Package 'llrem'

May 19, 2026

Title LLM Relational Event Models

Version 0.1.1

Description Fit Cox proportional hazards relational event models (REMs), including a separable formulation that partitions events into initiation and continuation sub-models. Optionally augments REM simulations with large language model (LLM) agents that select targets conditioned on event history, supporting multiple providers ('OpenAI', 'Anthropic', 'xAI'/'Grok', 'Google Gemini', 'Ollama', 'AWS Bedrock') through a common interface. See Butts (2008) <doi:10.1111/j.1467-9531.2008.00203.x> for description of relational event modeling.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports ggplot2, httr2 (>= 1.1.0), Rcpp, survival

LinkingTo Rcpp

Suggests igraph, parallel, relevent, testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation yes

Author C. Ben Gibson [aut, cre]

Maintainer C. Ben Gibson <cbengibson@gmail.com>

Repository CRAN

Date/Publication 2026-05-19 07:10:08 UTC

Contents

`attach_covariates_df` *Attach exogenous covariates to a pre-built risk-set data frame*

Description

Convenience wrapper for applying a `rem_covariates()` object to the output of `remstats_generate()` after the fact. Useful for testing or for workflows that build the risk set once and experiment with different covariate specifications.

Usage

```
attach_covariates_df(pe, covariates)
```

Arguments

pe	Data frame returned by `remstats_generate()` (or equivalently, the `pe` object inside `fit_rem()`), with a `time` column.
covariates	A `rem_covariates()` object.

Value

`pe` with additional covariate columns appended.

Examples

```
## Not run:
data(edgelist)
el   <- prepare_edgelist(edgelist)
gdp  <- cov_node_static("GDP", data.frame(
  node = sort(unique(c(el$sender, el$receiver))),
  value = runif(length(unique(c(el$sender, el$receiver))))
))
covs <- rem_covariates(gdp)
fit  <- fit_rem(el[1:100, ],
  survival::Surv(time1, occurred) ~ GDP + Repetition +
    survival::strata(time2),
  n_nonevents = 30L, method = "clogit", covariates = covs)

## End(Not run)
```

`compare_rem_models` *Forest plot comparing fitted REM coefficients*

Description

Produces a coefficient plot with 95% confidence intervals for a named list of `coxph` models, with one colour per model source.

Usage

```
compare_rem_models(
  models,
  term_order = NULL,
  palette = NULL,
  title = "REM Coefficient Comparison"
)
```

Arguments

models	Named list of coxph objects.
term_order	Character vector giving the desired y-axis order (bottom to top). Defaults to reverse alphabetical over all terms present.
palette	Named character vector of hex colours keyed to names(models). A colour-blind-friendly default palette is applied when NULL.
title	Plot title string.

Value

A ggplot object (invisibly). The plot is also printed as a side-effect.

Examples

```
## Not run:
data(edgelist)
el <- prepare_edgelist(edgelist)
f  <- survival::Surv(time1, occurred) ~ Repetition + Recencylog +
         survival::strata(time2)
m1 <- fit_rem(el[1:150, ], f, n_nonevents = 30L, method = "cox")
m2 <- fit_rem(el[1:300, ], f, n_nonevents = 30L, method = "cox")
compare_rem_models(list("N=150" = m1, "N=300" = m2))

## End(Not run)
```

---

compute_hazard_surface

Compute the full hazard surface over all directed dyads

Description

Evaluates exp(beta^T * x_ij) for every directed dyad given the current event history. Used internally by run_multiagent_rem() to derive sender activation probabilities and the REM-hazard fallback for failed LLM parses.

Usage

```
compute_hazard_surface(history, base_riskset, beta, covariates = NULL)
```

Arguments

history	Data frame of past events (columns: time, sender, receiver, eveid).
base_riskset	Precomputed base risk set (output of get_base_riskset()).
beta	Named numeric vector of REM coefficients (from coef(rem_fit)).
covariates	Optional exogenous covariate list from rem_covariates().

Value

Data frame with columns `sender`, `receiver`, `hazard`.

Examples

```
data(edgelist)
el   <- prepare_edgelist(edgelist)
fit  <- fit_rem(el[1:100, ],
  survival::Surv(time1, occurred) ~ Repetition + survival::strata(time2),
  n_nonevents = 20L, method = "clogit")
nodes  <- sort(unique(c(el$sender, el$receiver)))
rs     <- expand.grid(sender = nodes, receiver = nodes)
rs     <- rs[rs$sender != rs$receiver, ]
rs$eid <- paste0(rs$sender, "_", rs$receiver)
beta <- c(Repetition = coef(fit)[["Repetition"]])
hz   <- compute_hazard_surface(el[1:5, ], rs, beta)
head(hz)
```

---

compute_seq_stat_means

Compute sufficient-statistic means for a sequence without fitting a REM

Description

Builds covariate values only for the realized events in `edgelist` (n_nonevents = 0L) and returns their column means. Much faster than `fit_rem()` since no Cox model is fitted and the risk set contains exactly one row per event. Use to measure convergence in sufficient-statistic space rather than coefficient space.

Usage

```
compute_seq_stat_means(
  edgelist,
  rem_formula,
  base_riskset = NULL,
  covariates = NULL
)
```

Arguments

`edgelist`	Prepared edgelist (output of `prepare_edgelist()`).
`rem_formula`	Survival formula — used to identify covariate columns.
`base_riskset`	Precomputed base risk set; constructed from `edgelist` if `NULL`.
`covariates`	Optional exogenous covariate list from `rem_covariates()`.

