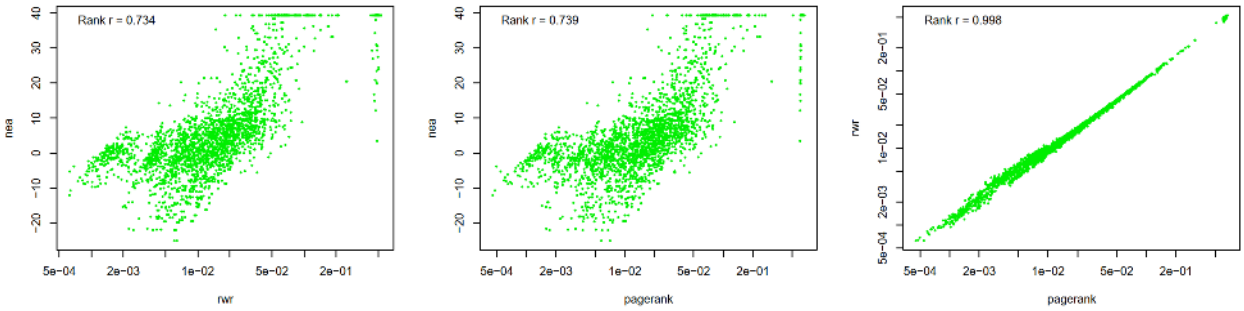

Figure 11: Point-by-point score agreement between renderers on same network.

14 References

Alexeyenko A, Lee W, Pernemalm M, et al. (2012). Network enrichment analysis: extension of gene-set enrichment analysis to gene networks. *BMC Bioinformatics*, 13:226. doi:10.1186/1471-2105-13-226.

Alexeyenko A, Brustugun OT, Eide IJZ, et al. (2022). Plasma RNA profiling unveils transcriptional signatures associated with resistance to osimertinib in EGFR T790M positive non-small cell lung cancer patients. *Translational Lung Cancer Research*, 11(10):2064-2078. doi:10.21037/tlcr-22-236.

Alexeyenko A (2026). Textural features for pathway-level representation of omics data in biological networks. *bioRxiv*. doi:10.64898/2026.07.12.737672.

Brink M, Lundquist A, Alexeyenko A, Lejon K, Rantapää-Dahlqvist S (2019). Protein profiling and network enrichment analysis in individuals before and after the onset of rheumatoid arthritis. *Arthritis Research & Therapy*, 21(1):288. doi:10.1186/s13075-019-2066-9.

Chung FRK (1997). *Spectral Graph Theory*. American Mathematical Society. ISBN 978-0-8218-0315-8.

Clauset A, Shalizi CR, Newman MEJ (2009). Power-law distributions in empirical data. *SIAM Review*, 51:661-703. doi:10.1137/070710111.

Fang H, Gough J (2014). The `dnet` approach promotes emerging research on cancer patient survival. *Genome Medicine*, 6:64.

Franco M, Jeggari A, Peuget S, Böttger F, Selivanova G, Alexeyenko A (2019). Prediction of response to anti-cancer drugs becomes robust via network integration of molecular data. *Scientific Reports*, 9(1):2379. doi:10.1038/s41598-019-39019-2.

Haenzelmann S, Castelo R, Guinney J (2013). GSVA: gene set variation analysis for microarray and RNA-seq data. *BMC Bioinformatics*, 14:7.

Haralick RM, Shanmugam K, Dinstein I (1973). Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3:610-621. doi:10.1109/TSMC.1973.4309314.

Jeggari A, Alexeyenko A (2017). NEArender: an R package for functional interpretation of 'omics' data via network enrichment analysis. *BMC Bioinformatics*, 18:118.

Kobayashi K, Akada M, Torigoe T, Imazu S, Sugiyama J (2015). Automated recognition of wood used in traditional Japanese sculptures by texture analysis of their low-resolution computed tomography data. *Journal of Wood Science*, 61:630-640. doi:10.1007/s10086-015-1507-6.

Maslov S, Sneppen K (2002). Specificity and stability in topology of protein networks. *Science*, 296:910-913. doi:10.1126/science.1065103.

Merid SK, Goranskaya D, Alexeyenko A (2014). Distinguishing between driver and passenger mutations in individual cancer genomes by network enrichment analysis. *BMC Bioinformatics*, 15:308. doi:10.1186/1471-2105-15-308.

Petrov I, Alexeyenko A (2022). Individualized discovery of rare cancer drivers in global network context. *eLife*, 11:e74010. doi:10.7554/eLife.74010.