

# Package ‘htna’

July 6, 2026

**Title** Heterogeneous Transition Network Analysis

**Version** 0.1.0

**Description** Implements the Heterogeneous Transition Network Analysis (HTNA) method described by López-Pernas et al. (2026) <[doi:10.1002/jcal.70285](https://doi.org/10.1002/jcal.70285)>. The method is an extension of transition network analysis (TNA) where actions or events belong to two or more distinct actor types (e.g. Human and AI), preserving the actor type partition on the resulting network. Provides a thin, focused API on top of the 'Nestimate' estimation engine and the 'cograph' rendering engine, so downstream bootstrap, permutation, reliability, centrality, and plotting functions treat each actor's codes as a distinct node group.

**License** MIT + file LICENSE

**URL** <https://sonsoles.me/htna/>

**BugReports** <https://github.com/sonsoleslp/htna/issues>

**Depends** R (>= 4.1.0)

**Imports** Nestimate, cograph, igraph

**Suggests** codyna, ggplot2, janitor, knitr, rmarkdown, gridExtra, testthat (>= 3.0.0), tna

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Encoding** UTF-8

**Language** en-US

**RoxygenNote** 7.3.3

**LazyData** true

**NeedsCompilation** no

**Author** Sonsoles López-Pernas [aut, cre, cph],  
Kamila Misiejuk [aut, cph],  
Mohammed Saqr [aut, cph]

**Maintainer** Sonsoles López-Pernas <[sonsoles.lopez@uef.fi](mailto:sonsoles.lopez@uef.fi)>

**Repository** CRAN

**Date/Publication** 2026-07-06 14:00:02 UTC

## Contents

ai_simplified	2
association_rules_htna	3
bootstrap	4
bootstrap_htna	5
build_htna	6
casedrop_reliability_htna	8
centralities_htna	10
centrality_stability_htna	11
edge_betweenness_htna	13
extract_meta_paths	14
frequencies_htna	16
htna_palette	17
human_ai	17
human_ai_codebook	19
human_simplified	19
markov_order_test_htna	20
mosaic_plot_htna	21
permutation_htna	22
plot.htna_stability	24
plot.htna_stability_group	25
plot_centralities	25
plot_edge_betweenness_htna	27
plot_frequencies_htna	28
plot_htna	29
plot_htna_bootstrap	30
plot_htna_diff	31
plot_sequences	32
print.htna_stability	33
print.htna_stability_group	34
reliability_htna	34
sequence_compare_htna	35
sequence_plot_htna	37
state_distribution_htna	38
state_frequencies_htna	39
summary.htna	40
<b>Index</b>	<b>42</b>

---

 ai\_simplified

*Simplified AI Interaction Sequences (per-actor frame)*


---

### Description

The AI-only slice of `human_ai`, in a long-format data frame ready to feed the named-list form of `build_htna()`. Codes have already been collapsed into the simplified AI alphabet; see `human_ai` for the remapping rules.

**Usage**

```
ai_simplified
```

**Format**

A data frame with 8551 rows and 10 columns. Schema matches [human\\_ai](#) minus the phase column; every value in actor\_type is "AI".

**Source**

Derived from `Nestimate::ai_long`; see `data-raw/human_ai.R` for the build script.

**See Also**

[human\\_ai](#), [human\\_simplified](#), [human\\_ai\\_codebook](#).

**Examples**

```
data(human_simplified, ai_simplified)
net <- build_htna(list(Human = human_simplified, AI = ai_simplified))
plot_htna(net)
```

---

association\_rules\_htna

*Association Rule Mining over Transitions*

---

**Description**

htna-named alias of `Nestimate::association_rules()`. Mines frequent itemsets and association rules over a transition-count matrix under support / confidence / lift thresholds.

**Usage**

```
association_rules_htna(  
  x,  
  min_support = 0.1,  
  min_confidence = 0.5,  
  min_lift = 1,  
  max_length = 5L  
)
```

**Arguments**

x	Input data. Accepts: <b>netobject</b> Uses <code>\$data</code> sequences — each sequence becomes a transaction of its unique states. <b>list</b> Each element is a character vector of items (one transaction). <b>data.frame</b> Wide format: each row is a transaction, character columns are item occurrences. Or a binary matrix (0/1). <b>matrix</b> Binary transaction matrix (rows = transactions, columns = items).
min_support	Numeric. Minimum support threshold. Default: 0.1.
min_confidence	Numeric. Minimum confidence threshold. Default: 0.5.
min_lift	Numeric. Minimum lift threshold. Default: 1.0.
max_length	Integer. Maximum itemset size. Default: 5.

**Details**

Foundation function in the htna-aware exploratory family. Works on transition-count input produced by `frequencies_htna()` or `Nestimate::build_network()`; the rules carry over to htna networks since the underlying transition counts are the same.

Suffixed `_htna` to avoid clashing with `Nestimate::association_rules()` when both packages are loaded.

**Value**

A list with the discovered rules and frequent itemsets. See `Nestimate::association_rules()` for details.

**See Also**

`frequencies_htna()`, `mosaic_plot_htna()`.

**Examples**

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
association_rules_htna(net, max_length = 3L)
```

---

bootstrap

*Bootstrap generic*


---

**Description**

S3 generic dispatched on the class of `x`. Provided so `bootstrap(net)` works directly on an htna network (see `bootstrap_htna()`).

**Usage**

```
bootstrap(x, ...)
```

**Arguments**

x                    An object to bootstrap.  
 ...                  Arguments forwarded to the method.

**Value**

An object whose structure is method-defined.

**Examples**

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
bootstrap(net, iter = 50)
```

---

 bootstrap\_htna

*Bootstrap an HTNA Network*


---

**Description**

Thin wrapper around [Nestimate::bootstrap\\_network\(\)](#) that runs the standard state-level edge bootstrap on an htna network and tags the result so it is identifiable as an htna bootstrap. The bootstrap inference itself is Nestimate's: per-edge mean, sd, p-value, significance, and confidence / credibility intervals across iter resamples. State-level granularity is the correct level for edge-stability analysis - aggregating up to actor pairs would smear over real edge-by-edge differences.

**Usage**

```
bootstrap_htna(x, ...)

## S3 method for class 'htna'
bootstrap(x, ...)
```

**Arguments**

x                    [htna]  
                     A network built with [build\\_htna\(\)](#).  
 ...                  Arguments forwarded to [Nestimate::bootstrap\\_network\(\)](#) (e.g. iter, ci\_level, consistency\_range, seed).

## Details

What this wrapper adds is identity and downstream class continuity:

- The returned object has class "htna\_bootstrap" ahead of "net\_bootstrap", so inherits(boot, "htna\_bootstrap") works.
- The actor partition (\$nodes\$groups, \$node\_groups) is restored on boot\$model from the input network so cograph's auto-detect still recognizes the heterogeneous schema downstream.
- boot\$model is re-tagged with "htna" at the front of its class chain so the chain is c("htna", "netobject", "cograph\_network") - identical to what `build_htna()` returns.

## Value

An object of class c("htna\_bootstrap", "net\_bootstrap"). All slots produced by `Nestimate::bootstrap_network()` are preserved verbatim (\$summary, \$mean, \$sd, \$p\_values, \$significant, \$ci\_lower/\$ci\_upper, \$scr\_lower/\$scr\_upper, \$model, \$original, etc.).

## See Also

`Nestimate::bootstrap_network()` for the underlying algorithm and slot documentation. `build_htna()` for constructing the input.