Value

Named numeric vector: mean of each REM covariate across events.

Examples

```
data(edgelist)
el <- prepare_edgelist(edgelist)
compute_seq_stat_means(
  el[1:100, ],
  survival::Surv(time1, occurred) ~ Repetition + RecRecencyLog +
    survival::strata(time2)
)
```

---

cov_dyad_static *Create a static dyad-level covariate*

Description

Adds one column to each risk-set row based on the (sender, receiver) pair.

Usage

```
cov_dyad_static(name, data, warn_missing = TRUE)
```

Arguments

name	Character scalar. Column name added to the risk set.
data	Data frame with columns sender (integer-coercible), receiver (integer-coercible), and value (numeric). Need not be a complete cross-product; unmatched dyads receive 0 (with optional warning).
warn_missing	Logical.

Value

An object of class c("cov_dyad_static", "rem_cov").

Examples

```
alliance <- data.frame(
  sender = c(1L, 2L), receiver = c(2L, 1L), value = c(1, 1)
)
cov <- cov_dyad_static("Alliance", alliance)
```

cov_dyad_temporal *Create a temporal dyad-level covariate*

Description

Like `cov_dyad_static()` but values change at known time breakpoints.

Usage

```
cov_dyad_temporal(name, data, warn_missing = TRUE)
```

Arguments

`name`	Character scalar.
`data`	Data frame with columns `sender`, `receiver`, `time_start`, and `value`. Rows need not be sorted.
`warn_missing`	Logical.

Value

An object of class `c("cov_dyad_temporal", "rem_cov")`.

Examples

```
treaty <- data.frame(
  sender = c(1L, 1L), receiver = c(2L, 2L),
  time_start = c(0, 15), value = c(0, 1)
)
cov <- cov_dyad_temporal("Treaty", treaty)
```

cov_node_static *Create a static node-level covariate*

Description

Adds one or two columns to each risk-set row based on the sender's and/or receiver's value in a node attribute table.

Usage

```
cov_node_static(
  name,
  data,
  role = c("both", "sender", "receiver"),
  warn_missing = TRUE
)
```

Arguments

name	Character scalar. Column name(s) added to the risk set. When role = "both", two columns are added: <name>_snd and <name>_rec. Otherwise a single column named <name> is added.
data	Data frame with columns node (integer-coercible) and value (numeric). One row per node.
role	One of "both" (default), "sender", or "receiver".
warn_missing	Logical. Warn when a node in the risk set has no matching row in data (default TRUE).

Value

An object of class c("cov_node_static", "rem_cov").

Examples

```
gdp <- data.frame(node = 1:4, value = c(1.2, 0.8, 2.1, 0.5))
cov <- cov_node_static("GDP", gdp, role = "sender")
```

---

cov_node_temporal *Create a temporal node-level covariate*

Description

Like cov_node_static() but values change at known time breakpoints. The active value at event time t is the one with the largest time_start <= t (step function).

Usage

```
cov_node_temporal(
  name,
  data,
  role = c("both", "sender", "receiver"),
  warn_missing = TRUE
)
```

Arguments

name	Character scalar.
data	Data frame with columns node (integer-coercible), time_start (numeric), and value (numeric). Rows need not be sorted; sorting is done internally.
role	One of "both" (default), "sender", or "receiver".
warn_missing	Logical.

Value

An object of class `c("cov_node_temporal", "rem_cov")`.

Examples

```
sanctions <- data.frame(
  node = c(1L, 1L, 2L), time_start = c(0, 10, 5), value = c(0, 1, 1)
)
cov <- cov_node_temporal("Sanctioned", sanctions, role = "receiver")
```

---

edgelist *Militarized Interstate Dispute conflict edgelist*

Description

A directed event sequence of militarized conflict acts between sovereign states, drawn from the Militarized Interstate Dispute (MID) dataset. Each row records one country using military force against another.

Usage

```
data(edgelist)
```

Format

A data frame with 1,708 rows and 3 variables:

ev Integer event index (1–1708).

id_snd Integer sender country ID.

id_rec Integer receiver country ID.

Source

Militarized Interstate Dispute (MID) dataset.

`fit_rem` *Fit a Relational Event Model to an edgelist*

Description

Builds risk-set statistics for each event in the sequence, adds survival model timing columns (`time1`, `time2`), and fits a Cox proportional hazards model. All covariate computation is handled internally.

Usage

```
fit_rem(
  edgelist,
  formula,
  n = NULL,
  n_nonevents = 200L,
  base_riskset = NULL,
  cl = NULL,
  method = c("cox", "clogit", "temporal"),
  covariates = NULL
)
```

Arguments

`edgelist`	Data frame with columns `time`, `sender`, `receiver`, `eveid`. Use `prepare_edgelist()` to create this from a raw matrix.
`formula`	A `survival::Surv` formula. Must reference `time1` and `time2` (via `strata()`).

