

Package ‘Gtheory4LLM’

October 11, 2026

Type Package

Title Generalizability Theory for LLM Subjective Tasks

Version 0.5.0

Author Jin Liu [aut, cre, cph]

Maintainer Jin Liu <Veronica.Liu0206@gmail.com>

Description Studies the reliability and generalizability of subjective judgments produced by large language models (LLMs), including annotation, rating, and structured LLM-as-a-judge tasks. Specifies evaluator, prompt, generation, and repeated-run facets through crossed or explicitly nested random sources with configurable item interactions. Fits univariate models, joint Gaussian models, and joint discrete models for binary, ordinal, and unordered categorical outcomes, with source-specific covariance. Gaussian models use exact balanced likelihood; discrete models use a dense first-order Laplace approximation with Gaussian latent random effects. Supported balanced decision studies compare evaluator, prompt, and replication allocations using observed Gaussian or explicitly requested latent binary and ordinal reliability, with random or fixed facets after Brennan (2001). Gaussian fits report asymptotic Wald standard errors for their variance components and delta-method intervals for the coefficients; discrete fits report point estimates only. Provides fixed-layout Gaussian batch projections, bounded cost-aware allocation comparisons, and Gaussian simulation-based pilot precision planning. Scalar nominal reliability and joint Gaussian-discrete fitting are not implemented. Includes three publicly archived LLM annotation datasets covering hate-speech, mental-health, and drug-review tasks. Discrete fitting is limited to small models; the preflight report describes supported designs and computational limits. Generalizability coefficients follow the variance-decomposition framework of Brennan (2001) <[doi:10.1007/978-1-4757-3456-0](https://doi.org/10.1007/978-1-4757-3456-0)>.

License GPL-3

Encoding UTF-8

Depends R (>= 4.5.0)

Imports Matrix (>= 1.6.0), OpenMx (>= 2.22.11), graphics, methods, stats, utils

Suggests lme4, ordinal, knitr, rmarkdown

VignetteBuilder knitr

URL <https://github.com/Veronica0206/Gtheory4LLM>

BugReports <https://github.com/Veronica0206/Gtheory4LLM/issues>

Collate 'design.R' 'batch.R' 'family.R' 'gaussian_retry.R'
 'gaussian_engine.R' 'gaussian.R' 'discrete_response.R'
 'discrete_dense.R' 'discrete_sparse.R' 'discrete_mode.R'
 'discrete_sparse_mode.R' 'discrete.R' 'preflight.R'
 'diagnostics_stages.R' 'fit.R' 'methods.R' 'reliability.R'
 'batch_reliability.R' 'planning.R' 'simulation.R'
 'visualization.R' 'report.R' 'examples.R'

NeedsCompilation no

Repository CRAN

Date/Publication 2026-10-11 04:00:02 UTC

Contents

Gtheory4LLM-package	3
gtheory_datasets	4
gtheory_drug_review	8
gtheory_hate_speech	9
gtheory_mental_health	11
gt_batch	13
gt_batch_dstudy	16
gt_batch_reliability	17
gt_components	20
gt_control	21
gt_design	25
gt_diagnostics	27
gt_dstudy	29
gt_example	32
gt_family	34
gt_fit	35
gt_fit_methods	38
gt_pilot_plan	40
gt_plan	42
gt_preflight	46
gt_reliability	51
gt_report	55
gt_score	57
gt_simulate	58
plot.gt_reliability	59
print.summary.gt_fit	61

Index	63
--------------	-----------

Description

Studies the reliability and generalizability of subjective judgments produced by large language models (LLMs), including annotation, rating, and structured LLM-as-a-judge tasks. Supported balanced decision studies compare projected reliability across evaluator, prompt, and repeated-run allocations.

Details

Use `gt_design()` to declare random sources, `gt_preflight()` to inspect design and resource limits, `gt_family()` to distinguish the observation family, and `gt_fit()` to estimate a model. Inspect `gt_diagnostics()` before interpreting coefficients, then use `gt_dstudy()` to compare an allocation grid with a substantively chosen reliability target.

The modeling interface supports selected item interactions, crossed and explicitly nested sources, and joint multivariate source covariances. Measurement roles are specified by the research design; facet names do not automatically impose nesting or remove item interactions. Traditional method-of-moments G theory already includes random effects and crossed/nested designs. This package integrates design-specific covariance models, outcome-appropriate likelihoods, and supported decision studies; superiority in estimation accuracy requires comparative validation.

Gaussian outcomes use exact balanced likelihood. Binary, ordinal, and unordered categorical outcomes use their specified observation distributions with Gaussian latent random effects, fitted by a dense first-order Laplace approximation. Joint Gaussian models and joint discrete models are supported; a joint Gaussian-discrete likelihood is not implemented.

Observed Gaussian reliability and latent binary or ordinal reliability are distinct estimands. Scalar nominal reliability is not implemented. Decision studies require a complete balanced coded panel. Their projections depend on fitted source covariances and the specified universes of evaluators and prompts. Adding arbitrarily chosen LLMs does not ensure those assumptions hold. Gaussian projections carry delta-method intervals for estimation uncertainty in the fitted source covariances; they propagate component-estimation uncertainty under the declared model, not violations of facet exchangeability or annotation accuracy against human judgments. Coefficients alone do not select a minimum-cost allocation. Discrete projections are point estimates only.

The study-planning functions distinguish pilot item count, annotations per item and items per request. Use `gt_plan` for bounded cost-aware allocation search with explicit user-supplied prices, and `gt_batch_reliability` for fixed-layout Gaussian batch targets. These do not infer annotation accuracy, optimal provider identities or a new batch-size effect from an existing fit.

Three publicly archived LLM annotation panels cover hate-speech, mental-health, and drug-review tasks, with eight outcome sets and fifteen response codings. The [dataset overview](#) and dedicated [Hate-Speech](#), [Mental-Health](#), and [Drug-Review](#) topics describe variables, category mappings, pre-processing, and source and annotation-study references. The package contains design identifiers and modeled annotations from the public OSF deposit, [doi:10.17605/OSF.IO/K9CAJ](https://doi.org/10.17605/OSF.IO/K9CAJ).

Package code is distributed under GPL-3; real annotation data retain the CC BY 4.0 license in the public deposit's codebook. See 'DATA_LICENSE.md'. Numerical acceptance does not establish Laplace approximation adequacy, a global optimum, or empirical validity.

Gaussian fits report asymptotic Wald standard errors for their source variance components and delta-method intervals for G and Phi; see [gt_reliability](#). Model-based Gaussian simulation and pilot precision planning are available through [gt_simulate](#) and [gt_pilot_plan](#); these do not provide calibrated bootstrap intervals. Bootstrap and jackknife intervals are not implemented, and the discrete Laplace engine reports point estimates only.

Why this package declares R ($\geq 4.5.0$): Nothing in this package’s own R code needs R 4.5. The floor comes from the OpenMx version it supports. OpenMx 2.22.11 calls `Rf_isDataFrame`, a C entry point R introduced in 4.5.0, while declaring only R ($\geq 3.5.0$) for itself. Declaring the floor here is what prevents an install on an older R from failing later with an unexplained compilation error. Earlier combinations such as R 4.3 with OpenMx 2.21.11 have been reported to run this package’s checks; that pairing is not part of its validated gate. To use one, install from source with a relaxed floor, or load the implementation directly by sourcing ‘load_functions.R’, which imposes no version requirement of its own.

The vignette “How many evaluators, prompts, and runs?” demonstrates fitting through reliability, intervals, fixed facets, and a decision study using synthetic continuous judgments. Open it with `browseVignettes()`. The installed ‘doc/LLM-workflow.R’ is the same workflow as a script.

Author(s)

Jin Liu (author and maintainer), <Veronica.Liu0206@gmail.com>

References

- Brennan, R. L. (2001). *Generalizability Theory*. Springer. doi:10.1007/9781475734560.
- Jiang, Z., Raymond, M., Shi, D., and DiStefano, C. (2020). Using a linear mixed-effect model framework to estimate multivariate generalizability theory parameters in R. *Behavior Research Methods*, 52, 2383–2393. doi:10.3758/s1342802001399z.

See Also

[gt_design](#), [gt_preflight](#), [gt_fit](#), [gt_dstudy](#), [gt_example](#), [glm](#)

gtheory_datasets

Three LLM Annotation Datasets: Design, Coding, and Sources

Description

Documents the Hate-Speech, Mental-Health, and Drug-Review modeling panels distributed with Gtheory4LLM. The three underlying datasets provide eight outcome sets and fifteen combinations of outcome set and coding through [gt_example](#).

Format

Calling `gt_example(name)` returns a list. Its data element is a data frame containing the five shared factor columns below and the selected outcome column(s). The other elements identify the outcomes, observation families, object and facets, coding, source, and interpretation notes. Installed resources also include `source_provenance`.

item 100 text-object identifiers retained from the corresponding public annotation table.

evaluator Four stored run identifiers:

- `gemini-2.5-flash`
- `gpt-4o-mini`
- `llama-70b` (Llama-3.3-70B)
- `mistral-small` (Mistral-Small-24B)

prompt Three prompt identifiers: `cot`, `minimal`, and `rubric`. The `cot` condition adds a structured reasoning checklist.

temp Six temperature levels: 0, 0.2, 0.4, 0.6, 0.8, and 1. They are stored as factor levels.

seed Three replicate-run identifiers: 1001, 2001, and 3001. A repeated seed does not guarantee identical provider output.

Details

Each dataset contains a 100-item crossed LLM annotation panel from the public OSF annotation deposit (Liu, 2026). Task definitions and annotation protocols are described in the three task-specific studies cited in the dedicated dataset topics. HateXplain, Sentiment Analysis for Mental Health, and Drugs.com reviews supplied the original text material.

Items were selected using screening-pass entropy stratification. Each of the 100 items was evaluated under all combinations of four evaluators, three prompt types, six temperatures, and three seeds, yielding 216 rows per item and 21,600 rows per outcome set. These are repeated measurements on 100 text objects; the row count is not the number of independent texts. The selected items and evaluator panel define the scope of these examples.

The installed resources contain design identifiers and modeled LLM outcomes. The source study's parsing and post-processing are retained, including default labels for omitted items within successfully parsed batches. There are no missing modeling values in the stored panels; this does not establish that every original API response supplied a usable item annotation. Raw texts, prompt text, original corpus reference labels, and API metadata are outside these minimal modeling tables.

Temperature, seed, and the intended universe

Temperature is a controlled setting. Whether it is treated as fixed or random in a reliability analysis depends on the intended universe of generalization. If inference concerns exactly the observed temperature settings, an analyst can declare `gt_reliability(fit, fixed = "temp")`. Generalizing beyond those settings instead requires an explicit population and exchangeability assumptions; selection by the researcher alone does not determine the choice.