## Examples

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
boot <- bootstrap_htna(net, iter = 50)
inherits(boot, "htna_bootstrap")           # TRUE
inherits(boot$model, "htna")               # TRUE
head(boot$summary)                         # state-level edge stability
```

---

build\_htna

*Build a Heterogeneous Transition Network (HTNA)*

---

## Description

Builds a transition network over a combined sequence of two or more actor groups (e.g. Human and AI) and preserves the actor partition on the result so downstream plotting and analysis can treat each actor's codes as a distinct node group.

## Usage

```
build_htna(
  data,
  actor_type = NULL,
  actor = NULL,
```

```

    node_groups = NULL,
    action = "code",
    session = "session_id",
    order = "order_in_session",
    method = "relative",
    group = NULL,
    disambiguate = FALSE,
    ...
)

```

## Arguments

data	<p>Either:</p> <ul style="list-style-type: none"> <li>• A named list of long-format data frames, one per actor type (e.g. <code>list(Human = human_simplified, AI = ai_simplified)</code>). All frames must share the same column schema.</li> <li>• A single long-format data frame. In that case either <code>actor_type</code> (row-level actor-type IDs) or <code>node_groups</code> (node-level actor-type lookup) must be supplied.</li> </ul>
actor_type	Character. Name of the column tagging each row's actor type / group (e.g. "Human" vs "AI") when data is a single data frame. Ignored when data is a named list or when <code>node_groups</code> is supplied.
actor	Character. Name of an optional column identifying the individual actor that performed each event (e.g. a learner / user id). Forwarded to <code>build_network</code> with the same semantics; orthogonal to <code>actor_type</code> , which encodes the group/type partition over codes.
node_groups	<p>Node-to-actor-type lookup, in either of two forms:</p> <ul style="list-style-type: none"> <li>• A <b>named list</b> mapping actor-type labels to character vectors of code names, e.g. <code>list(Human = c("Specify", "Command"), AI = c("Plan", "Execute"))</code>.</li> <li>• A <b>2-column data frame</b> with one column named after action (the codes) and one other column tagging each code with its actor type, e.g. <code>data.frame(code = c("Specify", "Plan"), actor_type = c("Human", "AI"))</code>. The actor-type ordering follows the column's factor levels or, for character vectors, the order of first appearance.</li> </ul> <p>Use <code>node_groups</code> when data is a single long-format frame with no actor-type column and you want to declare the node-to-type partition directly. Each code in <code>data[[action]]</code> must appear in exactly one entry. Mutually exclusive with <code>actor_type</code>.</p>
action	Character. Name of the action/code column. Default "code".
session	Character. Name of the session column. Default "session_id".
order	Character. Name of the within-session order column. Default "order_in_session".
method	Character. Transition method passed to <code>build_network</code> : "relative" (default), "frequency", or "attention".
group	Character. Optional name of a column in data used to split sessions into cohorts (e.g. "cluster"). When supplied, one network is built per group level

and the result is a named list of htna networks with class `c("htna_group", "netobject_group")`.

`disambiguate` Logical. If FALSE (default), the function errors when a code label appears in more than one actor-type group. If TRUE, codes are prefixed with the actor-type label ("Human:Ask", "AI:Ask") so they become distinct nodes.

... Additional arguments forwarded to `build_network`.

### Value

A `netobject` with the actor partition stored in `cograph`'s canonical schema, so `cograph::plot_htna(net)` auto-detects the groups with no further arguments:

- `$nodes$groups` - actor label per node (the column name `cograph::plot_htna` auto-detects).
- `$node_groups` - data frame with columns `node` and `group` (canonical `cograph_network` schema, also readable by `cograph::get_groups`, `cluster_summary`, and the print method for `cograph_network`).

All other slots are exactly as returned by `build_network`.

### See Also

[build\\_network](#), [build\\_tna](#), [plot\\_htna](#)

### Examples

```
data(human_ai, human_simplified, ai_simplified)

# Form 1: named list of per-actor frames
net <- build_htna(list(Human = human_simplified, AI = ai_simplified))
head(net$node_groups)

# Form 2: single combined frame with a row-level actor-type tag
net <- build_htna(human_ai, actor_type = "actor_type")
```

---

`casedrop_reliability_htna`

*Edge-Weight Case-Dropping Stability*

---

### Description

`htna`-named alias of `Nestimate::casedrop_reliability()`. Computes the CS-coefficient for the edge-weight vector of a network: the maximum proportion of cases (rows of `x$data`) that can be dropped while the flattened edge-weight vector of the re-estimated network still correlates with the original above threshold in at least certainty of iterations.

**Usage**

```

casedrop_reliability_htna(
  x,
  iter = 1000L,
  drop_prop = seq(0.1, 0.9, by = 0.1),
  threshold = 0.7,
  certainty = 0.95,
  method = c("spearman", "pearson", "kendall"),
  include_diag = FALSE,
  seed = NULL
)

```

**Arguments**

x	A <code>net_casedrop_reliability_group</code> object.
iter	Integer. Iterations per drop proportion. Default 1000.
drop_prop	Drop proportion at which to report the four metrics (mean +/- sd per network). Must be one of the drop proportions the object was built with. Defaults to the object's median grid value (the stored grid is used, not an assumed 0.7); pass an explicit value not in the grid to get an error listing the available proportions.
threshold	Numeric in $[0, 1]$ . Minimum edge-vector correlation for an iteration to count as stable. Default 0.7.
certainty	Numeric in $[0, 1]$ . Required fraction of iterations whose correlation must exceed threshold for a drop proportion to qualify. Default 0.95.
method	Correlation method: "pearson" (weight magnitudes), "spearman" (ranks, robust to scale), or "kendall". Default "spearman" because edge weights often span several orders of magnitude and rank stability is the typical target.
include_diag	Logical. Include diagonal (self-loop) edges in the edge vector. Default FALSE.
seed	Optional integer for reproducibility.

**Details**

Works on htna networks and grouped htna networks directly.

Suffixed `_htna` to avoid clashing with `Nestimate::casedrop_reliability()` when both packages are loaded.

**Value**

An object of class `net_casedrop_reliability` (single network) or `net_casedrop_reliability_group` (grouped htna). See [Nestimate::casedrop\\_reliability\(\)](#) for the full component list and the corresponding `plot()` method.

**See Also**

[reliability\\_htna\(\)](#), [centrality\\_stability\\_htna\(\)](#), [bootstrap\\_htna\(\)](#).

## Examples

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
casedrop_reliability_htna(net, iter = 20, seed = 1)
```

---

centralities\_htna      *Compute Centrality Measures for an htna Network*

---

## Description

Computes a fixed panel of centrality measures via `cograph::cograph` and returns them as a tidy data frame: one row per node, one column per measure (plus node, an actor column when the htna partition is set, and a group column when the input is an htna\_group).

## Usage

```
centralities_htna(
  x,
  measures = c("OutStrength", "InStrength", "ClosenessIn", "ClosenessOut", "Closeness",
    "Betweenness", "BetweennessRSP", "Diffusion", "Clustering"),
  ...
)
```

## Arguments

<code>x</code>	An htna network from <code>build_htna()</code> or an htna_group.
<code>measures</code>	Character vector of measure names. Defaults to all nine.
<code>...</code>	Forwarded to the underlying cograph centrality functions.

## Details

The measures are: OutStrength, InStrength, ClosenessIn, ClosenessOut, Closeness, Betweenness, BetweennessRSP, Diffusion, Clustering. Path-based measures (closeness and betweenness variants) are computed with `invert_weights = TRUE`, since in transition networks larger weight = stronger link = shorter distance. Strength and the closed-form measures (Diffusion, Clustering) use raw weights.