Available covariates

Name	Definition	relevent analogue	No
Dyad activity			
`Repetition`	dyad_occ / sender_outDeg	`FrPSndSnd`	Sa
`SndInitiation`	I(dyad i→j has ever fired)	—	llre
`DyadRecencyLog`	-log1p(time since last i→j)	`RSndSnd`	Sa
`WeightedRep`	w_ij / max(w_ij in risk set)	—	llre
Reciprocity			
`RRecInitiation`	I(reverse dyad j→i has ever fired)	`PSAB-BA` (one-step)	rel
`RecRecencyLog`	-log1p(time since last j→i)	`RRecSnd`	Bo
`RecRecencyRank`	1 / rank(j in i's incoming recency list)	`RRecSnd`	Ex
`WeightedRec`	min(w_ij, w_ji) / max(w_ij, w_ji)	—	Sq
`FrRecSnd`	w_ji / sender_inDeg	`FrRecSnd`	Fra
Degree			
`OutDegSnd`	sender out-degree / (n-1)	`NODSnd`	rel
`OutDegRec`	receiver out-degree / (n-1)	`NODRec`	sa
`InDegSnd`	sender in-degree / (n-1)	`NIDSnd`	sa
`InDegRec`	receiver in-degree / (n-1)	`NIDRec`	sa
`TotDegSnd`	(OutDegSnd + InDegSnd) / 2	`NTDegSnd`	bo

TotDegRec	(OutDegRec + InDegRec) / 2	NTDegRec	bot
Triadic closure			
OTP	out-two-path count	OTPSnd	raw
ITP	in-two-path count	ITPSnd	raw
OSP	out-shared-partner count	OSPSnd	raw
ISP	in-shared-partner count	ISPSnd	raw
NOTPSnd	OTP / max(OTP in risk set)	—	llre
NITPSnd	ITP / max(ITP in risk set)	—	llre
NOSPSnd	OSP / max(OSP in risk set)	—	llre
NISPSnd	ISP / max(ISP in risk set)	—	llre
NTriadic	(OTP + ITP + OSP + ISP) / max(OTP + ITP + OSP + ISP)	—	llre
Imitation			
Imitation	(recInDg - w_ij) / max per sender	—	Tir
PreImitation	recInDg - w_ij (raw, unnormalised)	—	Ra
P-shifts			
IOviolation	PSAB_XA + PSAB_XB + PSAB_XY (any turn-usurp)	PSAB-XA/PSAB-XB/PSAB-XY (separate)	llre
PSAB_BA	I(last event was j→i, this is i→j)	PSAB-BA	exa
PSAB_BY	I(last event was j→i, this is i→y, y!=j)	PSAB-BY	exa
PSAB_AY	I(last event was i→j, this is i→y, y!=j)	PSAB-AY	exa
PSAB_XA	I(last event was x→i, this is j→i)	PSAB-XA	exa
PSAB_XB	I(last event was x→j, this is i→j)	PSAB-XB	exa
PSAB_XY	I(last event was x→y, this is i→j; x!=i, y!=j)	PSAB-XY	exa

`n` Number of nodes. Inferred from `edgelist` if `NULL`.

`n_nonevents` Sampled non-event risk-set size per event. `NULL` uses the full risk set (slow for large networks; 200 is a sensible default).

`base_riskset` Precomputed base risk-set data frame. Constructed from `n` if `NULL`. Pass a cached object when calling `fit_rem()` and `run_llm_rem()` in the same session to avoid redundant work.

`cl` Optional PSOCK cluster from `parallel::makeCluster()` for parallelised risk-set computation.

`method` Estimation method: `"cox"` (default) fits a Cox proportional hazards model via `survival::coxph()` with one stratum per event. `"clogit"` fits a conditional logit (discrete-choice/ordinal) model via BFGS; substantially faster than `"cox"` for large sequences and asymptotically equivalent under the ordinal likelihood. `"temporal"` fits the full interval-censored REM likelihood, which adds a waiting-time penalty `Delta_t * Sigma exp(eta)` to each event's contribution. Requires that `edgelist$time` contains real inter-event gaps, not just ranks; coefficients are no longer scale-free.

Available covariates: see `@param formula` above.

Note on naming vs **relevent**: `OutDegSnd/OutDegRec/InDegSnd/InDegRec` scale by `(n-1)` (fixed maximum degree), unlike relevent's `NODSnd/NIDRec` which scale by the running event count. `DyadRecencyLog` is `-log1p(clock-time of last i→j event)`; relevent's `RSndSnd` is a recency rank – both measure same-dyad recency but are not numerically comparable. `RecRecencyLog` (`-log1p(time since last j→i)`)

and RecRecencyRank (1/rank of j in i's incoming recency list) are both analogues of relevent's RRecSnd; RecRecencyRank is the exact functional equivalent.

covariates — Optional exogenous covariate list created by rem_covariates(). NULL uses only the built-in endogenous statistics.

Value

For method = "cox": a coxph object. For method = "clogit" or "temporal": a rem_clogit object with elements coef, vcov, loglik, and converged.

Examples

```
data(edgelist)
el <- prepare_edgelist(edgelist)
fit <- fit_rem(
  el[1:100, ],
  survival::Surv(time1, occurred) ~ Repetition + RecRecencyLog +
    survival::strata(time2),
  n_nonevents = 20L, method = "clogit"
)
print(fit)
```

---

fit_rem_sep *Fit a Separable Relational Event Model*

Description

Partitions the risk set by prior contact status (SndInitiation) and fits two independent sub-models: one for **initiation** events (sender has never previously contacted receiver; SndInitiation == 0) and one for **continuation** events (sender has contacted receiver before; SndInitiation == 1). The approach is analogous to separable TERGMs (Krivitsky and Handcock, 2014): because the two subsets are disjoint, their parameters are conditionally independent and can be estimated separately via standard Cox partial likelihood on each partition.