The observed agreement among seed replicates changes across temperature. The table gives the proportion of item-by-evaluator-by-prompt cells in which all three seeds return the same label:

temperature	0	0.2	0.4	0.6	0.8	1
Hate-Speech	0.915	0.862	0.837	0.817	0.780	0.733
Mental-Health (nominal)	0.914	0.842	0.835	0.789	0.749	0.678
Drug-Review	0.840	0.767	0.711	0.686	0.631	0.581

These are descriptive observed-label agreement rates. Their differences motivate checking the response and covariance model, but do not by themselves establish that a latent seed variance is heterogeneous. For discrete outcomes, changes in category probabilities can change observed agreement even with constant latent random-effect variance. Agreement is below one at temperature 0, so that setting does not guarantee identical recorded labels across runs.

Declaring `fixed = "temp"` changes the reliability estimand after fitting; it does not change the G-study likelihood or correct covariance misspecification. A scientifically defined analysis within one temperature is one possible sensitivity analysis. In that case, omit the now-constant temperature column from the fitted facet declaration while recording the conditioned setting in the analysis. A model explicitly allowing variation by temperature is another possible approach, provided its implementation and assumptions are justified. Neither treatment is selected automatically from this descriptive table.

Dataset topics and outcome sets

Hate-Speech One ordinal outcome set. See the [Hate-Speech data topic](#) for category definitions and the original corpus and annotation-study references.

Mental-Health Five outcome sets. See the [Mental-Health data topic](#) for working-score mappings, binary flags, unordered labels, and source references.

Drug-Review Two ordinal outcome sets. See the [Drug-Review data topic](#) for holistic sentiment, four aspect definitions, and source references.

Coding and use

The default coding = "native" preserves the binary, ordinal, or unordered categorical endpoint where defined. The Mental-Health 7L and 3L variants remain explicitly Gaussian working scores under both coding options; they are not established clinical severity scales. Seven outcome sets also support coding = "manuscript", which reproduces the original Gaussian working-score analysis. The nominal Mental-Health set has native coding only. The eight sets are alternative views of three panels, not eight independent datasets.

Use `gt_example()` to list the catalog and `gt_example(name)$data` to access a table. Each panel has 21,600 rows, so every native discrete coding exceeds the dense discrete engine's default limits by more than an order of magnitude, and raising the limits does not make the dense engine practical at that size. These are data resources, not full-panel discrete fitting examples: use `gt_preflight()` before requesting a discrete fit, and do not remove item interactions merely to obtain an accepted fit. Discrete demonstrations require a scientifically specified small panel and supported source structure; successful loading alone does not establish fitting feasibility. Numerical model assumptions and coefficient scales are described in [gt_fit](#) and [gt_reliability](#).

Source

Liu's public OSF annotation deposit:

[doi:10.17605/OSF.IO/K9CAJ](https://doi.org/10.17605/OSF.IO/K9CAJ).

The source files are:

- ‘hate_labeling_final.csv’
- ‘mh_labeling_final.csv’
- ‘drug_labeling_final.csv’

Their checksums match the public deposit. Package resources retain only design identifiers and modeled annotations; category types, working-score mappings, and grouped flags are defined by `gt_example()`.

The deposit’s data codebook licenses derived artifacts under CC BY 4.0. The installed supporting files are:

- ‘DATA_LICENSE.md’: attribution and modification details.
- ‘extdata/manifest.csv’: source and resource checksums.
- ‘DATASET_REFERENCES.bib’: the complete task bibliography.

References

Liu, J. (2026). *LLM Annotation Reliability: A Generalizability Theory Analysis of Hate Speech, Mental Health, and Drug Review Tasks*. Open Science Framework, public annotation dataset.

[doi:10.17605/OSF.IO/K9CAJ](https://doi.org/10.17605/OSF.IO/K9CAJ).

The three dedicated dataset topics provide the original corpus and related annotation-study references.

See Also

[gt_example](#), [Hate-Speech data](#), [Mental-Health data](#), [Drug-Review data](#)

Examples

```
gt_example()
x <- gt_example("hate_speech")
dim(x$data)
vapply(x$data[x$facets], nlevels, integer(1))
x$outcomes
system.file("DATASET_REFERENCES.bib", package = "Gtheory4LLM")
```

Description

Two outcome sets from repeated LLM judgments of 100 medication reviews: five-category holistic sentiment and four three-category aspects. The modeling tables are accessed with [gt_example](#).

Format

Each call returns a list whose data element has 21,600 rows. The five shared factor columns are item, evaluator, prompt, temp, and seed: 100 reviews crossed with four evaluators, three prompts, six temperatures, and three replicate seeds. Their stored levels are documented in the [dataset overview](#).

drug_review One score column. Native coding is an ordered factor: 1 = VERY_NEGATIVE, 2 = NEGATIVE, 3 = NEUTRAL, 4 = POSITIVE, 5 = VERY_POSITIVE. These are holistic sentiment categories, even though the source CSV calls its score column severity.

drug_review_4aspect Four outcome columns: efficacy, safety, burden, and cost. Each is an ordered factor with levels NEGATIVE, NEUTRAL, POSITIVE under native coding.

The native families are ordinal with probit links. For either set, coding = "manuscript" returns numerical Gaussian working scores: 1–5 for holistic sentiment and 1–3 for each aspect. The loader also returns outcome names, families, design roles, coding, source, and notes; installed resources include source_provenance.

Details

The aspect annotations summarize the review's sentiment about:

efficacy Whether the treatment helped or worked (positive) or did not work or worsened the problem (negative).

safety Tolerability or absence of side effects (positive) versus side effects or adverse events (negative).

burden Convenience or ease of use (positive) versus inconvenience, adherence difficulty, or regimen complexity (negative).

cost Affordability or coverage (positive) versus expense, coverage denial, or high out-of-pocket costs (negative).

The study's rubric assigned NEUTRAL when an aspect was unclear, unmentioned, or mixed. This category should not uniformly be read as an explicitly neutral patient opinion. Aspects can disagree with the holistic sentiment response.

These are LLM-generated annotations of reviews drawn from the UCI Drug Review corpus, collected from Drugs.com. They are not the original patient-provided 1–10 ratings. The installed tables omit review text, medication and condition metadata, source ratings, probability vectors, and API metadata.

Every modeling row is complete after the source study's preprocessing, which allowed NEUTRAL defaults for items omitted from an otherwise parsed response batch. The package preserves these stored annotations and does not identify defaults separately. The two sets share the same sampled reviews and design; they are not independent datasets. The complete discrete tables exceed current dense backend limits; the examples below only load and summarize them.

Source

Derived from 'drug_labeling_final.csv' in the public OSF annotation deposit (Liu, 2026), [doi:10.17605/OSF.IO/K9CAJ](https://doi.org/10.17605/OSF.IO/K9CAJ). Gräßer et al. (2018) describe the source drug-review data; Liu (2026) describes the related aspect-level LLM annotation study. The common G-theory source is cited in the [dataset overview](#).

References

Gräßer, F., Kallumadi, S., Malberg, H., and Zaunseder, S. (2018). Aspect-Based Sentiment Analysis of Drug Reviews Applying Cross-Domain and Cross-Data Learning. *Proceedings of the 2018 International Conference on Digital Health*, 121–125. [doi:10.1145/3194658.3194677](https://doi.org/10.1145/3194658.3194677).

Liu, J. (2026). Learning to Fuse Aspect-Level LLM Annotations for Low-Quality Ordinal Sentiment Supervision. *SSRN working paper*, 6244519; revised June 17, 2026.

[doi:10.2139/ssrn.6244519](https://doi.org/10.2139/ssrn.6244519).

See Also

[Dataset overview](#), [gt_example](#), [gt_family](#)

Examples

```
holistic <- gt_example("drug_review")
str(holistic$data$score)
table(holistic$data$score)

aspects <- gt_example("drug_review_4aspect")
str(aspects$data[aspects$outcomes])
table(aspects$data$efficacy, aspects$data$safety)
```

Description

An ordinal outcome set containing repeated LLM judgments of 100 HateXplain text items. The installed package includes the modeling table and source provenance, accessed through [gt_example](#).

Format

The loader returns a list; its data element is a data frame with 21,600 rows and six columns. Each item has 216 measurements in the complete item-by-evaluator-by-prompt-by-temperature-by-seed panel.

item Factor identifying the 100 text items.

evaluator, prompt, temp, seed Factors identifying the four evaluators, three prompt conditions, six temperatures, and three replicate seeds. Their stored levels and interpretation are documented in the [dataset overview](#).

score With coding = "native", an ordered factor with levels 1, 2, 3: NORMAL, OFFENSIVE, and HATE, respectively. With coding = "manuscript", the same values are numeric Gaussian working scores.

The remaining list elements identify the outcome, family, object, facets, coding, source, and interpretation notes; installed resources also provide source_provenance.

Details

The native family is ordinal with a probit link. The category order expresses the study's normal/offensive/hate distinction; using numeric manuscript scores additionally treats the category steps as equally spaced.

These outcomes are the repeated LLM annotations used in the G-theory study, not the original HateXplain crowdworker labels. The data table contains design identifiers and the modeled outcome only; original posts, crowdworker labels, probability vectors, prompts, and API metadata are not included.

Every modeling row has a recorded value after the original study's preprocessing. That pipeline could assign NORMAL to items omitted from an otherwise parsed response batch. The package preserves the stored outcomes and does not separately identify such defaults. Completeness of the modeling table therefore does not imply that every original response was complete.

The full table exceeds the current dense discrete fitting limits. The examples below describe the data without fitting the full panel; see the [dataset overview](#) for the shared design and fitting scope.

Source

Derived from 'hate_labeling_final.csv' in the public OSF annotation deposit (Liu, 2026), [doi:10.17605/OSF.IO/K9CAJ](https://doi.org/10.17605/OSF.IO/K9CAJ). HateXplain is the original text corpus (Mathew et al., 2021); Liu (2026) describes the related rubric-conditioned annotation study. These references concern corpus and annotation provenance, respectively. The common G-theory source is cited in the [dataset overview](#).

References

- Mathew, B., Saha, P., Yimam, S. M., Biemann, C., Goyal, P., and Mukherjee, A. (2021). HateXplain: A Benchmark Dataset for Explainable Hate Speech Detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(17), 14867–14875. [doi:10.1609/aaai.v35i17.17745](https://doi.org/10.1609/aaai.v35i17.17745).
- Liu, J. (2026). Rubric-conditioned large language model labeling: Agreement, uncertainty, and label consistency in subjective text annotation. *Computers in Human Behavior*, 181, 108988. [doi:10.1016/j.chb.2026.108988](https://doi.org/10.1016/j.chb.2026.108988).