Mapping from measure name to cograph implementation:

- OutStrength → `cograph::centrality_outstrength()`
- InStrength → `cograph::centrality_instrength()`
- ClosenessIn → `cograph::centrality_incloseness()` (inverted)
- ClosenessOut → `cograph::centrality_outcloseness()` (inverted)
- Closeness → `cograph::centrality_closeness()` (inverted)
- Betweenness → `cograph::centrality_betweenness()` (inverted)

- BetweennessRSP → `cograph::centrality_current_flow_betweenness()`
- Diffusion → `cograph::centrality_diffusion()`
- Clustering → `cograph::centrality_transitivity()`

### Value

A data frame with one row per node.

### Examples

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
centralities_htna(net)
```

---

centrality\_stability\_htna

*Centrality Stability for an htna Network*

---

### Description

Thin wrapper around `Nestimate::centrality_stability()` that validates the input is an htna network (or htna\_group), forwards the call, and tags the result with an htna\_stability class so callers can dispatch on the htna form.

### Usage

```
centrality_stability_htna(
  x,
  measures = c("InStrength", "OutStrength", "Betweenness"),
  iter = 1000L,
  drop_prop = seq(0.1, 0.9, by = 0.1),
  threshold = 0.7,
  certainty = 0.95,
  method = "pearson",
  centrality_fn = NULL,
  loops = FALSE,
  seed = NULL
)
```

### Arguments

`x` An htna network from `build_htna()` or an htna\_group.

measures	Character vector of centrality measures to assess. Default <code>c("InStrength", "OutStrength", "Betweenness")</code> — the three measures whose values are bit-equal between htna's cograph engine and Nestimate's default centrality implementation. To assess htna's nine measures (including the three that differ between engines: <code>BetweennessRSP</code> , <code>Diffusion</code> , <code>Clustering</code> ), pass them explicitly together with a custom <code>centrality_fn</code> .
iter	Integer. Number of resamples per drop proportion. Default 1000.
drop_prop	Numeric vector. Proportions of sessions to drop. Default <code>seq(0.1, 0.9, by = 0.1)</code> .
threshold	Numeric in [0, 1]. Correlation threshold for the case-dropping stability coefficient. Default 0.7.
certainty	Numeric in [0, 1]. Probability of meeting the threshold required for a drop proportion to count as stable. Default 0.95.
method	Correlation method, one of "pearson", "spearman", "kendall". Default "pearson".
centrality_fn	Optional function to compute centralities. Default NULL (use Nestimate's internal implementation). See <code>Nestimate::centrality_stability()</code> for the expected signature.
loops	Logical. Whether to retain self-loops. Default FALSE.
seed	Optional integer seed for reproducibility. Default NULL (no seed reset).

### Details

For an `htna_group`, the call is iterated per cohort and the result is a named list (one `htna_stability` per cohort), with class `c("htna_stability_group", "list")`.

### Value

For a single htna network, an object of class `c("htna_stability", "net_stability")` with the same components as `Nestimate::centrality_stability()`: `cs` (the stability coefficients, one per measure), `correlations` (per-drop-proportion correlation traces), and the parameters that were used. For an `htna_group`, a named list of such objects with class `c("htna_stability_group", "list")`.

### See Also

`Nestimate::centrality_stability()`, `bootstrap_htna()`, `reliability_htna()`.

### Examples

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
cs <- centrality_stability_htna(net, iter = 30, seed = 1)
cs$cs
```

---

*edge\_betweenness\_htna Edge Betweenness Network*

---

**Description**

Computes the edge betweenness of every existing edge in an htna network and returns a copy whose `$weights` slot stores those betweenness scores instead of the original transition weights. The actor partition is preserved, so the result can be plotted with `plot_htna()` to visualise which edges carry the most shortest-path traffic.

**Usage**

```
edge_betweenness_htna(x, directed = TRUE, invert = TRUE)
```

**Arguments**

<code>x</code>	An htna network from <code>build_htna()</code> .
<code>directed</code>	If TRUE (default), shortest paths follow edge direction.
<code>invert</code>	If TRUE (default), use $1 / \text{weight}$ as the edge cost when computing shortest paths.

**Details**

Mirrors `tna::betweenness_network()`: edge weights are inverted to costs by default (`invert = TRUE`) – in transition networks a larger transition probability means the edge is "easier" and so the equivalent path cost is smaller.

**Value**

A copy of `x` whose `$weights` matrix entries are edge-betweenness scores at every position where the original network had a non-zero transition. Class is `c("htna_edge_betweenness", class(x))`.

**See Also**

`centralities_htna()` for node-level centrality measures, `tna::betweenness_network()` for the tna equivalent.

**Examples**

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
eb <- edge_betweenness_htna(net)
plot_htna(eb) # edge thickness = betweenness score
```

---

extract\_meta\_paths      *Extract Path Patterns from a Heterogeneous Transition Network*

---

### Description

Discovers recurring patterns in the actor-typed sequences that produced a heterogeneous transition network. Operates at two levels:

### Usage

```
extract_meta_paths(
  x,
  level = c("state", "type"),
  length = 2:4,
  schema = NULL,
  type = c("contiguous", "gapped"),
  gap = 1L,
  min_count = 1L,
  min_support = 0,
  min_lift = 0,
  start = NULL,
  end = NULL,
  contain = NULL
)
```

### Arguments

x	[htna_network] A network built with <code>build_htna()</code> . Must have <code>\$nodes\$groups</code> , <code>\$node_groups</code> , and <code>\$data</code> populated.
level	[character(1): "state"] "state" returns concrete state-level patterns with a <code>meta_schema</code> rollup column. "type" returns the type-level meta-path summary.
length	[integer(): 2:4] Pattern lengths to enumerate. Ignored when schema is supplied.
schema	[character(1)] Optional. A path template written as elements separated by "->". Each element can be a type name, a concrete state, or "*". See Details.
type	[character(1): "contiguous"] "contiguous" (default) considers consecutive positions; "gapped" considers positions spaced by <code>gap + 1</code> apart.
gap	[integer(): 1L] Gap size(s) used when <code>type = "gapped"</code> . Multiple values produce one row per (length, gap) combination.
min_count	[integer(1): 1L] Minimum total occurrences to retain.

min_support	[numeric(1): 0] Minimum proportion of sequences containing the pattern at least once.
min_lift	[numeric(1): 0] Minimum lift (observed / expected under marginal independence). 0 disables the filter.
start, end, contain	[character()] Filter rows whose first / last element is in start / end, or whose element sequence contains all of contain. Compared against the alphabet of the chosen level (types or concrete states).

### Details

- level = "state" (default) - enumerate concrete state-level patterns (e.g. Command->Execute->Command) and annotate each row with the type-level template it instantiates (e.g. Human->AI->Human).
- level = "type" - enumerate type-level meta-paths only (one row per actor-type pattern, summed over its concrete instances).

At either level, a schema can filter the search. Schema parts can be type names (expand to every concrete code in that group), concrete states, or "\*" (any element). Parts can be mixed freely - e.g. "Human->Ask->Human" means "any Human code, then Ask, then any Human code". At level = "type" concrete codes resolve to their type, so the same schema becomes Human->AI->Human.

Operates on the actor partition stored on the network by [build\\_htna\(\)](#).

### Value

An object of class c("htna\_meta\_paths", "htna\_paths", "data.frame"). At level = "state" the columns are schema, meta\_schema, length, gap, count, n\_sequences, support, frequency, lift. At level = "type" the meta\_schema column is omitted (the schema column already holds the type pattern). Attributes: n\_sequences, alphabet, level, and (when supplied) schema.

### See Also

[build\\_htna\(\)](#).

### Examples

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")

# Concrete state-level patterns of length 2..4 (default)
extract_meta_paths(net)

# Type-level meta-path summary
extract_meta_paths(net, level = "type")

# Concrete instances of a type-level template
extract_meta_paths(net, schema = "Human->AI->Human")

# Mix types, concrete codes, and wildcards
```

```

extract_meta_paths(net, schema = "Human->Ask->Human")
extract_meta_paths(net, schema = "Human->*->Human")

# Gapped patterns; lift threshold
extract_meta_paths(net, length = 3, type = "gapped", gap = 1:2)
extract_meta_paths(net, length = 3, min_lift = 1.2)

```

---

frequencies\_htna

*Tidy Per-Actor Frequency Table for an htna Network*


---

### Description

Returns the within-network state frequency table, partitioned by actor. One row per (actor, state) pair, with count and proportion columns. The data side of [plot\\_frequencies\\_htna\(\)](#).