Usage

```
fit_rem_sep(
  edgelist,
  formula_init,
  formula_cont,
  n = NULL,
  n_nonevents = 200L,
  base_riskset = NULL,
  cl = NULL,
  method = c("cox", "clogit", "temporal"),
  covariates = NULL,
  min_events = 2L
)
```

Arguments

edgelist	Data frame with columns time, sender, receiver, eveid. Use prepare_edgelist() to create this from a raw matrix.
formula_init	Survival formula for the initiation sub-model (events where SndInitiation == 0). Typical terms: RRecInitiation, NOTPSnd, NITPSnd. Must include survival::strata(time2).
formula_cont	Survival formula for the continuation sub-model (events where SndInitiation == 1). Typical terms: Repetition, RecRecencyLog. Must include survival::strata(time2).
n	Number of nodes. Inferred from edgelist if NULL.
n_nonevents	Sampled non-event risk-set size per event. NULL uses the full risk set. Passed to fit_rem() internally.
base_riskset	Precomputed base risk-set data frame.
cl	Optional PSOCK cluster for parallelised risk-set computation.
method	Estimation method: "cox" (default), "clogit", or "temporal". Applied independently to each sub-model.
covariates	Optional exogenous covariate list from rem_covariates().
min_events	Minimum number of focal events required in each partition to attempt fitting. Partitions below this threshold return NULL for that sub-model with a warning. Default 2.

Details

The combined log-likelihood is loglik_init + loglik_cont, which is directly comparable to a pooled five-parameter model with the same degrees of freedom (see paper for details).

Value

A rem_sep object (a named list) with elements:

init Fitted initiation sub-model (same class as fit_rem() output for the chosen method), or NULL if too few events.

cont Fitted continuation sub-model, or NULL.

loglik Named numeric vector: init, cont, sep (= init + cont), and null (= init_null + cont_null).

n_init Number of initiation events in the sequence.

n_cont Number of continuation events.

init_fraction Proportion of events that are initiations.

method The estimation method used.

Examples

```
data(edgelist)
el <- prepare_edgelist(edgelist)
fit <- fit_rem_sep(
  el[1:200, ],
  formula_init = survival::Surv(time1, occurred) ~
```

```
    RRecInitiation + survival::strata(time2),
  formula_cont = survival::Surv(time1, occurred) ~
    Repetition + RecRecencyLog + survival::strata(time2),
  n_nonevents = 20L, method = "clogit"
)
print(fit)
```

`llm_call` *Call an LLM provider*

Description

Generic dispatch function. Passes a list of role/content message pairs to the provider's internal call function and returns the text response.

Usage

```
llm_call(provider, messages, max_tokens = 15L, temperature = 0.7)
```

Arguments

`provider`	An `llm_provider` object.
`messages`	List of role/content pairs in OpenAI format, e.g. `list(list(role = "system", content = "..."), list(role = "user", content = "..."))`.
`max_tokens`	Token budget for the response.
`temperature`	Sampling temperature.

Value

A character string, or `NA_character_` on failure.

Examples

```
## Not run:
p    <- llm_provider_openai()
msgs <- list(
  list(role = "system", content = "Reply with a single integer."),
  list(role = "user",   content = "What is 2 + 2?")
)
llm_call(p, msgs, max_tokens = 5L)

## End(Not run)
```

`llm_provider_anthropic` *Create an Anthropic Claude LLM provider*

Description

Anthropic's API separates the system prompt as a top-level field and uses a different response shape than the OpenAI format. This constructor handles both translations internally.

Usage

```
llm_provider_anthropic(
  api_key = Sys.getenv("ANTHROPIC_API_KEY"),
  model = "claude-sonnet-4-6",
  anthropic_version = "2023-06-01",
  max_retries = 5L
)
```

Arguments

`api_key`	Defaults to `Sys.getenv("ANTHROPIC_API_KEY")`.
`model`	Default `"claude-3-5-sonnet-20241022"`.
`anthropic_version`	API version header string.
`max_retries`	Maximum request attempts.

Value

An `llm_provider` object.

Examples

```
## Not run:
p <- llm_provider_anthropic()   # reads ANTHROPIC_API_KEY from env
print(p)

## End(Not run)
```

`llm_provider_bedrock` *Create an AWS Bedrock LLM provider*

Description

Uses Bedrock's Converse API with AWS SigV4 signing. Supports any model available in your Bedrock account, including Anthropic Claude and Meta Llama families.

Usage

```
llm_provider_bedrock(
  model,
  region = Sys.getenv("AWS_DEFAULT_REGION", unset = "us-east-1"),
  access_key = Sys.getenv("AWS_ACCESS_KEY_ID"),
  secret_key = Sys.getenv("AWS_SECRET_ACCESS_KEY"),
  session_token = Sys.getenv("AWS_SESSION_TOKEN"),
  name = "AWS Bedrock",
  max_retries = 5L
)
```

Arguments

`model`	Bedrock model ID string (see above).
`region`	AWS region. Defaults to `AWS_DEFAULT_REGION` env var, then `"us-east-1"`.
`access_key`	AWS access key ID. Defaults to `AWS_ACCESS_KEY_ID`.
`secret_key`	AWS secret access key. Defaults to `AWS_SECRET_ACCESS_KEY`.
`session_token`	Optional STS session token. Defaults to `AWS_SESSION_TOKEN` (ignored if unset).
`name`	Display name for print output and plot labels.
`max_retries`	Maximum request attempts.