See Also

[Dataset overview](#), [gt_example](#), [gt_family](#)

Examples

```
hate <- gt_example("hate_speech")
dim(hate$data)
str(hate$data)
table(hate$data$score)

# The historical numeric coding represents the same stored judgments.
working <- gt_example("hate_speech", coding = "manuscript")
str(working$data$score)
```

gtheory_mental_health *LLM Mental-Health Text Annotations in Five Outcome Sets*

Description

Five representations of repeated LLM annotations of the same 100 text items from the Sentiment Analysis for Mental Health corpus: unordered labels, six binary flags, three binary groups, and two historical working-score codings. Load each modeling table with [gt_example](#).

Format

Each call returns a list whose data element has 21,600 rows. The five shared factor columns are item, evaluator, prompt, temp, and seed: 100 items crossed with four evaluators, three prompts, six temperatures, and three replicate seeds. Stored factor levels are documented in the [dataset overview](#). Additional columns depend on the requested set:

mental_health_nominal One label column, an unordered factor with levels NORMAL, STRESS, ANXIETY, DEPRESSION, BIPOLAR, PERSONALITY_DISORDER, and SUICIDAL. The categorical family uses a softmax link and NORMAL as reference. Only coding = "native" is available.

mental_health_6flag Six integer 0/1 columns:

- depression
- anxiety
- suicidal
- stress
- bipolar
- personality_disorder

A value of one indicates the corresponding LLM-assigned evidence flag. Native families are binary with probit links.

mental_health_3group Three integer 0/1 columns:

- stress: the original stress flag.

- `anxiety_depression`: anxiety OR depression.
- `bipolar_personality_suicidal`: the logical OR of bipolar, personality-disorder, and suicidal flags.

The groups may co-occur; they are not mutually exclusive categories. Native families are binary with probit links.

mental_health_7L One integer score column: NORMAL = 1, STRESS = 2, ANXIETY = 3, DEPRESSION = 4, BIPOLAR = 5, PERSONALITY_DISORDER = 6, SUICIDAL = 7. This is a Gaussian working score under both coding options.

mental_health_3L One integer score column: NORMAL or STRESS = 1; ANXIETY or DEPRESSION = 2; BIPOLAR, PERSONALITY_DISORDER, or SUICIDAL = 3. This is a Gaussian working score under both coding options.

The loader also returns outcome names, families, design roles, coding, source, and interpretation notes; installed resources include `source_provenance`.

Details

The 7L and 3L codings reproduce study-defined numerical analyses. They do not establish a clinical severity order or equal distances between mental-health categories. The nominal set preserves the unordered label identity. The six flags describe requested textual evidence and are not clinical diagnoses or a set of mutually exclusive classifications. The three-group binary set differs from the single-outcome 3L working score.

For the six-flag and three-group sets, `coding = "manuscript"` retains the same 0/1 values but declares Gaussian working-score families. The two working-score sets remain Gaussian even with `coding = "native"`.

All sets reuse LLM-generated annotations of the same sampled items, not independent samples or the original corpus's reference status labels. Original texts, corpus reference labels, probability vectors, and API metadata are not bundled. The stored tables have no missing modeling values after the original preprocessing, which allowed NORMAL defaults for items omitted from an otherwise parsed batch. Defaults are not separately identified in these modeling resources. See the [dataset overview](#) for common provenance and the full-panel discrete fitting limit.

Source

Derived from `'mh_labeling_final.csv'` in the public OSF annotation deposit (Liu, 2026).

[doi:10.17605/OSF.IO/K9CAJ](https://doi.org/10.17605/OSF.IO/K9CAJ).

Sarkar's corpus is the original text source. Liu's public annotation study describes the associated multi-view labeling work; its current title is listed below. The common G-theory source is cited in the [dataset overview](#).

References

Sarkar, S. (n.d.). *Sentiment Analysis for Mental Health*. Kaggle dataset; accessed September 12, 2026. The original source URL is retained in the installed task bibliography.

Liu, J. (2025). LLM-Based Annotation as Multi-View Supervision for Affective Text Classification: Soft Labels, Multi-Task Decomposition, and Entropy Diagnostics. *SSRN working paper*, 5964738; revised March 19, 2026. [doi:10.2139/ssrn.5964738](https://doi.org/10.2139/ssrn.5964738).

See Also

[Dataset overview](#), [gt_example](#), [gt_family](#), [gt_score](#)

Examples

```
nominal <- gt_example("mental_health_nominal")
str(nominal$data$label)
table(nominal$data$label)

flags <- gt_example("mental_health_6flag")
str(flags$data[flags$outcomes])
table(flags$data$depression, flags$data$anxiety)

groups <- gt_example("mental_health_3group")
str(groups$data[groups$outcomes])
working <- gt_example("mental_health_3L")
table(working$data$score)
```

gt_batch	<i>Declare That Items Were Annotated Several to a Call</i>
----------	--

Description

Declares that the object of measurement was submitted in calls of several items, so that items sharing a call are not independent. Without a declaration every function treats items as independent given the declared sources.

Usage

```
gt_batch(size, order = NULL, by = NULL, sequential = FALSE,
         neighbor = FALSE, id = NULL)
```

Arguments

size	The number of items in a call: one integer of at least 2. With id, the most a call may hold.
order	Optional name of a numeric column giving the order in which items were submitted. Without id, each item has one distinct value, the same in every condition, and NULL uses the order in which items first appear in the data. With id, it gives the position of each row's item in its call, and NULL uses the stored order of the rows of each call.
by	Optional name of an instrumentation facet across whose levels the dependence may differ, for example the evaluator.
sequential	FALSE, or how many preceding items may affect an item: a whole number from 1 to size - 1. TRUE means every preceding item in the call.

neighbor	FALSE, or how many positions apart two items may be and still affect each other, in either direction: a whole number from 1 to size - 1. TRUE means any distance within the call.
id	Optional name of a column identifying the call that produced each row. NULL, the default, infers the batches from the item order.

Details

A declaration is passed to `gt_design` and travels with the design to every function that receives it or a fit made from it.

By default items in one call are assumed to affect each other equally, whatever their order, and items in different calls are independent given the declared sources. That the dependence varies with position is a separate, optional claim, and two patterns fit it. In a sequential pattern an item is affected by items submitted before it. In a neighbour pattern items close together affect each other in either direction. Data in which every item keeps the same position cannot always tell the two apart, so each is declared on its own, and declaring one asks for it to be examined rather than asserting it.

Inferred and recorded calls. Which items shared a call is either inferred or recorded. Without `id` it is inferred: the items present in the data are cut into batches of size in their order, the last holding what remains, and a batch is the same items, in the same order, in every condition. That is all that data without a call column allow, and it is a reconstruction. An item removed after collection moves every later item into another position and possibly another batch, and nothing in the data shows it; calls of other sizes and regrouping between conditions cannot be seen either. With `id` the calls are read from the data. They may then hold different items in different conditions and fewer items than size; a call may not span conditions, hold an item twice, or hold more than size items. Record the call when collecting data, and prefer `id` whenever that record exists.

Recorded calls are still read from the rows supplied, and three things follow. A call is counted by the items of it that are present, so a call that was sent size items and lost one afterwards looks the same as a call sent one item short. A call with no row left is not counted at all. And a position is a rank among the items present in a call: an order column holding 1, 3 and 4 gives positions 1, 2 and 3. Positions are the positions at which items were sent only when nothing was removed after collection, so a distance between two positions is not yet a distance in the call as it was sent. Keep the unfiltered rows, or the number of items sent in each call, if that distinction will matter.

What a fit does with a declaration. A Gaussian fit models the declaration when the batches form a complete factorial with the facets: the same batches in every condition, every batch at the declared size, at least two batches, and one row in each cell. It then estimates one shared call effect, a random effect for each batch under each condition that its items share equally. The effect is reported as the source `Call` beside the declared sources, with its standard error; with several outcomes it has a covariance between them, structured through covariance like any other source. The likelihood is the exact balanced one, in which an item is a batch-by-slot cell; which slot an item occupies has no effect of its own.

Any other declaration is recorded and audited but not modelled: batches of unequal size, batches that differ between conditions, more than one row in a cell, a single batch, and every binary, ordinal or unordered outcome. The fit then treats items in one call as independent and says so, with the reason. `sequential`, `neighbor` and `by` are recorded and not yet estimated in either case. `gt_preflight` reports beforehand which of the two a fit will do.

Coefficients. A call effect is shared by the items of one call and by no others, so its part in relative error depends on which items share calls. The ordinary `gt_reliability` and `gt_dstudy` weights do not encode that layout and refuse such a fit. Use `gt_batch_reliability` or `gt_batch_dstudy` for explicit item-score, contrast or aggregate targets conditional on the fitted equal fixed Gaussian layout. Those projections are point estimates and do not generalize to regrouped items or a new batch size.

Every result says what happened to a declaration. Fits and diagnostics print the status, the data-frame exports of coefficients carry it as `batch_status`, and `gt_report` records it. It takes one of three values:

- "not_declared": the design declares no batch.
- "declared_not_modelled": items in one call are treated as independent, for the reason given.
- "modelled": the fit estimates the shared call effect.

Value

A `gt_batch` list holding the size, the order column, the facet named in `by`, the two reaches as integers with 0 for a pattern that is off, the call column, and the assumption they state in words.

See Also

`gt_design`, `gt_preflight`

Examples

```
calls <- gt_batch(20)
calls$assumption
local <- gt_batch(20, by = "evaluator", sequential = 3, neighbor = TRUE)
c(sequential = local$sequential, neighbor = local$neighbor)

d <- expand.grid(item = 1:40, evaluator = 1:3, prompt = 1:2)
d$score <- sin(d$item) + d$evaluator / 5 + cos(d$item * d$prompt) / 4
design <- gt_design("item", c("evaluator", "prompt"), batch = calls)
report <- gt_preflight(d, "score", design)
report$batch_audit[c("membership", "batches", "conditions", "calls")]

# Calls recorded in the data are read, not reconstructed.
d$call <- paste(d$evaluator, d$prompt, (d$item - 1) %/% 20)
recorded <- gt_design("item", c("evaluator", "prompt"),
  batch = gt_batch(20, id = "call"))
gt_preflight(d, "score", recorded)$batch_audit[c("membership", "calls")]
```

Description

Apply `gt_batch_reliability` to a bounded grid of instrumentation counts, retaining the same item grouping and the same explicitly weighted targets.