### Usage

```
frequencies_htna(x)
```

### Arguments

x                    An htna network from [build\\_htna\(\)](#).

### Details

Internally a thin wrapper around the htna S3 method shipped by Nestimate (`state_distribution.htna`); exposed under the `frequencies_htna()` name as the canonical "give me the frequency table for this network" entry point.

Suffixed `_htna` to avoid clashing with `Nestimate::frequencies()` (which takes raw long-format data and a column-name spec) when both packages are loaded. If you have raw data rather than an htna network, call `Nestimate::frequencies()` directly.

### Value

A data frame with columns `group` (actor), `state`, `count`, `proportion`.

### See Also

[plot\\_frequencies\\_htna\(\)](#) for the rendered version, [state\\_distribution\\_htna\(\)](#) (same function under the upstream name), [mosaic\\_plot\\_htna\(\)](#).

### Examples

```

data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
frequencies_htna(net)

```

---

htna_palette	<i>Default HTNA colour palette</i>
--------------	------------------------------------

---

### Description

A colour palette for up to 6 actor groups, used by `plot_htna()` when no explicit colours are supplied.

### Usage

```
htna_palette
```

### Format

An object of class character of length 6.

### Value

A character vector of 6 hex colour codes.

### Examples

```
htna_palette
```

---

human_ai	<i>Simplified Human + AI Interaction Sequences</i>
----------	--

---

### Description

A long-format heterogeneous sequence dataset built from `Nestimate::human_long` and `Nestimate::ai_long` by collapsing several near-synonym codes into a smaller, more interpretable alphabet. Suitable as a teaching example for `build_htna()`.

### Usage

```
human_ai
```

### Format

A data frame with 19347 rows and 11 columns:

**message\_id** Integer. Source message identifier.

**project** Character. Project label (e.g. "Project\_1").

**session\_id** Character. Session identifier — pass to `build_htna()` as the session key.

**timestamp** Integer. Unix timestamp of the message.

- session\_date** Character. Date of the session (YYYY-MM-DD).
- code** Character. Simplified action code; the code-column value passed to `build_htna()`.
- cluster** Character. Original cluster label from the source data, retained for reference (see Details).
- code\_order** Integer. Order of the code within the message.
- order\_in\_session** Integer. Order of the row within the session — pass as the order key to `build_htna()`.
- actor\_type** Character. "AI" or "Human" — the actor partition for `build_htna(actor_type = "actor_type")`.
- phase** Factor with levels "Early" and "Late". Session-level cohort tag from a chronological split: sessions ordered by their first `session_date` (with `session_id` as a deterministic tiebreak), then split in half. Suitable for `build_htna(group = "phase")`.

## Details

Code remapping (all other codes pass through unchanged):

- AI: Investigate, Ask, Inquire → Ask; Explain, Report → Report.
- Human: Command, Request → Request; Correct, Verify → Check; Interrupt, Frustrate → Frustrate.

Resulting alphabets:

- AI (6): Ask, Delegate, Execute, Plan, Repair, Report.
- Human (6): Check, Frustrate, Inquire, Refine, Request, Specify.

Because two source codes can map to the same simplified code while originally belonging to different cluster values, a single simplified code may appear with more than one cluster label across rows. The cluster column is preserved verbatim from the source data and should be treated as informational only.

## Source

Derived from `Nestimate::human_long` and `Nestimate::ai_long`; see `data-raw/human_ai.R` for the build script.

## See Also

[human\\_simplified](#), [ai\\_simplified](#), [human\\_ai\\_codebook](#).

## Examples

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
plot_htna(net)
```

---

human\_ai\_codebook      *Code → Actor-Type Codebook for human\_ai*

---

### Description

A tidy two-column lookup tagging every simplified code in [human\\_ai](#) with its actor type. Ready to pass as `node_groups` to `build_htna()` (Form 3b in the vignette("input-formats", package = "htna")).

### Usage

```
human_ai_codebook
```

### Format

A data frame with 12 rows and 2 columns:

**code** Character. Simplified action code (one of the 12 Human + AI codes in [human\\_ai](#)).

**actor\_type** Character. "Human" or "AI".

### Source

Derived from [human\\_simplified](#) and [ai\\_simplified](#); see `data-raw/human_ai.R` for the build script.

### See Also

[human\\_ai](#), [human\\_simplified](#), [ai\\_simplified](#).

### Examples

```
data(human_ai, human_ai_codebook)
net <- build_htna(human_ai, node_groups = human_ai_codebook)
plot_htna(net)
```

---

human\_simplified      *Simplified Human Interaction Sequences (per-actor frame)*

---

### Description

The Human-only slice of [human\\_ai](#), in a long-format data frame ready to feed the named-list form of `build_htna()`. Codes have already been collapsed into the simplified Human alphabet; see [human\\_ai](#) for the remapping rules.

**Usage**

```
human_simplified
```

**Format**

A data frame with 10796 rows and 10 columns. Schema matches [human\\_ai](#) minus the phase column; every value in actor\_type is "Human".

**Source**

Derived from `Nestimate::human_long`; see `data-raw/human_ai.R` for the build script.

**See Also**

[human\\_ai](#), [ai\\_simplified](#), [human\\_ai\\_codebook](#).

**Examples**

```
data(human_simplified, ai_simplified)
net <- build_htna(list(Human = human_simplified, AI = ai_simplified))
plot_htna(net)
```

---

markov\_order\_test\_htna

*Markov-Order Adequacy Test*

---

**Description**

htna-named alias of `Nestimate::markov_order_test()`. Tests whether a first-order Markov model is adequate for the observed sequences, or whether a higher order is required, by comparing the empirical transition structure against orders  $1 \dots \text{max\_order}$  via permutation.

**Usage**

```
markov_order_test_htna(  
  data,  
  max_order = 3L,  
  n_perm = 500L,  
  alpha = 0.05,  
  parallel = FALSE,  
  n_cores = 2L,  
  seed = NULL  
)
```

**Arguments**

<code>data</code>	A <code>data.frame</code> (wide format, one sequence per row) or list of character vectors (one per trajectory). NAs are treated as end of sequence.
<code>max_order</code>	Integer. Highest Markov order to test. Default 3.
<code>n_perm</code>	Integer. Number of within- <i>w</i> permutations per order. Default 500.
<code>alpha</code>	Numeric. Significance level for order selection. Default 0.05.
<code>parallel</code>	Logical. Use <code>parallel::mclapply</code> for permutations. Default FALSE (set TRUE only on Unix-like systems).
<code>n_cores</code>	Integer. Cores for parallel execution. Default 2.
<code>seed</code>	Optional integer seed for reproducibility.

**Details**

Operates on raw sequence data (a list of character vectors or a wide-format data frame), not on an `htna` network object. Use alongside `reliability_htna()` and `casedrop_reliability_htna()` when assessing whether a Markov-1 transition network is appropriate before trusting downstream `htna` analyses.

Suffixed `_htna` to avoid clashing with `Nestimate::markov_order_test()` when both packages are loaded.

**Value**

An object of class `markov_order_test` with components `test_table`, `optimal_order`, and the inputs. See `Nestimate::markov_order_test()` for full details and the corresponding `plot()` method.

**See Also**

`reliability_htna()`, `casedrop_reliability_htna()`, `centrality_stability_htna()`.

**Examples**

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
markov_order_test_htna(net$data, max_order = 2, n_perm = 50, seed = 1)
```

---

mosaic\_plot\_htna

*Chi-square Mosaic Plot of a Transition Network*


---

**Description**

`htna`-named alias of `Nestimate::mosaic_plot()`. Renders a chi-square mosaic where row x column area equals the joint share of (from, to) transitions and fill encodes the standardized residual (blue = over-represented, red = under-represented, white = at-expected).