Details

Credentials are resolved in order: explicit arguments → environment variables (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN`, `AWS_DEFAULT_REGION`).

Common model IDs:

- `"anthropic.claude-haiku-4-5-20251001-v1:0"`
- `"anthropic.claude-3-5-sonnet-20241022-v2:0"`
- `"meta.llama3-70b-instruct-v1:0"`
- `"meta.llama3-8b-instruct-v1:0"`
- `"mistral.mistral-7b-instruct-v0:2"`

Value

An `llm_provider` object.

Examples

```
## Not run:
p <- llm_provider_bedrock(
  model = "anthropic.claude-haiku-4-5-20251001-v1:0"
)   # reads AWS_* credentials from env
print(p)

## End(Not run)
```

`llm_provider_gemini` *Create a Google Gemini LLM provider*

Description

Uses Google's OpenAI-compatible endpoint.

Usage

```
llm_provider_gemini(
  api_key = Sys.getenv("GEMINI_API_KEY"),
  model = "gemini-2.0-flash",
  max_retries = 5L
)
```

Arguments

`api_key`	Defaults to `Sys.getenv("GEMINI_API_KEY")`.
`model`	Default `"gemini-2.0-flash"`.
`max_retries`	Maximum request attempts.

Value

An `llm_provider` object.

Examples

```
## Not run:
p <- llm_provider_gemini()   # reads GEMINI_API_KEY from env
print(p)

## End(Not run)
```

llm_provider_grok *Create an xAI Grok LLM provider*

Description

Create an xAI Grok LLM provider

Usage

```
llm_provider_grok(
  api_key = Sys.getenv("XAI_API_KEY"),
  model = "grok-3",
  max_retries = 5L
)
```

Arguments

api_key	Defaults to Sys.getenv("XAI_API_KEY").
model	Default "grok-3".
max_retries	Maximum request attempts.

Value

An llm_provider object.

Examples

```
## Not run:
p <- llm_provider_grok()   # reads XAI_API_KEY from env
print(p)

## End(Not run)
```

llm_provider_mock *Create a mock LLM provider for testing*

Description

Returns an llm_provider that calls a local strategy_fn instead of making any network requests. Useful for testing pipeline logic and custom query_fn closures without incurring API costs.

Usage

```
llm_provider_mock(strategy_fn, name = "Mock")
```

Arguments

`strategy_fn`	Function with signature `f(messages, max_tokens, temperature) -> character`. The return value must be parseable by the corresponding query factory (a stringified integer for roleplay scenarios; two space-separated integers for the predict scenario).
`name`	Display name for `print()` output.

Details

Three ready-made strategies are provided:

- `mock_strategy_min_id()`: always picks the smallest valid target ID.
- `mock_strategy_max_id()`: always picks the largest valid target ID.
- `mock_strategy_highest_indegree()`: picks the valid target with the highest cumulative in-degree in the event history.

Value

An `llm_provider` object.

Examples

```
p <- llm_provider_mock(mock_strategy_min_id)
print(p)
```

`llm_provider_ollama` *Create a local or remote Ollama LLM provider*

Description

Create a local or remote Ollama LLM provider

Usage

```
llm_provider_ollama(
  model = "llama3.2",
  host = "localhost",
  port = 11434L,
  base_url = NULL,
  max_retries = 3L
)
```

Arguments

model	Model tag, e.g. "llama3.2" or "mistral".
host	Hostname or IP of the Ollama server. Defaults to "localhost". Override to target a remote machine, e.g. host = "192.168.1.65".
port	Port the Ollama server is listening on (default 11434).
base_url	Full endpoint URL. When supplied, host and port are ignored — useful as a manual escape hatch.
max_retries	Maximum request attempts.

Value

An llm_provider object.

Examples

```
## Not run:
p <- llm_provider_ollama(model = "llama3.2")   # requires local Ollama server
print(p)

## End(Not run)
```

---

llm_provider_openai *Create an OpenAI LLM provider*

Description

Create an OpenAI LLM provider

Usage

```
llm_provider_openai(
  api_key = Sys.getenv("OPENAI_API_KEY"),
  model = "gpt-4o",
  max_retries = 5L
)
```

Arguments

api_key	Defaults to Sys.getenv("OPENAI_API_KEY").
model	Default "gpt-4o".
max_retries	Maximum request attempts.

Value

An llm_provider object.

Examples

```
## Not run:
p <- llm_provider_openai()   # reads OPENAI_API_KEY from env
print(p)

## End(Not run)
```

`llm_provider_openai_compat` *Create a generic OpenAI-compatible LLM provider*

Description

Suitable for any API that mirrors the OpenAI chat completions wire format. Used internally by `llm_provider_openai()`, `llm_provider_grok()`, `llm_provider_gemini()`, and `llm_provider_ollama()`.

Usage

```
llm_provider_openai_compat(
  api_key,
  model,
  base_url,
  name = "OpenAI-compatible",
  max_retries = 5L
)
```

Arguments

`api_key`	API key string.
`model`	Model identifier.
`base_url`	Full chat-completions endpoint URL.
`name`	Display name used in `print()` output and plot labels.
`max_retries`	Maximum request attempts before returning `NA`.