Usage

```
gt_batch_dstudy(fit, grid, contrasts = NULL, aggregate = NULL, score = NULL)
## S3 method for class 'gt_batch_dstudy'
print(x, ..., digits = 4L, max_rows = 12L)
## S3 method for class 'gt_batch_dstudy'
as.data.frame(x, row.names = NULL,
  optional = FALSE, ...)
```

Arguments

<code>fit</code>	An accepted shared-call Gaussian fit with a recorded fixed layout.
<code>grid</code>	A nonempty data frame of named positive integer instrumentation counts, with at most 1000 rows. Unspecified facets retain fitted counts. Batch size and item count cannot be varied.
<code>contrasts</code>	Named item contrast weights, as in <code>gt_batch_reliability</code> .
<code>aggregate</code>	Named aggregate item weights, as in <code>gt_batch_reliability</code> .
<code>score</code>	Optional <code>gt_score</code> outcome weights.
<code>x</code>	A <code>gt_batch_dstudy</code> object.
<code>digits</code>	Significant digits for printing.
<code>max_rows</code>	Maximum number of printed rows.
<code>row.names</code>	Optional row names for tabular export.
<code>optional</code>	Accepted for generic compatibility; column names are preserved.
<code>...</code>	Reserved for additional print or data-frame arguments.

Details

Every candidate uses the target and assumptions documented in [gt_batch_reliability](#). Name target weights using the item keys in `gt_batch_reliability(fit)$layout$item`. Numeric keys preserve the original values and can differ from ordinary printed numeric labels. All instrumentation facets remain random; this function cannot improve a projection by changing the universe to fixed facets. It neither generates allocations nor optimizes costs. It returns all supplied candidates, including repeated rows. No intervals are created. At most one million source-contribution rows are permitted, bounding the returned output size, checked before computing the first candidate. Annotations per item and total calls must remain below 2^{53} for exact integer accounting.

Value

A `gt_batch_dstudy` list with `summary` and `source_contributions` tables carrying a `design_id` referring to the corresponding row of `grid`. The `grid` includes all fitted instrumentation facets with unspecified counts filled in. The item layout, target weights, original counts, batch size, point-only uncertainty record and interpretation are retained. `as.data.frame()` returns the summary with `allocation_<facet>` columns joined by `design_id`, including unchanged fitted counts for unspecified facets. The prefix prevents facet names from colliding with summary or context columns. Additional columns retain `pilot_items`, `batch_size`, `annotations_per_item`, `projected_calls`, `extrapolated`, `scale`, `uncertainty_available`, `uncertainty_reason`, and interpretation through a CSV roundtrip. `extrapolated` identifies allocations with any instrumentation count larger than its fitted count. `Grid`, `interpretation` and `uncertainty` attributes are retained for compatibility. CSV does not preserve those attributes or the full item layout and target weights; retain the original list, for example with `saveRDS()`, when those details are needed to reproduce a weighted target.

See Also

[gt_batch_reliability](#), [gt_dstudy](#)

Examples

```
candidates <- expand.grid(rater = 2:4, repetition = 1:3)
# gt_batch_dstudy(fit, candidates,
#   contrasts = list(pair = c(item1 = 1, item2 = -1)))
```

`gt_batch_reliability` *Project Gaussian Score Error Conditional on a Fixed Batch Layout*

Description

Compute annotation error for absolute item scores, named item contrasts, and weighted aggregates under an accepted shared-call Gaussian model. The calculation contracts source kernels directly without forming an observation covariance matrix.

Usage

```
gt_batch_reliability(fit, counts = NULL, contrasts = NULL,
  aggregate = NULL, score = NULL)
## S3 method for class 'gt_batch_reliability'
print(x, ..., digits = 4L, max_rows = 12L)
## S3 method for class 'gt_batch_reliability'
as.data.frame(x, row.names = NULL,
  optional = FALSE, ...)
```

Arguments

fit	An accepted Gaussian <code>gt_fit</code> that models equal, fixed batches and records its item-to-batch layout. Older saved fits without this record must be refitted; membership is never guessed.
counts	Optional named positive integer instrumentation counts. Unspecified facets retain their fitted counts. Item count and batch size cannot be changed. All instrumentation facets remain random.
contrasts	A numeric vector named by item identifiers and summing to zero, or a named list of such vectors. Omitted items have weight zero. For example, <code>list(within = c(item1 = 1, item2 = -1))</code> declares an item difference. Use the item keys in the returned <code>layout\$item</code> ; see Details.
aggregate	A numeric vector named by item identifiers, or a named list of such vectors. Every vector must have nonnegative weights summing to one. Omitted items have weight zero; weights are not normalized.
score	Optional <code>gt_score</code> weights for a composite across outcomes. Per-outcome results are also returned.
x	An object returned by this function.
digits	Significant digits for printing.
max_rows	Maximum number of rows printed.
row.names	Optional row names for the exported data frame.
optional	Accepted for generic compatibility; column names are preserved.
...	Reserved for additional print or data-frame arguments.

Details

Item keys. Call the function without target weights to obtain `layout$item`, then use these keys to name contrast and aggregate weights. Character identifiers and factor labels are unchanged. Plain integer and double identifiers use distinct character keys with 17 significant digits, independent of the `digits`, `scipen`, and `OutDec` options. Numeric keys always use a decimal point, including in comma-decimal locales. Reuse these keys as character strings; do not convert them back to numbers. These keys can differ from ordinary printed labels, particularly for large or fractional numbers. Positive and negative zero identify the same item. The fit retains the original typed identifiers for membership checks, including when observation retention is disabled. Row order does not define target identity.

Target and scope. The universe score is the object main effect. Each final item score averages a complete panel of all instrumentation conditions with equal weights. The fitted object grouping is held fixed. Facet counts may change under the model's exchangeability assumptions, but every planned condition must retain that same item grouping. The helper does not generalize over randomized regrouping or predict a change in shared-call variance when batch size changes. It does not support fixed instrumentation facets, unequal observation weights, incomplete panels, within-cell replication, or declared positional or facet-varying batch patterns that the fitted model did not estimate.

Error contractions. For a supplied item-weight vector w , a source containing the object contributes its variance times $\sum_i w_i^2$; an instrumentation-only source contributes its variance times $(\sum_i w_i)^2$.

Each is divided by the product of planned counts of the instrumentation facets in its grouping term. Residual variance is multiplied by $\sum_i w_i^2$ and divided by the number of annotations per item. The shared-call contribution is its variance times $\sum_b (\sum_{i \in b} w_i)^2$, divided by the number of annotations per item. For a multivariate score, each source variance is the corresponding quadratic form in score's outcome weights.

Thus a shared call shift cancels from a same-batch item difference, remains in a between-batch difference, and remains in an absolute item score. A weighted aggregate depends on how its weights are distributed across batches. Instrumentation-only sources remain in an aggregate even as the number of averaged items increases. There is no universal batch penalty.

Interpretation of ratios. Each target has an explicit coefficient name:

- Absolute item scores: `Phi`.
- Item contrasts: `contrast_reliability`.
- Weighted aggregates: `aggregate_dependability`.

Each ratio divides the universe variance of its exact weighted target by that variance plus annotation-error variance. A contrast ratio is not a single G coefficient for every possible item ranking. Aggregate error conditions on the selected items and excludes corpus-sampling uncertainty. It is not uncertainty about an unknown population mean. Zero total variance returns NA.

All results are point projections; no sampling standard errors or confidence bounds are provided. The reported `error_sd` is the model-implied annotation-error standard deviation. It is not a standard error of an estimated coefficient.

At most one million source-contribution rows are permitted, checked before materializing target results. Nonfinite quadratic forms and numerically unrepresentable weight scales are refused; rescale extreme target weights.

The fitted layout and returned target weights contain item identifiers. Treat these objects as analysis data. Portable `gt_report(fit)` output does not include the layout; this result is not silently inserted into it.

Value

A `gt_batch_reliability` list. `summary` has one row per target and outcome, plus requested composites, with universe variance, error variance, error standard deviation, ratio and its explicit name. `source_contributions` records each source's role, kernel contraction, divisor and variance contribution. `layout` and `target_weights` record the item keys, grouping and weighted targets. Other fields record fitted and planned facet counts, pilot-item count, annotations per item, projected calls, unchanged batch size, extrapolation, scale and interpretation. `uncertainty$available` is `FALSE`. `as.data.frame()` returns `summary` and retains counts, interpretation and uncertainty as attributes; CSV output does not retain those attributes.

See Also

[gt_batch](#), [gt_batch_dstudy](#), [gt_reliability](#), [gt_score](#)

Examples

```
# Weights identify the target explicitly; use fitted item IDs in an analysis.
within_pair <- c(item1 = 1, item2 = -1)
```

```
corpus_weights <- c(item1 = 0.25, item2 = 0.25, item3 = 0.5)
# gt_batch_reliability(fit, contrasts = list(within = within_pair),
#   aggregate = list(corpus = corpus_weights))
```

gt_components

Extract Source Covariances or Correlations

Description

Returns source-specific matrices on their modeled scale without transforming latent variances into observed-score variances.

Usage

```
gt_components(fit, correlation = FALSE, tolerance = 1e-8)
```

Arguments

fit	A <code>gt_fit</code> object.
correlation	Whether to return correlation matrices instead of covariances.
tolerance	Nonnegative relative variance threshold for reporting correlations. Entries involving a variance at or below this threshold times the reference variance scale are NA.

Details

Gaussian correlation checks use observed outcome variances as the reference scales. When those are unavailable, the largest variance within each source matrix is the reference. Covariances for discrete outcomes refer to identified latent predictors or category contrasts. Correlations involving negligible source variances are suppressed rather than presented as stable estimates.

Value

A named list of matrices, with scale and interpretation attributes. Correlation extraction adds the threshold and boundary policy.

See Also

[gt_fit](#), [gt_diagnostics](#)

Examples

```
d <- expand.grid(item = 1:12, rater = 1:3)
d$score <- sin(d$item) + d$rater / 5 + cos(d$item * d$rater) / 4
design <- gt_design("item", "rater", random = ~ item + rater)
fit <- gt_fit(d, "score", design,
  control = gt_control(gaussian = list(check_hessian = FALSE, retry_seed = 42)))
gt_components(fit)
```

Description

Collects backend controls in separate named lists, together with which optional components a fitted object keeps. Backend-specific names and values are checked when a model is fitted.

Usage

```
gt_control(gaussian = list(), discrete = list(), retain = list())
```

Arguments

gaussian	Uniquely named list of Gaussian settings, described below.
discrete	Uniquely named list of discrete settings, described below.
retain	Uniquely named list of logical retention switches, described below. Every default is TRUE.