**Usage**

```
mosaic_plot_htna(x, ...)
```

**Arguments**

`x` One of the four data-bearing Nestimate classes: `netobject` (single mosaic of `$weights`), `netobject_group` (one panel per group), `mcml` (between-cluster mosaic by default; per-cluster panels with `level = "within"`), or `htna` (single mosaic of `$weights`; `htna` inherits `netobject` so the geometry matches). Also accepts a contingency table or plain numeric matrix for ad-hoc plotting.

`...` Ignored.

**Details**

Designed with `htna` in mind: Nestimate ships an explicit `mosaic_plot.htna` S3 method that recognises the actor partition on `htna` networks. Pass an `htna` network from `build_htna()` directly. The underlying transition matrix must be integer-weighted (build with `method = "frequency"`), since the chi-square test needs counts.

Axis tick labels are coloured by actor group using the `htna` palette (`htna_palette`), matching the colours used elsewhere in `htna` (e.g. `plot_htna()`).

Suffixed `_htna` to avoid clashing with `Nestimate::mosaic_plot()` when both packages are loaded.

**Value**

A ggplot or gtable, depending on input shape. See `Nestimate::mosaic_plot()` for full details and the per-class S3 methods.

**See Also**

`plot_frequencies_htna()` for the marginal-frequency companion view.

**Examples**

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type",
                 method = "frequency")
mosaic_plot_htna(net, n_perm = 50, seed = 1)
```

---

permutation\_htna

*Permutation Test for Network Differences*


---

**Description**

`htna`-named alias of `Nestimate::permutation()`. Tests whether observed edge-weight differences between two networks (or all pairwise differences within a `netobject_group`) exceed what would be expected under a null of identical generating processes.

**Usage**

```
permutation_htna(
  x,
  y = NULL,
  iter = 1000L,
  alpha = 0.05,
  paired = FALSE,
  adjust = "none",
  nlambda = 50L,
  seed = NULL
)
```

**Arguments**

x	A netobject (from <a href="#">build_network</a> ).
y	A netobject (from <a href="#">build_network</a> ). Must use the same method and have the same nodes as x.
iter	Integer. Number of permutation iterations (default: 1000).
alpha	Numeric. Significance level (default: 0.05).
paired	Logical. If TRUE, permute within pairs (requires equal number of observations in x and y). Default: FALSE.
adjust	Character. p-value adjustment method passed to <a href="#">p.adjust</a> (default: "none"). Common choices: "holm", "BH", "bonferroni".
nlambda	Integer. Number of lambda values for the glassopath regularisation path (only used when method = "glasso"). Higher values give finer lambda resolution at the cost of speed. Default: 50.
seed	Integer or NULL. RNG seed for reproducibility.

**Details**

Works on htna networks: the actor partition (`$node_groups`, `$actor_levels`, htna class) is preserved on `result$x` / `result$y`, so [plot\\_htna\\_diff\(\)](#) can render the result with htna's colour and layout conventions.

Suffixed `_htna` to avoid clashing with `Nestimate::permutation()` when both packages are loaded.

**Value**

An object of class `net_permutation` (single pair) or `net_permutation_group` (multiple pairs). See [Nestimate::permutation\(\)](#) for the full slot list.

**See Also**

[plot\\_htna\\_diff\(\)](#) to plot the result.

## Examples

```
data(human_ai)
grp <- build_htna(human_ai, actor_type = "actor_type", group = "phase")
permutation_htna(grp$Early, grp$Late, iter = 50)
```

---

plot.htna\_stability *Plot Method for htna Centrality Stability Objects*

---

## Description

S3 method for plotting htna centrality stability results. Dispatches to `cograph`'s plotting method with htna-specific styling.

## Usage

```
## S3 method for class 'htna_stability'
plot(x, ...)
```

## Arguments

`x` An object of class `htna_stability` from `centrality_stability_htna()`.  
`...` Additional arguments passed to the plotting method.

## Value

A ggplot object or plot output, depending on the underlying method.

## Examples

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
cs <- centrality_stability_htna(net, iter = 20, seed = 1)
plot(cs)
```

---

```
plot.htna_stability_group
```

*Plot Method for htna Stability Groups*

---

**Description**

S3 method for plotting grouped htna stability results.

**Usage**

```
## S3 method for class 'htna_stability_group'  
plot(x, ...)
```

**Arguments**

`x` An object of class `htna_stability_group` from `centrality_stability_htna()`.  
`...` Additional arguments passed to the plotting method.

**Value**

A combined plot or list of plots for each group.

**Examples**

```
data(human_ai)  
grp <- build_htna(human_ai, actor_type = "actor_type", group = "phase")  
cs <- centrality_stability_htna(grp, iter = 20, seed = 1)  
plot(cs)
```

---

```
plot_centralities
```

*Plot Centrality Measures*

---

**Description**

Faceted bar plot of node-level centralities, one panel per measure. Mirrors the look of `tna::plot.tna_centralities()`.

**Usage**

```
plot_centralities(  
  x,  
  measures = c("OutStrength", "InStrength", "ClosenessIn", "ClosenessOut", "Closeness",  
              "Betweenness", "BetweennessRSP", "Diffusion", "Clustering"),  
  by = c("state", "group"),  
  reorder = TRUE,  
  ncol = 3,
```

```
scales = c("free_x", "fixed"),
colors = NULL,
labels = TRUE,
...
)
```

### Arguments

x	An htna network, htna_group, or htna_centralities data frame from <a href="#">centralities_htna()</a> .
measures	Centralities to plot. Default: all nine.
by	"state" (default) gives each node its own colour; "group" colours by actor group (Human, AI, ...) using <a href="#">htna_palette</a> .
reorder	If TRUE, sort nodes by value within each measure.
ncol	Number of facet columns. Default 3.
scales	Facet scaling: "free_x" (default) or "fixed".
colors	Optional fill colours, recycled per group/node.
labels	If TRUE (default), draw the value next to each bar.
...	Forwarded to <a href="#">centralities_htna()</a> when computing on the fly.

### Details

Accepts an htna network, an htna\_group, or a data frame produced by [centralities\\_htna\(\)](#). For groups, bars are coloured by group within each panel.

### Value

A ggplot object.

### See Also

[centralities\\_htna\(\)](#).

### Examples

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
plot_centralities(net)

grp <- build_htna(human_ai, actor_type = "actor_type", group = "phase")
plot_centralities(grp)
```

---

`plot_edge_betweenness_htna`*Plot the Edge Betweenness Network*

---

### Description

Computes `edge_betweenness_htna()` and renders it with the htna actor styling (multi-actor circular layout, `htna_palette` node colours, actor legend). Edges are scaled by their betweenness score, so the most-traveled shortest-path edges stand out.

### Usage

```
plot_edge_betweenness_htna(x, directed = TRUE, invert = TRUE, ...)
```

### Arguments

<code>x</code>	An htna network from <code>build_htna()</code> or, equivalently, the result of <code>edge_betweenness_htna()</code> .
<code>directed</code> , <code>invert</code>	Forwarded to <code>edge_betweenness_htna()</code> when <code>x</code> is a plain htna network. Ignored when <code>x</code> already inherits from <code>htna_edge_betweenness</code> .
<code>...</code>	Forwarded to <code>plot_htna()</code> (e.g. <code>minimum</code> , <code>layout</code> , <code>group_colors</code> ).

### Value

The value returned by `plot_htna()` (invisibly).

### See Also

`edge_betweenness_htna()`, `plot_htna()`.

### Examples

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
plot_edge_betweenness_htna(net)
plot_edge_betweenness_htna(net, minimum = 5)
```

---

plot\_frequencies\_htna *Plot State Frequencies (htna-named)*

---

### Description

Renders one of three views of an htna network's state frequencies: the upstream treemap (default), a combined actor-coloured bar chart, or a per-actor faceted bar chart.