Value

An `llm_provider` object.

Examples

```
## Not run:
p <- llm_provider_openai_compat(
  api_key  = Sys.getenv("OPENAI_API_KEY"),
  model    = "gpt-4o-mini",
  base_url = "https://api.openai.com/v1/chat/completions"
)
print(p)
```

```
## End(Not run)
```

`make_behavior_queries` *Generate plain-language behavioral guidelines from empirical sufficient stats*

Description

Translates the mean sufficient statistics for realized conflict events into numbered natural-language instructions suitable for inclusion in an LLM system prompt. Intended to ground LLM predictions in the empirical base rates of each relational pattern.

Usage

```
make_behavior_queries(stat_means, event_value = 1, digits = 0)
```

Arguments

`stat_means`	Data frame returned by `make_stat_means()`: one row per value of `occurred`, one column per REM term.
`event_value`	The value of `occurred` that identifies realized events (default 1).
`digits`	Decimal digits for percentage rounding (default 0).

Value

A single character string: a numbered list of behavioral guidelines.

Examples

```
sm <- data.frame(
  occurred  = c(0, 1),
  Repetition = c(0.05, 0.45),
  IOviolation = c(0.12, 0.08)
)
cat(make_behavior_queries(sm))
```

make_query_predict *Query factory: LLM predicts the next event*

Description

Returns a query_fn closure that shows the LLM the event history and asks it to predict both the sender and receiver of the next event. Tests the model's implicit understanding of network dynamics rather than its own behavioural tendencies.

Usage

```
make_query_predict(cfg)
```

Arguments

cfg	A rem_cfg object.

Value

A function with signature f(past_events, empirical_event, all_nodes) -> c(sender, receiver).

Examples

```
## Not run:
cfg <- rem_cfg(
  provider    = llm_provider_openai(),
  context     = "Conflict events between states.",
  action_verb = "attacked",
  actor_noun  = "Country"
)
query_fn <- make_query_predict(cfg)

## End(Not run)
```

make_query_roleplay_random *Query factory: LLM roleplays as a randomly assigned node*

Description

Returns a query_fn closure that assigns a uniformly random node identity at each step and asks the LLM to choose a receiver. Provides a baseline that isolates how much the assigned identity shapes LLM choices, independent of the empirical sender sequence.

Usage

```
make_query_roleplay_random(cfg)
```

Arguments

cfg	A rem_cfg object.

Value

A function with signature f(past_events, empirical_event, all_nodes) -> c(sender, receiver).

Examples

```
## Not run:
cfg <- rem_cfg(
  provider    = llm_provider_openai(),
  context     = "Conflict events between states.",
  action_verb = "attacked",
  actor_noun  = "Country"
)
query_fn <- make_query_roleplay_random(cfg)

## End(Not run)
```

---

make_query_roleplay_sender

Query factory: LLM roleplays as the empirical sender

Description

Returns a query_fn closure in which the LLM is told it IS the empirical sender at time *t* and asked to choose a receiver. The generated sequence preserves the empirical sender ordering but uses LLM-selected receivers.

Usage

```
make_query_roleplay_sender(cfg)
```

Arguments

cfg	A rem_cfg object.

Value

A function with signature f(past_events, empirical_event, all_nodes) -> c(sender, receiver).

Examples

```
## Not run:
cfg <- rem_cfg(
  provider    = llm_provider_openai(),
  context     = "Conflict events between states.",
  action_verb = "attacked",
  actor_noun  = "Country"
)
query_fn <- make_query_roleplay_sender(cfg)

## End(Not run)
```

`make_stat_means` *Compute per-term mean sufficient statistics for realized events*

Description

Aggregates a risk-set data frame by `occurred` and returns means of the REM covariates, plus the median of `Recency` for realized events (if present).

Usage

```
make_stat_means(rem_formula, poss_edges)
```

Arguments

`rem_formula`	Survival formula.
`poss_edges`	Risk-set data frame containing `occurred` and `Recency`.

Value

A data frame suitable for passing to `make_behavior_queries()`.

Examples

```
## Not run:
data(edgelist)
el  <- prepare_edgelist(edgelist)
f   <- survival::Surv(time1, occurred) ~ Repetition + SndInitiation +
        survival::strata(time2)
# poss_edges is the internal risk-set frame; obtain via fit_rem internals

## End(Not run)
```

make_suffstats *Extract sufficient-statistic means from a REM risk-set data frame*

Description

Returns a data frame with one row per value of `occurred` (0/1), containing the mean of each REM covariate. Used as input to `make_behavior_queries()`.

Usage

```
make_suffstats(rem_formula, data, drop_strata = TRUE)
```

Arguments

`rem_formula`	Survival formula whose RHS terms are the covariates.
`data`	Risk-set data frame (expanded counting-process format) with an `occurred` column.
`drop_strata`	If `TRUE` (default), drop strata columns from the model matrix before computing means.

Value

A data frame with columns `occurred` plus one column per covariate.

Examples

```
data(edgelist)
el  <- prepare_edgelist(edgelist)
fit <- fit_rem(el[1:100, ],
  survival::Surv(time1, occurred) ~ Repetition + survival::strata(time2),
  n_nonevents = 30L, method = "clogit")
# make_suffstats operates on the expanded risk-set data frame; see fit_rem
```

mock_strategy_highest_indegree *Mock strategy: pick the valid target with the highest in-degree*

Description

Parses the event history embedded in the prompt to tally cumulative in-degrees, then returns the valid target node with the highest count. Falls back to the smallest valid ID when no history is available yet.