Details

Retained components. retain decides which optional parts of a fitted object are kept. All four default to TRUE; dropping a component is always something the caller asked for.

- **data:** the modelled data frame. Dropping it removes every copy of the observations from the fit, including the summary copy the Gaussian OpenMx model carried and the data argument of the recorded call, which is replaced by a marker. Compact item-to-batch design labels remain on a supported shared-call fit so that `gt_batch_reliability()` can evaluate named targets; this switch does not de-identify the fitted design. Portable reports exclude these labels.
- **model:** the Gaussian OpenMx model and its factorial cross-products, and for a discrete fit the conditional predictors and probabilities held for every observation. The OpenMx model carries its own copy of the observations until data is dropped, so with them this is usually the largest component. The observed outcome variances are kept either way, so the correlations that `gt_components()` can report are unaffected.
- **retry_log:** the optimizer transcript and the bulky per-attempt payloads. Each attempt's label, error, optimizer code and message, and objective are kept, so `gt_diagnostics()` reports the same decisions and reasons.
- **session:** the `sessionInfo()` record. Keep it for reproducible research; it is worth its size when a result must be traced to an environment.

None of these components is used to compute an estimate, a diagnostic decision, a coefficient, or a decision study. `gt_reliability()` and `gt_dstudy()` work from the fitted source covariances, the resolved design, and either the retained data or the compact panel summary every fit records. With `data = TRUE` the balanced-panel rules are checked against the retained data, so a panel edited after fitting is still caught; with `data = FALSE` the same two rules are applied to the recorded summary. The fit keeps a retained record of what was dropped.

Gaussian controls and their defaults are:

- optimizer: CSOLNP.
- max_iterations: 3000.
- tolerance: 1e-12.
- check_hessian: TRUE.
- threads: 1.
- silent: TRUE.
- extra_tries: 9.
- retry_seed: NULL.
- start: NULL (automatic moment-based covariance starts).
- max_preparation_bytes: 512 * 1024^2.

The allocation guard estimates preparation memory, excluding caller data and downstream OpenMx objects. An explicit retry seed preserves the caller's random-number state. Gaussian start accepts a complete named list of covariance matrices for all model sources, including Residual. Fully named matrix axes are matched independently to the outcome names; both unnamed axes use outcome order. Incomplete, duplicate, or incompatible labels are rejected. Admissible matrices receive the engine's existing interior eigenvalue floor. Diagonal models retain their repaired diagonal entries, and a pooled residual starts from the mean of those entries. These values initialize estimated parameters; they do not fix their fitted values.

Discrete resource limits are:

- max_random_dimension: 200.
- max_observations: 1200.
- max_parameters: 80.
- max_dense_bytes: 512 * 1024^2.

Discrete optimizer controls are:

- maxit: 150.
- inner_maxit: 60.
- inner_tol: 1e-7.
- reltol: 1e-7.
- start_sd: 0.4.
- trace: 0.
- optimizer: "L-BFGS-B"; alternatively "nllminb".
- start: NULL (automatic response and covariance starts).
- covariance_parameterization: "auto".

The default "auto" uses direct nonnegative variance coordinates for univariate models and for joint models whose covariance is "diagonal". Joint models with unstructured covariance use "log_cholesky", retaining all cross-outcome covariance parameters. Explicit "variance" and "log_cholesky" selections remain available. Direct variance coordinates admit exact zeros; an explicit "variance" request with joint unstructured covariance is rejected rather than removing covariances. The variance upper bound is $\exp(10)$, matching the largest diagonal variance allowed

by the log-Cholesky bounds. `start_sd` still denotes a starting standard deviation and must be positive.

Discrete `start` is a finite, uniquely named numeric vector of optimizer parameters, with complete or partial overrides of the automatic starts. Names are matched exactly; unknown names and values outside the model bounds are rejected. Automatic starts are also checked against the bounds before fitting; the optimizer never silently projects an invalid starting vector. Labels that produce ambiguous parameter names are rejected. The fitted `fit$parameters` vector can initialize a refit of the same model, with unchanged family links, category coding, covariance structure, and covariance parameterization. These are optimizer coordinates: ordinal threshold parameters contain the first threshold followed by log positive gaps; log-Cholesky diagonal parameters are log standard deviations; direct-variance parameters are variances. Actual and automatic initial vectors are retained in the diagnostics, and `fit$starting_parameters` records the primary start. Alternative starts perturb fixed-effect coordinates from the primary start and use the automatic covariance starts with distinct deterministic scale perturbations. If clipping would duplicate an earlier start, a bounded fixed-effect perturbation makes the additional trial distinct.

The selected discrete optimizer remains fixed for primary fits, alternatives, and tight refinements; the engine never silently switches algorithms. `maxit` limits iterations per optimization. For `nlminb`, the function evaluation budget is $\max(200, 10 * \text{maxit})$, and `reltol` supplies its relative objective tolerance. Each optimizer uses its own native stopping rules; the same independent numerical-acceptance checks apply afterward. Optimizer choice is not evidence of approximation accuracy.

Zero is a natural lower boundary in direct variance coordinates, not an artificial optimizer floor. Within `gt_diagnostics(fit)$diagnostics`, the `zero_variance_parameters` field lists exact zeros. These estimates must still pass independent covariance-scale stationarity and restart checks. All upper bounds and fixed-effect/threshold bounds remain artificial and still cause rejection on contact. This mode does not establish approximation accuracy or sampling properties for boundary estimates. The diagnostics field `covariance_parameterization_requested` records the control selection; `covariance_parameterization` records the resolved coordinates ("variance", "log_cholesky", or "fixed").

Discrete numerical-acceptance controls are:

- `stationarity_tol`: 1e-3.
- `validation_reltol`: 1e-10.
- `validation_inner_tol`: 1e-9.
- `alternative_starts`: 1.
- `stability_objective_tol`: 1e-6.
- `stability_parameter_tol`: 0.02.
- `bound_tol`: 1e-4.

Setting alternative starts to zero disables numerical acceptance, not just an optional report. The discrete optimization-attempt budget is $2 * (1 + \text{alternative_starts})$: one preliminary and one tight fit for the primary start and each alternative. Final-mode and stationarity evaluations are recorded separately and are not additional optimization trials. Gaussian `extra_tries` and discrete `alternative_starts` therefore have different meanings. Covariance derivative accuracy uses at most six step halvings when needed, without changing the stationarity or derivative-disagreement tolerances. Every stationarity evaluation, including the center and all fixed-effect and covariance perturbations, must have a valid, converged conditional mode and usable finite likelihood. A failed

evaluation ends the diagnostic and records a validation error; an optimizer penalty is never treated as a derivative.

For a known-covariance calculation, `fixed_covariance` can provide one positive-semidefinite matrix for every discrete random source. Fully named matrices are matched and reordered to the latent dimensions; unnamed matrices are positional. Partial or incompatible dimension names are rejected. These matrices are fixed assumptions rather than estimated source covariances. The covariance parameterization setting is not used when all covariances are fixed.

The attempts entry of `gt_diagnostics(fit)$diagnostics` retains starts, controls, optimizer codes, messages, errors, warnings, elapsed time, and returned parameters. A computational failure during refinement, a restart, or stationarity checking prevents numerical acceptance but preserves an otherwise usable primary or alternative fit for inspection. Unusable optimizer returns are retained unchanged as `raw_result`, separately from the computational placeholder. The selected attempt, optimizer, trial count, and budget are also reported. If no fitted candidate supplies a usable conditional mode, a `gt_discrete_numerical_failure` error condition retains the attempt records instead. Invalid input still fails before fitting. Warnings captured during numerical attempts are retained in these diagnostics; user interrupts are not swallowed.

`max_dense_bytes` bounds the dense algebra that the other discrete limits imply, and is checked before anything is allocated. The estimate covers the random-design matrix, the conditional Hessian and its factor, the slices of the design matrix that forming that Hessian materializes, the linear predictor and gradient, and a planning multiplier for copy-on-modify. It is not peak resident memory: it excludes the caller's data, the optimizer's own state, and the allocation the finite-difference stationarity pass repeats, so it underestimates what a fit really uses. Its purpose is to refuse the obviously impossible with a message naming the observation count, the random dimension, and the size of the random-design matrix, rather than to certify that a permitted model is comfortable. `gt_preflight()` reports the same estimate and its breakdown under `$resources`. Raising this limit makes neither the dense algebra practical nor the approximation accurate.

Optimizer completion, numerical acceptance, and approximation adequacy are separate. Tighter numerical tolerances do not establish the adequacy of a first-order Laplace approximation.

Value

A `gt_control` object with separate Gaussian and discrete control lists and a resolved retention record.

See Also

[gt_diagnostics](#), [gt_fit](#)

Examples

```
gt_control(gaussian = list(retry_seed = 123, check_hessian = FALSE),
  discrete = list(max_random_dimension = 100, alternative_starts = 1))
# Auto selects boundary-capable coordinates when the covariance model permits.
gt_control(discrete = list(covariance_parameterization = "auto"))
gt_control(discrete = list(optimizer = "nllminb", alternative_starts = 2))
# A lean fit for a large panel: estimates, diagnostics, reliability and
# decision studies are unchanged; the stored copies of the data are not kept.
gt_control(retain = list(data = FALSE, model = FALSE))
```

gt_design

*Declare a Generalizability-Theory Random-Source Design***Description**

Declares an object of measurement, any number of instrumentation facets, selected object interactions, and explicit parent-scoped nesting. This constructor preserves the requested source terms; fitting validates the observed data and resolves any residual alias.

Usage

```
gt_design(object, facets, crossed = facets, nested = NULL,
  item_interactions = "complete", instrument_interactions = "complete",
  item_nested = character(), random = NULL, full_cell = TRUE,
  replicates = 1L, max_terms = 4096L, batch = NULL)
```

Arguments

object	One column name for the object of measurement.
facets	Unique instrumentation column names, excluding the object.
crossed	A nonempty subset of facets allowed to form direct object interactions. Defaults to all facets.
nested	A named list giving the instrumentation parents of every non-crossed facet. Multiple parents define their joint group. Ancestors are expanded recursively; cycles are rejected.
item_interactions	"complete", "additive", or the maximum number of crossed instrumentation facets in an object interaction. The object itself is excluded from this order. Additive adds individual object-by-crossed-facet interactions.
instrument_interactions	"complete", "additive", or a maximum interaction order among crossed instrumentation facets. Declared nested groups remain present.
item_nested	Nested child names whose complete ancestor-expanded group should also interact with the object. This does not implicitly add other ancestor-by-object interactions.
random	Optional explicit one-sided formula or character vector of source terms. Formula syntax is limited to names, +, :, *, parentheses, and 0/1. Cannot be combined with explicitly supplied automatic selectors.
full_cell	Retain the object-by-all-facets source when the requested expansion contains it. FALSE removes exactly that one term and leaves every other requested source unchanged. Unlike the other selectors this may be combined with random.
replicates	Positive integer observations per complete object-by-facets cell. Observed replication is checked during fitting.
max_terms	Positive integer bound on source expansion, checked before expansion.