### Usage

```
plot_frequencies_htna(x, view = c("treemap", "bars", "facet"), ...)
```

### Arguments

x	An htna network from <a href="#">build_htna()</a> .
view	One of "treemap" (default), "bars", or "facet". <ul style="list-style-type: none"> <li>"treemap" forwards to <a href="#">Nestimate::plot_state_frequencies()</a> and renders the chart automatically; returns the underlying state_freq object invisibly.</li> <li>"bars" builds a combined bar chart with all states on one y-axis, sorted by count, fill coloured by actor (using htna_palette keyed off x\$actor_levels). Returns the ggplot.</li> <li>"facet" builds a per-actor faceted bar chart with scales = "free_y" so each panel only shows its own actor's states. Returns the ggplot.</li> </ul>
...	Forwarded to <a href="#">Nestimate::plot_state_frequencies()</a> when view = "treemap". Ignored for "bars" and "facet" — those return ggplot objects, so customisation works through standard ggplot composition (+ theme(...), + labs(...), + scale_fill_*(...), etc.).

### Details

Suffixed \_htna to avoid clashing with [Nestimate::plot\\_state\\_frequencies\(\)](#) when both packages are loaded.

### Value

For view = "treemap": the state\_freq object invisibly. For "bars" / "facet": a ggplot, returned visibly so the chart auto-prints and standard + composition works.

### See Also

[frequencies\\_htna\(\)](#) for the underlying tidy table, [state\\_frequencies\\_htna\(\)](#), [state\\_distribution\\_htna\(\)](#), [mosaic\\_plot\\_htna\(\)](#).

**Examples**

```

data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
plot_frequencies_htna(net, view = "treemap")
if (requireNamespace("ggplot2", quietly = TRUE)) {
  plot_frequencies_htna(net, view = "bars")
}

```

plot\_htna

*Plot a Heterogeneous Transition Network***Description**

Wrapper around `cograph::plot_htna()` with defaults suited for HTNA networks built by `build_htna()`. When `layout = "circular"`, a custom layout is computed so that the first actor group appears on the left.

**Usage**

```

plot_htna(
  x,
  layout = "circular",
  group_colors = htna_palette,
  group_shapes = htna_shape_palette,
  minimum = 0.05,
  ...
)

## S3 method for class 'htna'
plot(x, ...)

## S3 method for class 'htna_group'
plot(x, ...)

```

**Arguments**

<code>x</code>	A network object produced by <code>build_htna()</code> .
<code>layout</code>	Character. Layout algorithm. Default <code>"circular"</code> .
<code>group_colors</code>	Character vector of colours, one per actor group. Defaults to the built-in <code>htna_palette</code> .
<code>group_shapes</code>	Character vector of node shapes, one per actor group. Defaults to the built-in <code>htna_shape_palette</code> .
<code>minimum</code>	Numeric. Minimum absolute edge weight to display. Edges below this threshold are hidden. Default <code>0.05</code> .
<code>...</code>	Additional arguments passed to <code>cograph::plot_htna()</code> .

**Value**

Called for its side effect (a plot). Returns `x` invisibly.

**See Also**

[cograph::plot\\_htna\(\)](#), [build\\_htna\(\)](#)

**Examples**

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
plot_htna(net)
plot_htna(net, layout = "auto", minimum = 0.1)
```

---

plot\_htna\_bootstrap     *Plot an htna Bootstrap Result*

---

**Description**

Renders the bootstrap result via [cograph::splot.net\\_bootstrap\(\)](#) so the CI / significance / dashed-edge overlay is preserved, but overrides the layout and node styling to match [plot\\_htna\(\)](#) (multi-group circular layout, warm [htna\\_palette](#), darkened donut).

**Usage**

```
plot_htna_bootstrap(boot, group_colors = htna_palette, ...)
```

```
## S3 method for class 'htna_bootstrap'
plot(x, ...)
```

**Arguments**

<code>boot</code>	An object from <a href="#">bootstrap_htna()</a> (or a <code>htna_bootstrap_group</code> list of them).
<code>group_colors</code>	Character vector of colours, one per actor group. Defaults to the built-in <a href="#">htna_palette</a> .
<code>...</code>	Forwarded to <a href="#">cograph::splot.net_bootstrap()</a> (e.g. <code>display</code> , <code>show_ci</code> , <code>show_stars</code> ). User args win.
<code>x</code>	Same as <code>boot</code> ; used when calling via the <code>plot()</code> generic.

**Details**

For an `htna_bootstrap_group` (i.e. the result of running [bootstrap\\_htna\(\)](#) on a `build_htna(..., group = ...)` output), each element is plotted individually with its name as the title - this function does not manage `par(mfrow=...)`.

**Value**

The value returned by `cograph::splot.net_bootstrap()` (invisibly), or the input group invisibly.

**See Also**

`bootstrap_htna()`, `plot_htna()`, `cograph::splot.net_bootstrap()`.

**Examples**

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
boot <- bootstrap_htna(net, iter = 50)
plot_htna_bootstrap(boot)
plot_htna_bootstrap(boot, display = "significant")
```

---

plot\_htna\_diff

*Plot the Difference Between Two htna Networks*


---

**Description**

Renders the elementwise edge-weight difference  $x - y$  as a heterogeneous transition network. Accepts:

**Usage**

```
plot_htna_diff(
  x,
  y = NULL,
  pos_color = "#009900",
  neg_color = "#C62828",
  group_colors = htna_palette,
  ...
)
```

**Arguments**

<code>x</code>	One of: an htna network, a <code>net_permutation</code> result, or a <code>net_permutation_group</code> .
<code>y</code>	htna network (only used when <code>x</code> is also an htna network).
<code>pos_color</code>	Edge colour for positive differences. Default <code>"#009900"</code> .
<code>neg_color</code>	Edge colour for negative differences. Default <code>"#C62828"</code> .
<code>group_colors</code>	Character vector of colours, one per actor group. Defaults to the built-in <code>htna_palette</code> .
<code>...</code>	Forwarded to the underlying <code>splot</code> dispatcher ( <code>cograph::splot()</code> or <code>cograph::splot.net_permutation</code> ).

**Details**

- Two htna networks (x, y): plots x - y via `cograph::splot()` with positive/negative edge coloring.
- A `net_permutation` result: routes through `cograph::splot.net_permutation()` so the significance overlay (significant pos/neg edges, dashed non-significant edges, stars) is drawn upstream.
- A `net_permutation_group`: iterates over all pairwise comparisons and plots each (no `par(mfrow=...)` management - user controls layout).

In all cases the layout, node colours and donut match `plot_htna()`.

**Value**

The value of the underlying `splot` call (invisibly).

**See Also**

`plot_htna()`, `build_htna()`, `Nestimate::permutation()`.

**Examples**

```
data(human_ai)
grp <- build_htna(human_ai, actor_type = "actor_type", group = "phase")
plot_htna_diff(grp$Early, grp$Late)

perm <- permutation_htna(grp$Early, grp$Late, iter = 50)
plot_htna_diff(perm)
plot_htna_diff(perm, show_nonsig = TRUE)
```

---

plot_sequences	<i>Sequence plot generic</i>
----------------	------------------------------

---

**Description**

S3 generic dispatched on x. Calling `plot_sequences(net)` on an htna network forwards to `sequence_plot_htna()`.