Usage

```
mock_strategy_highest_indegree(messages, max_tokens, temperature)
```

Arguments

messages	List of message objects passed by the provider interface.
max_tokens	Maximum tokens (ignored by mock; present for interface compatibility).
temperature	Sampling temperature (ignored by mock; present for interface compatibility).

Value

A strategy function for use with `llm_provider_mock()`.

Examples

```
msgs <- list(list(role = "user", content = paste(
  "Event 1: Actor 1 (1) attacked Actor 2 (2).",
  "Event 2: Actor 1 (1) attacked Actor 2 (2).",
  "Valid targets: (2) (3) (4)", sep = "\n"
)))
mock_strategy_highest_indegree(msgs, 15L, 0.7)
```

---

mock_strategy_max_id *Mock strategy: always pick the largest valid target*

Description

Mock strategy: always pick the largest valid target

Usage

```
mock_strategy_max_id(messages, max_tokens, temperature)
```

Arguments

messages	List of message objects passed by the provider interface.
max_tokens	Maximum tokens (ignored by mock; present for interface compatibility).
temperature	Sampling temperature (ignored by mock; present for interface compatibility).

Value

A strategy function for use with `llm_provider_mock()`.

Examples

```
msgs <- list(list(role = "user",
  content = "Valid targets: (2) (3) (4)"))
mock_strategy_max_id(msgs, 15L, 0.7)
```

mock_strategy_min_id *Mock strategy: always pick the smallest valid target*

Description

Mock strategy: always pick the smallest valid target

Usage

```
mock_strategy_min_id(messages, max_tokens, temperature)
```

Arguments

messages	List of message objects passed by the provider interface.
max_tokens	Maximum tokens (ignored by mock; present for interface compatibility).
temperature	Sampling temperature (ignored by mock; present for interface compatibility).

Value

A strategy function for use with `llm_provider_mock()`.

Examples

```
msgs <- list(list(role = "user",
  content = "Valid targets: (2) (3) (4)"))
mock_strategy_min_id(msgs, 15L, 0.7)
```

n_nodes *Number of nodes in the MID edgelist*

Description

Total number of unique country nodes in the bundled MID conflict edgelist (95 countries).

Usage

```
data(n_nodes)
```

Format

An integer scalar.

Source

Derived from the MID dataset bundled in this package.

prepare_edgelist *Normalise a raw edgelist to the standard three-column format*

Description

Accepts a matrix or data frame with columns in order: time, sender, receiver. Returns a data frame with named integer columns and an `eveid` dyad key (`"sender_receiver"`).

Usage

```
prepare_edgelist(raw)
```

Arguments

`raw`	Matrix or data frame with at least 3 columns.

Value

A data frame with columns `time` (integer), `sender` (integer), `receiver` (integer), and `eveid` (character).

Examples

```
el <- prepare_edgelist(data.frame(t = 1:5, s = c(1,2,3,1,2), r = c(2,3,1,3,1)))
head(el)
```

rem_cfg *Create a prompt configuration object*

Description

Bundles everything that affects prompt construction into a single reusable object. One `rem_cfg` can be passed to multiple `make_query_roleplay_sender()`, `make_query_predict()`, and `make_query_roleplay_random()` calls. Swap `provider` to replicate a study with a different model.

Usage

```
rem_cfg(
  provider,
  context,
  action_verb,
  actor_noun,
  node_labels = NULL,
  history_window = 30L
)
```

Arguments

provider	An llm_provider object created by one of the llm_provider_*() functions.
context	One or two sentences describing the network domain in second person (used as the system message).
action_verb	Verb describing a directed event, e.g. "attacked", "emailed", "called".
actor_noun	Capitalised singular noun for nodes, e.g. "Country", "Person", "Actor".
node_labels	Optional named character vector mapping integer node IDs (as strings) to readable labels, e.g. c("1" = "USA", "2" = "Russia"). NULL generates generic "<actor_noun> <id>" labels.
history_window	Maximum number of past events included in each prompt. NULL includes the full history.

Value

A rem_cfg object.

Examples

```
## Not run:
cfg <- rem_cfg(
  provider    = llm_provider_openai(),
  context     = "These are militarized conflict events between states.",
  action_verb = "attacked",
  actor_noun  = "Country",
  history_window = 20L
)
print(cfg)

## End(Not run)
```

---

rem_covariates *Collect covariate objects for use in fit_rem*

Description

Validates that all covariate names are unique, do not collide with endogenous REM column names, and are proper rem_cov objects.

Usage

```
rem_covariates(...)
```

Arguments

...	Any number of objects created by cov_node_static(), cov_node_temporal(), cov_dyad_static(), or cov_dyad_temporal().

Value

An object of class "rem_covariates".

Examples

```
gdp  <- cov_node_static("GDP", data.frame(node = 1:4, value = c(1,2,1,3)))
ally <- cov_dyad_static("Alliance",
          data.frame(sender = 1L, receiver = 2L, value = 1))
covs <- rem_covariates(gdp, ally)
```

---

run_llm_rem *Generate an LLM event sequence and fit a REM*

Description

At each time step t (starting from 2), query_fn is called with the growing LLM-generated history, the empirical event at t, and the full node set. The returned sender/receiver pair is appended to the sequence. The sequence is seeded with the first empirical event so the LLM starts with at least one data point. After generation, risk-set statistics are built and a REM is fitted.