`batch` Optional `gt_batch` declaration that items were annotated several to a call. NULL, the default, leaves items independent given the declared sources. Its by must name a declared facet, and its order and id columns other than the object and the facets.

Details

The object main effect is always present. With all facets crossed, complete object and instrumentation interactions retain the full hierarchy. At one observation per cell, a Gaussian full-cell source is indistinguishable from a free residual and is combined with it during fitting. A requested observation-level source is rejected for discrete families, because it cannot be separated from the identified discrete observation model. It is never dropped silently: declare the reduced model with `full_cell = FALSE`, which removes only that source, or write the intended terms with `random`.

Nesting defines parent-scoped groups shared across objects. It does not establish that a physical sampling design is balanced or identifiable. Arbitrary design declaration does not imply that every backend supports every resulting model.

Balanced coded panel. Gaussian fitting, `gt_reliability()` and `gt_dstudy()` all require one. It is the complete Cartesian product of the observed coded levels of the object and of every instrumentation facet, carrying exactly replicates rows in each of those cells, with no missing outcome. Nothing is imputed, aggregated or dropped to reach it.

Coded is what makes a nested facet fit this description. A nested child is identified by its code within its parent, so child codes repeat across parents and the Cartesian product stays complete; the child's real identity is then the child-by-parent source, not the child column on its own. Globally unique child labels describe the same study but leave most child-by-parent cells structurally empty, and this backend rejects that panel instead of guessing which cells were never possible. The example below shows both codings of one six-item design.

Value

A `gt_design` list containing the object, facets, requested terms and their members, nesting meta-data, and pending data-validation information.

See Also

[gt_fit](#), [gt_family](#), [gt_batch](#)

Examples

```
complete <- gt_design("item", c("evaluator", "prompt", "temp", "seed"))
length(complete$terms_requested)
selected <- gt_design("item", c("evaluator", "prompt", "temp", "seed"),
  crossed = c("evaluator", "prompt"),
  nested = list(temp = c("evaluator", "prompt"), seed = "temp"))
selected$terms_requested
reduced <- gt_design("item", c("rater", "occasion"), random = ~ item + rater)
reduced$terms_requested
# The same crossed design without the observation-level source, which a
# binary or ordinal fit at one observation per cell cannot identify. This is
# where a first discrete fit should start; see the worked script in
```

```
# system.file("examples", "discrete-first.R", package = "Gtheory4LLM").
discrete_ready <- gt_design("item", "rater", full_cell = FALSE)
discrete_ready$terms_requested

# Globally unique child labels versus within-parent codes, for six items
# nested in two questions and rated by four people.
coded_cells <- function(data, columns)
  prod(vapply(data[columns], function(x) length(unique(x)), integer(1)))
physical <- expand.grid(person = 1:4, item = 1:6)
physical$question <- 1 + (physical$item > 3)
# Unique labels 1-6 imply 48 coded cells but supply 24: item 4 never appears
# in question 1, so the coded panel is structurally incomplete.
c(rows = nrow(physical),
  cells = coded_cells(physical, c("person", "item", "question")))
physical$item_code <- physical$item - 3 * (physical$question - 1)
# The within-parent code describes the same six items as a complete panel.
c(rows = nrow(physical),
  cells = coded_cells(physical, c("person", "item_code", "question")))
nested_design <- gt_design("person", c("item_code", "question"),
  crossed = "question", nested = list(item_code = "question"))
nested_design$terms_requested # item_code:question identifies the six items
nested_design$nested_groups
```

gt_diagnostics

*Inspect Numerical Acceptance and Backend Diagnostics***Description**

Reports optimizer completion separately from numerical acceptance and likelihood-approximation adequacy.

Usage

```
gt_diagnostics(fit)
## S3 method for class 'gt_diagnostics'
print(x, ...)
## S3 method for class 'gt_diagnostic_stages'
print(x, ...)
## S3 method for class 'gt_diagnostic_stage'
print(x, ...)
```

Arguments

fit	A <code>gt_fit</code> object.
x	A <code>gt_diagnostics</code> , <code>gt_diagnostic_stages</code> or <code>gt_diagnostic_stage</code> object.
...	Reserved for additional print arguments.

Details

Gaussian diagnostics include independent likelihood reconstruction, covariance-scale stationarity, boundary components, and optional Hessian diagnostics. Discrete diagnostics include covariance- and parameter-scale stationarity, bound locations, and tighter-tolerance alternative-start stability. A discrete fit also retains evaluations: how many objective evaluations the optimizer made, how many it could not use, how many valid conditional solves ended at the inner iteration budget, and, for the first eight invalid evaluations and for later solve- or factor-validity events until eight such events are held, the parameters they were asked about and the measurements each evaluation itself reported. These counts and records cover the optimizer's logged evaluations; a final check or a stationarity probe that returns a refusal keeps its own record with that check. Matrices are never retained in the fit. Setting the environment variable `GTHEORY_DISCRETE_SPECIMEN_DIR` to a directory writes one specimen file per retained failure and per refused start, holding the failed operation's matrix, right-hand side, step and factor where the failure produced them, its parameters and measurements, and the numerical environment, but no observations. The conditional-mode stage reports the three counts, and the count of validity events, among its measurements. Passing these checks does not prove structural identification, a global optimum, or adequate approximation in sparse or strongly dependent data.

`selected_attempt` identifies the returned candidate using the engine's recorded trial identifier. It is NA when no trial is recorded as having produced the returned fit. `acceptance_failures` gives reasons for rejecting the returned fit; `attempt_failures` is a two-column data frame of recorded attempt identifiers and error or rejection reasons, including earlier attempts. An empty attempt-failure table means no recorded trial reported an error or a rejection reason; it is not a statement about trials that were never run.

`standard_errors_available` is TRUE only for a Gaussian fit whose numerically differentiated Hessian was computed and is positive definite. The Gaussian engine forms the parameter covariance matrix as $2 * H^{-1}$ for the $-2 \log L$ fit function and maps it to source covariance entries by the delta method; `fit$uncertainty` retains that matrix, the entry Jacobian, and the entry covariance matrix used by `gt_reliability()`. The discrete Laplace engine computes no observed information, so its standard errors are always unavailable and its coefficients remain point estimates. A component at a variance boundary is flagged in `component_standard_errors`: Wald theory does not apply there, and the reported number is not a usable standard error.

`boundary_sources` describes natural zero or nearly singular covariance components and does not by itself imply rejection. `parameter_bounds` lists artificial discrete-parameter bound contacts, which do prevent numerical acceptance. These shared fields summarize existing engine decisions without repeating or changing numerical checks. `summary(fit)` prints a concise version while retaining full diagnostics for programmatic inspection.

Printing a `gt_diagnostics` object reports the same decisions in the same words. It reads recorded fields only: no check is re-run, no rejection is re-evaluated, and every field remains available in the returned list.

Value

A `gt_diagnostics` list with estimator, engine, optimizer and numerical acceptance status, approximation status, engine-specific diagnostics, a staged summary, resolved aliases, data-validation notes, and batch: whether the design declared batches and whether the fit modelled them, their size, whether membership was inferred or recorded, the reason a declared batch was not modelled, and the declared options not yet estimated.

The `stages` element summarises the numerical checks a fit passed through, so that a rejection names the stage responsible without the caller reading private optimizer records. Each stage reports status, a reason and supporting measurements. The status is one of "passed", "failed", "not_assessed" for a check that did not run or does not apply to this engine, or "inconclusive" for one that ran and neither clearly passed nor clearly failed. Absent evidence is reported as "not_assessed" and never as a pass. The summary is derived from evidence the fit already retained: it never refits, and it never changes an acceptance decision. In particular, optimizer completion reports what the optimizer reported; a retained attempt that never moved from its starting values is exposed through `max_abs_parameter_change` rather than by redefining completion. Shared diagnostic fields include:

- `optimizer`: the selected optimization engine.
- `optimization_trials`: the recorded trial count.
- `selected_attempt`: the returned candidate's identifier.
- `acceptance_failures`: reasons for rejecting the returned fit.
- `attempt_failures`: recorded trial errors or rejections.
- `boundary_sources`: zero or nearly singular covariance sources.
- `parameter_bounds`: artificial parameter-bound contacts.
- `standard_errors_available`: whether a usable parameter covariance matrix was obtained.
- `standard_errors_unavailable_reason`: why it was not, when it was not.
- `component_standard_errors`: per-source variance standard errors with a boundary flag.

See Also

[gt_control](#), [gt_fit](#)

Examples

```
d <- expand.grid(item = 1:12, rater = 1:3)
d$score <- sin(d$item) + d$rater / 5 + cos(d$item * d$rater) / 4
design <- gt_design("item", "rater", random = ~ item + rater)
fit <- gt_fit(d, "score", design,
  control = gt_control(gaussian = list(check_hessian = FALSE, retry_seed = 42)))
gt_diagnostics(fit)$numerically_accepted
print(gt_diagnostics(fit))
```

Description

Computes supported G and Phi coefficients over a grid of instrumentation counts, validating the fitted context once.

Usage

```

gt_dstudy(fit, grid, scale = NULL, score = NULL, fixed = character(),
  level = 0.95)
## S3 method for class 'gt_dstudy'
print(x, ..., digits = 4L, max_rows = 12L)
## S3 method for class 'gt_dstudy'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
gt_dstudy_target(x, target, coefficient = "Erho2", outcome = NULL,
  kind = "outcome")
## S3 method for class 'gt_dstudy'
plot(x, coefficient = "Erho2", interval = TRUE, ...,
  target = NULL, outcome = NULL, kind = NULL)

```

Arguments

fit	A numerically accepted gt_fit with a complete balanced coded panel.
grid	A nonempty data frame of positive integer facet counts. Columns name declared facets; omitted facets retain their fitted counts.
scale	"observed" for Gaussian outcomes or explicitly "latent" for binary/ordinal outcomes.
score	Optional composite weights created by gt_score().
fixed	Instrumentation facets treated as fixed, as in gt_reliability(). A fixed facet may not appear in grid: its universe is exactly its observed levels.
level	Confidence level for the reported coefficient intervals.
x	A gt_dstudy object.
coefficient	Plot or screen "Erho2" (default) or "Phi".
interval	Draw the interval bounds as vertical segments when the fit supplies them.
target	One finite coefficient threshold from 0 to 1, chosen and justified by the user. Required for screening. Optional in plotting, where NULL omits the reference line.
outcome	One exact reported outcome name. Screening requires it when more than one outcome exists within the selected kind. Plotting defaults to all outcomes.
kind	"outcome" for measured outcomes or "composite" for the weighted composite. Screening defaults to "outcome". Plotting defaults to both kinds; specify the kind when an explicitly selected outcome label also names a composite.
row.names	Optional row names for the exported data frame.
optional	Accepted for generic compatibility. Exported column names are always preserved without syntactic name repair.
digits	Significant digits used when printing.
max_rows	Maximum number of coefficient rows to print.
...	Additional arguments passed to graphics::plot(). A named argument such as col, pch, ylim, xlab or ylab replaces the corresponding default rather than duplicating it. Reserved in the print and data-frame methods.