**Usage**

```
plot_sequences(x, ...)
```

**Arguments**

x	An object to plot.
...	Forwarded to the method.

**Value**

Method-defined.

**Examples**

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
plot_sequences(net)
```

---

`print.htna_stability` *Print Method for htna Centrality Stability Objects*

---

**Description**

S3 method for printing htna centrality stability results.

**Usage**

```
## S3 method for class 'htna_stability'
print(x, ...)
```

**Arguments**

- `x` An object of class `htna_stability` from `centrality_stability_htna()`.
- `...` Additional arguments passed to the print method.

**Value**

Invisibly returns the object, prints stability information.

**Examples**

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
cs <- centrality_stability_htna(net, iter = 20, seed = 1)
print(cs)
```

---

```
print.htna_stability_group
```

*Print Method for htna Stability Groups*

---

### Description

S3 method for printing grouped htna stability results.

### Usage

```
## S3 method for class 'htna_stability_group'
print(x, ...)
```

### Arguments

`x` An object of class `htna_stability_group` from [centrality\\_stability\\_htna\(\)](#).  
`...` Additional arguments passed to the print method.

### Value

Invisibly returns the object, prints group stability information.

### Examples

```
data(human_ai)
grp <- build_htna(human_ai, actor_type = "actor_type", group = "phase")
cs <- centrality_stability_htna(grp, iter = 20, seed = 1)
print(cs)
```

---

```
reliability_htna
```

*Network Reliability for an htna Network*

---

### Description

Thin wrapper around [Nestimate::network\\_reliability\(\)](#) that preserves the htna actor partition on every model in the returned `$models` slot, so downstream htna-aware code (plotting, centralities, etc.) keeps working on the reliability output.

### Usage

```
reliability_htna(..., iter = 1000L, split = 0.5, scale = "none", seed = NULL)
```

**Arguments**

...	One or more htna networks built by <a href="#">build_htna()</a> .
iter	Integer. Number of split-half iterations. Default 1000.
split	Numeric in (0, 1). Proportion of sessions used for the first half-network. Default 0.5.
scale	One of "none", "minmax", "standardize", "proportion". Forwarded to <a href="#">Nestimate::network_reliability()</a> .
seed	Optional integer seed for reproducibility. Default NULL (no seed reset).

**Details**

Mirrors the signature of the underlying function: pass one or more networks via ... plus the optional iter, split, scale, and seed arguments. All networks passed through ... must be htna networks built by [build\\_htna\(\)](#).

**Value**

An object of class c("htna\_reliability", "net\_reliability"), with the same components as [Nestimate::network\\_reliability\(\)](#) — iterations, summary, models, iter, split, scale — and each entry of \$models carrying the htna actor partition (\$nodes\$groups, \$node\_groups, \$actor\_levels).

**See Also**

[Nestimate::network\\_reliability\(\)](#), [bootstrap\\_htna\(\)](#).

**Examples**

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
rel <- reliability_htna(net, iter = 30, seed = 1)
rel$summary
```

---

sequence\_compare\_htna *Subsequence Pattern Comparison Across Groups*

---

**Description**

Extends [Nestimate::sequence\\_compare\(\)](#) with a level argument that lets the comparison run on type-level (meta-path) sequences as well as state-level sequences.

**Usage**

```
sequence_compare_htna(
  x,
  group = NULL,
  level = c("state", "type"),
  sub = 3:5,
  min_freq = 5L,
  test = c("permutation", "chisq", "none"),
  iter = 1000L,
  adjust = "fdr"
)
```

**Arguments**

<code>x</code>	A <code>netobject_group</code> (from <code>grouped build_network</code> ), a <code>netobject</code> (requires <code>group</code> ), or a wide-format data frame (requires <code>group</code> ).
<code>group</code>	Character or vector. Column name or vector of group labels. Not needed for <code>netobject_group</code> .
<code>level</code>	Character. "state" (default) compares concrete state-level k-grams; "type" first replaces each state with its actor type so the comparison runs on meta-paths (e.g. Human->AI->Human). Only meaningful when <code>x</code> is an htna network with an actor partition.
<code>sub</code>	Integer vector. Pattern lengths to analyze. Default: 3:5.
<code>min_freq</code>	Integer. Minimum frequency in each group for a pattern to be included. Default: 5.
<code>test</code>	Character. Inference method: one of "permutation" (default), "chisq", or "none". See Details.
<code>iter</code>	Integer. Permutation iterations. Only used when <code>test = "permutation"</code> . Default: 1000.
<code>adjust</code>	Character. P-value correction method (see <a href="#">p.adjust</a> ). Default: "fdr".

**Details**

Extracts all k-gram patterns (subsequences of length `k`) from the sequences in each cohort, computes standardised residuals against the independence model, and optionally runs a permutation or chi-square test for differences in pattern rates between cohorts.

Operates on grouped htna networks (built via `build_htna(..., group = ...)`), single htna networks paired with a `group` argument, or wide-format sequence data with an explicit `group` argument. The actor partition itself is not consumed when `level = "state"` — sequence comparison is between cohorts of sessions, not between actors. When `level = "type"`, concrete codes are folded into their actor type before pattern enumeration, so the comparison runs on meta-paths.

**Value**

An object of class `net_sequence_compare`. See `Nestimate::sequence_compare()` for full details and the corresponding `plot()` method.

**See Also**

[permutation\\_htna\(\)](#) for whole-network differences, [mosaic\\_plot\\_htna\(\)](#) for single-step transition residuals, [extract\\_meta\\_paths\(\)](#) for descriptive meta-path enumeration.

**Examples**

```
data(human_ai)
grp <- build_htna(human_ai, actor_type = "actor_type", group = "phase")

# State-level comparison (default)
sequence_compare_htna(grp, iter = 50)

# Meta-path comparison
sequence_compare_htna(grp, level = "type", iter = 50)
```

---

sequence\_plot\_htna      *Plot Per-Actor Sequences From an htna Network*

---

**Description**

Extracts each actor's events from the combined wide sequence in `net$data` (using `net$node_groups` as the node-to-actor lookup), compresses each session into the actor's own ordered events, pads to a common width, and renders with `Nestimate::sequence_plot()` grouped by actor. Each session contributes one row per actor that had at least one event in it.

**Usage**

```
sequence_plot_htna(
  net,
  by = c("state", "group"),
  type = c("index", "heatmap", "distribution"),
  grouped_legend = TRUE,
  ...
)

## S3 method for class 'htna'
plot_sequences(x, ...)
```

**Arguments**

<code>net</code>	An htna network from <a href="#">build_htna()</a> . Must have <code>\$data</code> and <code>\$node_groups</code> populated.
<code>by</code>	"state" (default) keeps state-level colouring with one row per (session, actor) extracted from <code>net\$data</code> ; "group" re-colours the original combined session matrix by actor (each cell = the actor that acted at that time step).

type	Sequence plot layout: "index" (default; one panel per actor, vertically stacked), "heatmap" (single carpet with a white separator at the actor boundary, controllable via <code>k_line_width</code> ), or "distribution" (one stacked-area panel per actor).
grouped_legend	Logical. If TRUE (default) and <code>by = "state"</code> , the per-state legend is split into one block per actor with the actor name as a sub-title.
...	Forwarded to <code>Nestimate::sequence_plot()</code> .
x	Same as <code>net</code> ; used when calling via the <code>plot_sequences()</code> generic.

**Value**

Invisibly, the list returned by `Nestimate::sequence_plot()`.

**See Also**

`Nestimate::sequence_plot()`, `build_htna()`.

**Examples**

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")

sequence_plot_htna(net)           # index, faceted
sequence_plot_htna(net, type = "heatmap") # single carpet, white gulf
sequence_plot_htna(net, type = "distribution")
```

---

state\_distribution\_htna

*State Distribution Across Time*

---

**Description**

htna-named alias of `Nestimate::state_distribution()`. Returns the per-timestep distribution of states across sequences, suitable for driving stacked-area or bar plots.