Usage

```
run_llm_rem(
  edgelist,
  formula,
  query_fn,
  n_events = NULL,
  n = NULL,
  n_nonevents = 200L,
  base_riskset = NULL,
  cl = NULL,
  seed = 8214L,
  checkpoint_path = NULL,
  use_empirical_history = FALSE,
  rate_limit_secs = 0.3,
  fit = TRUE,
  method = c("cox", "clogit", "temporal"),
  covariates = NULL
)
```

Arguments

edgelist	Prepared edgelist (output of prepare_edgelist()).
formula	Survival formula passed to fit_rem().
query_fn	A closure produced by make_query_roleplay_sender(), make_query_predict(), or make_query_roleplay_random(), or any function with the signature: f(past_events, empirical_event, all_nodes) -> c(sender, receiver).

n_events	Number of events to generate. NULL uses the full edgelist.
n	Number of nodes. Inferred if NULL.
n_nonevents	Sampled risk-set size passed to fit_rem().
base_riskset	Precomputed base risk set.
cl	Optional PSOCK cluster for parallelised risk-set computation.
seed	RNG seed for random fallbacks and random-node sampling.
checkpoint_path	File path for incremental saves every 10 events. NULL disables checkpointing.
use_empirical_history	If FALSE (default), the LLM is shown its own previously generated events as history, so any divergence from reality compounds over time. If TRUE, the LLM is shown the empirical event history at each step, isolating the effect of the generation strategy from the effect of accumulated sequence drift.
rate_limit_secs	Seconds to pause between LLM calls.
fit	If TRUE (default), fits a REM on the generated sequence before returning. Set to FALSE to skip model fitting and return only the sequence and raw responses — useful when running many sequences and deferring all model fitting to a separate step.
method	Passed to fit_rem(): "cox" (default), "clogit", or "temporal".
covariates	Optional exogenous covariate list passed to fit_rem().

Value

A named list with elements:

- sequence: generated event sequence as a data frame.
- responses: raw LLM response text and parsed sender/receiver per event.
- rem: fitted model (only present when fit = TRUE).

Examples

```
## Not run:
data(edgelist)
el  <- prepare_edgelist(edgelist)
cfg <- rem_cfg(
  provider    = llm_provider_openai(),
  context     = "These are militarized conflict events between countries.",
  action_verb = "attacked",
  actor_noun  = "Country"
)
res <- run_llm_rem(
  el[1:50, ],
  survival::Surv(time1, occurred) ~ Repetition + RecRecencyLog +
    survival::strata(time2),
  query_fn    = make_query_roleplay_sender(cfg),
  n_nonevents = 50L, method = "clogit"
```

```
)
print(res$rem)

## End(Not run)
```

`run_multiagent_rem` *Run a multi-agent REM-LLM conflict simulation*

Description

At each tick, the REM hazard surface determines who acts (sender activation), and individual LLM agents decide against whom (receiver selection). The two components are complementary: the REM enforces endogenous structure; the LLM contributes identity-conditioned content.

Usage

```
run_multiagent_rem(
  edgelist,
  rem_fit,
  cfg,
  n_ticks,
  lambda = NULL,
  base_riskset = NULL,
  cold_start = FALSE,
  seed = 8214L,
  checkpoint_path = NULL,
  covariates = NULL
)
```

Arguments

`edgelist`	Prepared edgelist – used to infer the node set and optionally as the seed event.
`rem_fit`	Fitted `coxph` object (from `fit_rem()`).
`cfg`	A single `rem_cfg()` shared across all agents. Sender identity is injected dynamically at each query.
`n_ticks`	Number of simulation ticks.
`lambda`	Mean events per tick (Poisson rate). Estimated from `edgelist` inter-event times if `NULL`.
`base_riskset`	Precomputed base risk set. Built from `edgelist` if `NULL`.
`cold_start`	If `TRUE`, begin with a uniform hazard surface rather than seeding from the first empirical event. Cold-start runs have a burn-in period before endogenous structure accumulates.
`seed`	RNG seed.
`checkpoint_path`	File path for incremental saves every 10 ticks. `NULL` disables checkpointing. Existing checkpoint is resumed automatically.
`covariates`	Optional exogenous covariate list from `rem_covariates()`. Passed to `fit_rem()` when computing the hazard surface.

Value

A named list:

`sequence` Data frame of generated events (time, sender, receiver, eveid).

`sender_probs` `n_ticks` x `N` matrix of activation probabilities $p_i(t)$ at each tick.

`n_events_per_tick` Integer vector of K_t values.

Examples

```
## Not run:
data(edgelist)
el  <- prepare_edgelist(edgelist)
fit <- fit_rem(el[1:200, ],
  survival::Surv(time1, occurred) ~ Repetition + Recencylog +
    survival::strata(time2),
  n_nonevents = 50L, method = "cox")
cfg <- rem_cfg(
  provider    = llm_provider_openai(),
  context     = "Conflict events between states.",
  action_verb = "attacked",
  actor_noun  = "Country"
)
sim <- run_multiagent_rem(el[1:200, ], fit, cfg, n_ticks = 5L)
head(sim$sequence)

## End(Not run)
```

Index