Details

Tabular export. `as.data.frame()` returns an ordinary data frame joining each coefficient row to its allocation by the recorded `design_id`, even for multivariate results or reordered coefficient rows. It preserves coefficient-row order. `kind` remains "outcome" or "composite"; a measured outcome named "composite" is distinct from a weighted composite. Export fields follow the reliability data-frame method documented in [gt_reliability](#). All columns are atomic vectors suitable for `write.csv()`; additional R attributes must be retained separately when exporting to CSV.

Screening supplied candidates. `gt_dstudy_target()` returns an ordinary data frame with every supplied allocation for the selected outcome and kind, preserving coefficient-row order. The added fields are:

- `coefficient` and `target`: the chosen criterion.
- `estimate`: the selected coefficient value.
- `meets_target`: the point-estimate comparison.
- `fewest_measurements`: the qualifying minimum-count flag.

`meets_target` compares the point estimate with the target using $\geq$. It does not screen a confidence limit. An unavailable estimate gives NA, not FALSE. Among candidates known to meet the target, `fewest_measurements` marks all ties at the smallest measurement count, including duplicated supplied allocations. Unknown coefficients retain NA in this flag. If no known candidate meets the target, no row is marked TRUE. The `target_context` attribute records the chosen coefficient, target, outcome, kind, scale, point-estimate comparison, restriction to supplied candidates, and batch status. Other export context is retained, including the `batch_status` column: a screen of a design that declared batches is a screen of coefficients that do not model them.

Counts at or above 2^{53} , or overflowing to infinity, cannot be ranked as exact integer totals. Their `measurement_count_exact` flag is FALSE. If no qualifying candidate has a smaller exactly represented total, `fewest_measurements` remains NA for these qualifying candidates rather than reporting a potentially false tie. If a qualifying exact small total exists, these larger totals cannot be the minimum.

This helper does not generate allocations, refit models, use monetary costs, or find a global optimum. The target is a criterion applied to conditional point projections, not a guarantee of reliability in a new panel. Examine available intervals, extrapolation flags, and the model's generalization assumptions alongside the screen. Latent discrete coefficients remain point estimates with no intervals.

Plotting. By default allocations are unconnected points against measurements per object; different allocations can have the same count. Colours identify outcome/composite series, and open symbols distinguish allocations exceeding fitted facet counts from filled symbols within them. An optional target appears as a dashed horizontal line and is included in automatic vertical limits. Explicit graphical settings remain authoritative, including `pch` and `ylim`. For a design that declared batches with [gt_batch](#), the plot states above the plotting region that they are not modelled, whatever subtitle is supplied. Series filters do not change the returned object: plotting returns the full original study invisibly.

Balanced coded panel. This is the complete Cartesian product of the observed coded levels of the object and of every instrumentation facet, carrying exactly the declared number of rows in each of those cells, with no missing outcome. For a nested facet, *coded* means the child's within-parent code rather than a globally unique label, so the child's identity is the child-by-parent source; [gt_design](#)

works both codings through. A panel that is not balanced in this sense has no analytic coefficient here, and none is approximated.

These are conditional design projections using fitted source covariances. They do not refit the model or establish that larger allocations preserve the same source distributions. Reported intervals propagate estimation uncertainty in the fitted source covariances under the declared random-effects sampling model. They are not prediction intervals for the realized scores of a newly drawn panel, and they do not quantify failure of exchangeability when extending to new facet levels. Boundary and conditional-inference caveats are retained from the fitted model. Intervals are unavailable for discrete fits. Restrictions and coefficient definitions are those of `gt_reliability()`.

A nested child's count may be projected like any other random facet. Its own source and every source containing it are then divided by the joint count of child and parent, while sources built from the parent alone keep the parent count: adding children therefore reduces less error than adding an equal number of levels of a fully crossed facet. A fixed facet may not appear in `grid`, but its random children still may.

Value

A `gt_dstudy` list with completed allocation rows, coefficient results including standard errors and interval bounds, measurements per object, scale, fixed facets, the uncertainty record, extrapolation flags, and fitting diagnostics. The print method shows allocations and coefficients without expanding fitting diagnostics. Both methods return the object invisibly.

See Also

[gt_reliability](#), [gt_score](#)

Examples

```
d <- expand.grid(item = 1:12, rater = 1:3)
d$score <- sin(d$item) + d$rater / 5 + cos(d$item * d$rater) / 4
design <- gt_design("item", "rater", random = ~ item + rater)
fit <- gt_fit(d, "score", design,
  control = gt_control(gaussian = list(check_hessian = FALSE, retry_seed = 42)))
study <- gt_dstudy(fit, data.frame(rater = c(2, 3, 6)))
as.data.frame(study)
gt_dstudy_target(study, target = 0.80, coefficient = "Phi")
plot(study, coefficient = "Phi", target = 0.80)
```

gt_example

Load Modeling Tables from Three LLM-Annotated Datasets

Description

Lists eight documented outcome sets or loads their design columns and modeled outcomes. Installed resources omit raw text and API metadata.

Usage

```
gt_example(name = NULL, coding = c("native", "manuscript"),
           directory = NULL)
```

Arguments

name	Outcome-set name, or NULL to list available sets.
coding	Use "native" to preserve response types. Use "manuscript" for the seven original Gaussian working-score codings.
directory	Optional directory containing example RDS resources or source CSV files, supplied explicitly as one nonempty path. A file named <name>.rds takes precedence over CSV loading. An installed package always loads bundled RDS resources when this argument is omitted, regardless of workspace variables.

Details

Every set retains 21,600 measurement rows: 100 text objects by 4 evaluators, 3 prompts, 6 temperatures, and 3 seeds. Full tables exceed default dense discrete fitting limits; use a predeclared small panel or a future scalable backend for discrete examples.

Available outcome sets are:

- hate_speech: ordinal severity.
- mental_health_7L: original Gaussian working score.
- mental_health_3L: original Gaussian working score.
- mental_health_6flag: six binary flags.
- mental_health_3group: three binary flag groups.
- drug_review: ordinal holistic sentiment.
- drug_review_4aspect: four ordinal aspects.
- mental_health_nominal: seven unordered categories.

The nominal set has no manuscript-score coding. Detailed variable definitions, category mappings, preprocessing notes, and the corresponding annotation-paper and original-corpus references are provided in the [Hate-Speech](#), [Mental-Health](#), and [Drug-Review](#) data topics. The [dataset overview](#) documents the shared panel design, provenance, and installed bibliography.

Standalone functions can use a local `.gt_example_data_dir` binding to configure the CSV directory. This binding must be in the same environment into which the functions were sourced. The source loader configures the checkout's public `inst/extdata` directory. Parent environments and callers are not searched. This standalone convention has no effect on the installed package; use `directory` for an intentional override. An invalid explicit directory produces an error rather than silently falling back to bundled data.

The mental-health 7L/3L scores are study-defined working scores, not established clinical severity scales. Labels are model-generated and retain the source study's preprocessing; no new human adjudication is implied.

The source CSV files are:

- 'hate_labeling_final.csv'

- ‘mh_labeling_final.csv’
- ‘drug_labeling_final.csv’

Bundled data retain design identifiers and modeled outcomes only. Source checksums, the public OSF deposit link, and extraction details accompany installed resources. The annotation data retain the CC BY 4.0 license stated in the public data codebook; see the installed ‘DATA_LICENSE.md’.

Value

A catalog data frame when name is NULL. Otherwise a list with data, outcome names, observation families, object and facet names, example name, coding, source, and interpretation notes. Installed resources also provide portable source-provenance metadata.

See Also

[Dataset overview](#), [Hate-Speech data](#), [Mental-Health data](#), [Drug-Review data](#), [gt_family](#), [gt_fit](#)

Examples

```
gt_example()
x <- gt_example("hate_speech")
nrow(x$data)
is.ordered(x$data$score)
nominal <- gt_example("mental_health_nominal")
levels(nominal$data$label)
```

gt_family	<i>Specify a Gaussian, Binary, Ordinal, or Unordered Categorical Outcome</i>
-----------	--

Description

Keeps observation families distinct and defines their links and category interpretation.

Usage

```
gt_family(family = "gaussian", link = NULL, levels = NULL,
reference = NULL)
```

Arguments

family	One of "gaussian", "binary", "ordinal", or "categorical".
link	Gaussian uses identity; binary and ordinal use probit (default) or logit; unordered categorical uses softmax.
levels	Distinct category labels. Binary levels are negative then positive. Ordinal levels give the substantive order. Categorical levels identify categories without assigning an order. Gaussian outcomes have no category levels.
reference	One unordered categorical reference category. Defaults during fitting to the first declared category.

Details

Numeric binary responses must be 0/1. Ordinal and categorical families require at least three categories; use the separate binary family for two categories. Ordinal order must come from declared levels or an ordered factor. Every declared category must occur in the fitted data. Numeric Gaussian outcomes are never constructed by silently converting factors.

Categorical random effects use reference-category contrasts. An unstructured covariance can be transformed under a reference change; a diagonal contrast covariance imposes a model restriction that depends on the reference category.

Value

A `gt_family` object describing the observation distribution.

See Also

[gt_fit](#)

Examples

```
gt_family()
gt_family("binary", link = "logit")
gt_family("ordinal", levels = c("low", "middle", "high"))
gt_family("categorical", levels = c("red", "blue", "green"), reference = "blue")
```

gt_fit

Fit Univariate or Joint Multivariate Generalizability Models

Description

Fits selected random-source covariance models using an exact balanced Gaussian likelihood or a dense first-order Laplace likelihood for discrete outcomes.