**Usage**

```
state_distribution_htna(x, ...)
```

**Arguments**

x	A <code>netobject</code> , <code>netobject_group</code> , <code>mcmc1</code> , or <code>htna</code> object.
...	Currently unused.

## Details

Designed with htna in mind: Nestimate ships an explicit `state_distribution.htna` S3 method that uses the actor partition carried by `build_htna()` networks.

Suffixed `_htna` to avoid clashing with `Nestimate::state_distribution()` when both packages are loaded.

## Value

A data frame with one row per (timestep, state). See `Nestimate::state_distribution()` for full details.

## See Also

`state_frequencies_htna()` for the within-network summary.

## Examples

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
state_distribution_htna(net)
```

---

state\_frequencies\_htna

*Tidy State Frequency Table*

---

## Description

htna-named alias of `Nestimate::state_frequencies()`. Returns a tidy data frame with `state`, `count`, and `proportion` columns summarising the within-network state vocabulary.

## Usage

```
state_frequencies_htna(data)
```

## Arguments

`data` A list of character vectors (trajectories) or a `data.frame`.

## Details

Companion to `plot_frequencies_htna()` (which is htna-aware via an explicit S3 method). `state_frequencies_htna()` itself operates on the raw sequence data and is the data side of the same family used by the htna tutorials.

Suffixed `_htna` to avoid clashing with `Nestimate::state_frequencies()` when both packages are loaded.

**Value**

A data frame. See `Nestimate::state_frequencies()` for details.

**See Also**

`plot_frequencies_htna()`, `state_distribution_htna()`.

**Examples**

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
state_frequencies_htna(net$data)
```

---

summary.htna

*Summarise a Heterogeneous Transition Network*


---

**Description**

Prints a per-actor summary of an htna network: which nodes belong to which actor type and how the non-zero edges distribute across the actor partition. The result is also returned invisibly as a list so callers can inspect the structured summary programmatically.

**Usage**

```
## S3 method for class 'htna'
summary(object, max_nodes = 12L, ...)
```

```
## S3 method for class 'htna_group'
summary(object, max_nodes = 12L, ...)
```

**Arguments**

<code>object</code>	An htna network from <code>build_htna()</code> or an <code>htna_group</code> .
<code>max_nodes</code>	Integer. Maximum number of nodes to list per actor type before truncating with an ellipsis. Default 12.
<code>...</code>	Forwarded for compatibility; currently unused.

**Value**

Invisibly, a list with components:

- `actors` - data frame with one row per actor type (`actor`, `n_nodes`, `nodes`).
- `edges_by_actor` - integer matrix of non-zero edge counts, rows are source actor, columns are target actor.
- `n_nodes`, `n_edges`, `n_sessions`, `n_timesteps`, `method`.

**Examples**

```
data(human_ai)
net <- build_htna(human_ai, actor_type = "actor_type")
summary(net)
```

# Index

## \* datasets

- ai\_simplified, 2
  - htna\_palette, 17
  - human\_ai, 17
  - human\_ai\_codebook, 19
  - human\_simplified, 19
- ai\_simplified, 2, 18–20
- association\_rules\_htna, 3
- bootstrap, 4
- bootstrap.htna (bootstrap\_htna), 5
- bootstrap\_htna, 5
- bootstrap\_htna(), 4, 9, 12, 30, 31, 35
- build\_htna, 2, 6, 17–19
- build\_htna(), 5, 6, 10, 11, 13–16, 22, 27–30, 32, 35, 37–40
- build\_network, 7, 8, 23
- build\_tna, 8
- casedrop\_reliability\_htna, 8
- casedrop\_reliability\_htna(), 21
- centralities\_htna, 10
- centralities\_htna(), 13, 26
- centrality\_stability\_htna, 11
- centrality\_stability\_htna(), 9, 21, 24, 25, 33, 34
- cograph::centrality\_betweenness(), 10
- cograph::centrality\_closeness(), 10
- cograph::centrality\_current\_flow\_betweenness(), 11
- cograph::centrality\_diffusion(), 11
- cograph::centrality\_incloseness(), 10
- cograph::centrality\_instrength(), 10
- cograph::centrality\_outcloseness(), 10
- cograph::centrality\_outstrength(), 10
- cograph::centrality\_transitivity(), 11
- cograph::cograph, 10
- cograph::plot\_htna(), 29, 30
- cograph::splot(), 31, 32
- cograph::splot.net\_bootstrap(), 30, 31
- cograph::splot.net\_permutation(), 31, 32
- edge\_betweenness\_htna, 13
- edge\_betweenness\_htna(), 27
- extract\_meta\_paths, 14
- extract\_meta\_paths(), 37
- frequencies\_htna, 16
- frequencies\_htna(), 4, 28
- htna\_palette, 17, 22, 26, 27, 29–31
- human\_ai, 2, 3, 17, 19, 20
- human\_ai\_codebook, 3, 18, 19, 20
- human\_simplified, 3, 18, 19, 19
- markov\_order\_test\_htna, 20
- mosaic\_plot\_htna, 21
- mosaic\_plot\_htna(), 4, 16, 28, 37
- Nestimate::association\_rules(), 3, 4
- Nestimate::bootstrap\_network(), 5, 6
- Nestimate::build\_network(), 4
- Nestimate::casedrop\_reliability(), 8, 9
- Nestimate::centrality\_stability(), 11, 12
- Nestimate::markov\_order\_test(), 20, 21
- Nestimate::mosaic\_plot(), 21, 22
- Nestimate::network\_reliability(), 34, 35
- Nestimate::permutation(), 22, 23, 32
- Nestimate::plot\_state\_frequencies(), 28
- Nestimate::sequence\_compare(), 35, 36
- Nestimate::sequence\_plot(), 37, 38
- Nestimate::state\_distribution(), 38, 39
- Nestimate::state\_frequencies(), 39, 40
- p.adjust, 23, 36
- permutation\_htna, 22

permutation\_htna(), 37  
plot.htna (plot\_htna), 29  
plot.htna\_bootstrap  
    (plot\_htna\_bootstrap), 30  
plot.htna\_group (plot\_htna), 29  
plot.htna\_stability, 24  
plot.htna\_stability\_group, 25  
plot.centralities, 25  
plot.edge\_betweenness\_htna, 27  
plot.frequencies\_htna, 28  
plot.frequencies\_htna(), 16, 22, 39, 40  
plot\_htna, 8, 29  
plot\_htna(), 13, 17, 22, 27, 30–32  
plot\_htna\_bootstrap, 30  
plot\_htna\_diff, 31  
plot\_htna\_diff(), 23  
plot\_sequences, 32  
plot\_sequences.htna  
    (sequence\_plot\_htna), 37  
print.htna\_stability, 33  
print.htna\_stability\_group, 34  
  
reliability\_htna, 34  
reliability\_htna(), 9, 12, 21  
  
sequence\_compare\_htna, 35  
sequence\_plot\_htna, 37  
sequence\_plot\_htna(), 32  
state\_distribution\_htna, 38  
state\_distribution\_htna(), 16, 28, 40  
state\_frequencies\_htna, 39  
state\_frequencies\_htna(), 28, 39  
summary.htna, 40  
summary.htna\_group (summary.htna), 40  
  
tna::betweenness\_network(), 13  
tna::plot.tna\_centralities(), 25