Usage

```
gt_fit(data, outcomes, design, family = gt_family("gaussian"),
  estimator = NULL, covariance = "unstructured", residual = NULL,
  control = gt_control())
## S3 method for class 'gt_fit'
print(x, ...)
## S3 method for class 'gt_fit'
summary(object, ...)
```

Arguments

data	Data frame containing the object, all declared instrumentation columns, and outcomes. Missing values are not silently dropped.
outcomes	One or more unique outcome column names.
design	A declaration created by <code>gt_design()</code> .
family	One <code>gt_family</code> object for all outcomes, or a named list matching outcomes exactly. Multiple discrete families can be modeled jointly. Joint Gaussian-discrete likelihoods are not implemented.
estimator	Gaussian: "REML" (default) or "ML". Discrete: "ML_Laplace".
covariance	"unstructured" or "diagonal"; Gaussian also accepts named source overrides, with unspecified sources using diagonal covariance. Discrete source covariance uses the common requested structure.
residual	The Gaussian residual structure. Choose "unstructured", "diagonal", or "pooled"; pooled gives equal outcome variances. Omit for discrete outcomes.
control	A <code>gt_control</code> object.
x, object	A fitted <code>gt_fit</code> object for the print or summary method.
...	Further method arguments; currently unused.

Details

The Gaussian backend requires a complete coded Cartesian panel with one observation per cell and at most 4096 factorial strata. A design that declares equal fixed batches with `gt_batch` adds one source, `Call`, shared by the items of a batch under one condition, and one factorial axis, because an item is then a batch-by-slot cell; see `gt_batch` for when a declaration is modelled. Implicit normalized Helmert contrasts avoid quadratic allocation along a large object axis. Gaussian ML profiles one fixed intercept per outcome; REML includes the unscaled-intercept constant $D * \log(N)$. Gaussian ML AIC counts covariance parameters and profiled means. Its generic BIC uses N response vectors; `ml_BIC_scalar_scores` explicitly provides the alternative $N * D$ convention. `legacy_AIC` and `legacy_BIC` preserve the archive's variance-only parameter convention. For REML the fit's AIC and BIC elements are NA; `AIC()` and `BIC()` use the restricted likelihood with covariance parameters only, `BIC()` over N response vectors. The fit records the `AIC()` value, the `BIC()` value, and the alternative $N * D$ scalar-score BIC under explicit names, in that order:

- `reml_AIC_variance_parameters`
- `reml_BIC_response_vectors_variance_parameters`
- `reml_BIC_scalar_scores_variance_parameters`

Discrete models use binary links, ordinal thresholds, or unordered reference-category softmax contrasts. They do not estimate a free Gaussian observation residual. Probit latent residual variance is identified as 1; logit as $\pi^2 / 3$. Unsupported observation-level sources, aliased free source kernels, missing categories, and resource-limit violations are rejected.

Returned binary probabilities evaluate each tail directly, preserving small representable probabilities and the declared category order. These are conditional probabilities at the random-effect mode, not population-marginal probabilities.

Gaussian fitting also computes asymptotic Wald standard errors for the source variance components from the numerically differentiated restricted- or profile-likelihood Hessian. These are large-sample quantities conditional on the declared model; they do not establish coverage in small designs and do not apply to a component resting on a variance boundary. `check_hessian = FALSE` disables them along with the other derivative diagnostics.

Numerical acceptance includes optimizer completion and independent stationarity, bound, and tighter-tolerance restart checks. First-order stationarity does not establish a global optimum. A numerically accepted Laplace fit still requires an assessment of approximation adequacy for the intended analysis. Reliability and decision studies reject fits that fail numerical acceptance.

Value

A `gt_fit` object containing source covariance matrices, estimates, likelihood and parameter counts, the resolved design and observation families, modeled data, and numerical diagnostics. Gaussian objects also retain the OpenMx model, prepared sufficient statistics, an uncertainty record holding the parameter covariance matrix and its delta-method mapping to source covariance entries, and a `component_standard_errors` table. Discrete objects include thresholds or category contrasts and conditional probabilities evaluated at the random-effect mode. `print()` returns the fit invisibly. `summary()` returns a `summary.gt_fit` list with estimates, a source-variance table carrying standard errors and a boundary flag where they are available, full covariance components, and diagnostics. Its [print method](#) displays a concise report of numerical acceptance, selected attempts, boundaries, and estimates.

See Also

[gt_design](#), [gt_control](#), [gt_diagnostics](#). The package installs a worked first discrete fit as `discrete-first.R`; the example below shows how to source it.

Examples

```
d <- expand.grid(item = 1:12, rater = 1:3)
d$score <- sin(d$item) + d$rater / 5 + cos(d$item * d$rater) / 4
design <- gt_design("item", "rater", random = ~ item + rater)
fit <- gt_fit(d, "score", design,
  control = gt_control(gaussian = list(check_hessian = FALSE, retry_seed = 42)))
print(fit)
summary(fit)$minus2loglik
# A discrete outcome at one observation per cell: declare the design without
# the object-by-all-facets source, which no discrete fit there can identify.
b <- expand.grid(rater = 1:4, item = 1:10)
b$label <- as.integer((b$item + 2 * b$rater) %% 5 > 1)
binary_fit <- gt_fit(b, "label", gt_design("item", "rater", full_cell = FALSE),
  gt_family("binary"))
gt_diagnostics(binary_fit)$numerically_accepted
# The installed worked example, start to finish:
script <- system.file("examples", "discrete-first.R",
  package = "Gtheory4LLM")
file.exists(script)
```

Description

Log likelihood, observation count, fixed location estimates, and the sampling covariance of the estimated source covariances, following the usual R conventions where they apply and refusing where they do not.

Usage

```
## S3 method for class 'gt_fit'
logLik(object, ...)
## S3 method for class 'gt_fit'
nobs(object, ...)
## S3 method for class 'gt_fit'
coef(object, ...)
## S3 method for class 'gt_fit'
vcov(object, ...)
gt_component_vcov(fit, type = c("components", "parameters"), ...)
```

Arguments

<code>object</code>	A fitted <code>gt_fit</code> object.
<code>fit</code>	A fitted <code>gt_fit</code> object.
<code>type</code>	Which covariance matrix to return. "components" gives the delta-method covariance of the unique source covariance entries. Each is labelled by its source and matrix position. "parameters" gives the free optimizer-parameter covariance instead.
<code>...</code>	Further method arguments; currently unused.

Details

Parameter counts. Gaussian ML profiles one intercept per outcome. Those intercepts are estimated, so `df` counts them alongside the covariance parameters, and `AIC()` and `BIC()` then reproduce the fit's `ml_AIC` and `ml_BIC_response_vectors`. Gaussian REML uses a restricted likelihood whose `df` counts the covariance parameters only; the returned object sets `REML = TRUE` so nothing downstream mistakes it for an ML value, and `AIC()` reproduces `reml_AIC_variance_parameters`. `BIC()` on a REML fit uses `N` response vectors with that same parameter count, which the fit records as `reml_BIC_response_vectors_variance_parameters`; the $N * D$ scalar-score convention remains `reml_BIC_scalar_scores_variance_parameters`. A restricted-likelihood criterion compares models sharing the same fixed-effects structure. Every fit here has exactly one intercept per outcome, so covariance models remain comparable, but a REML value must never be compared with an ML value.

Observations. `nobs()` counts measurement rows, which are the response vectors of the likelihood. A joint fit of `D` outcomes on `N` rows has `N` response vectors and not $N * D$ independent observations,

because the outcomes within a row share the same random sources. BIC() therefore uses N. The alternative $N * D$ scalar-score convention and the archive's variance-only conventions remain in the fit object under explicit names; see [gt_fit](#).

Rejected fits. A fit that failed numerical acceptance is refused by logLik(), and through it by AIC() and BIC(), just as [gt_reliability](#) refuses it. The two apply one rule: a Gaussian fit is refused on an explicit failure, and a discrete fit must carry an affirmative acceptance, so a discrete object whose acceptance flag is missing or NA is refused rather than read as accepted. Its objective is retained in minus2loglik for diagnosis, but a likelihood that did not pass the fit's own numerical checks is not a basis for model selection.

Discrete fits. The log likelihood is a first-order Laplace approximation to the marginal log likelihood, labelled as such in the returned object. Comparing two Laplace-approximated values compares two approximations, not two exact likelihoods.

Why vcov() refuses. In R, vcov(fit) is the covariance matrix of coef(fit), and generic tooling relies on that pairing. This implementation does not estimate it: Gaussian outcome means are profiled out of the likelihood rather than fitted as free parameters, so no curvature for them exists, and the dense discrete engine computes no observed information for anything. Returning a different matrix under that name would be surprising however carefully it were documented, so vcov() raises an error naming the reason and pointing at gt_component_vcov().

gt_component_vcov() is the covariance that does exist: the sampling covariance of the estimated source covariances, which is what [gt_reliability](#) propagates into a coefficient interval. It raises an error carrying the fit's own recorded reason when none is available, rather than returning zeros or an invented matrix: a discrete fit always, a Gaussian fit computed with check_hessian = FALSE, and one whose Hessian is not positive definite. Where standard errors condition on boundary components being held fixed, conditional_on_fixed names them all and conditional_on_zero names the subset whose fitted covariance is entirely zero; a singular but nonzero source is held fixed at its fitted covariance, not at zero.

coef() returns location parameters on their own scale: Gaussian profiled means; binary intercepts; ordinal thresholds, which are that model's location parameters, with the structurally zero dimension mean omitted; and one intercept per non-reference contrast for unordered categorical outcomes. Variance components are not coefficients; use [gt_components](#).

Value

logLik() returns a logLik object carrying df, nobs, a REML flag, and a likelihood label. nobs() returns the number of measurement rows as an integer. coef() returns a named numeric vector of fixed location or contrast estimates. gt_component_vcov() returns a labelled covariance matrix with scope, method, and conditional_on_zero attributes, or raises an error explaining why none exists. vcov() always raises an error; see below.

See Also

[gt_fit](#), [gt_components](#), [gt_diagnostics](#)

Examples

```
d <- expand.grid(item = 1:12, rater = 1:3)
d$score <- sin(d$item) + d$rater / 5 + cos(d$item * d$rater) / 4
design <- gt_design("item", "rater", random = ~ item + rater)
```

```

fit <- gt_fit(d, "score", design,
  control = gt_control(gaussian = list(retry_seed = 42)))
logLik(fit)
nobs(fit)
coef(fit)
AIC(fit) # restricted-likelihood criterion for a REML fit
# The covariance of the estimated source covariances. vcov() deliberately
# refuses: this package does not estimate the covariance of coef().
dimnames(gt_component_vcov(fit))[[1L]]

```

gt_pilot_plan

Evaluate Pilot Precision under a Gaussian Parameter Scenario

Description

Simulates and refits proposed balanced Gaussian pilot panels, then records precision of a coefficient for one fixed target annotation protocol. Increasing pilot item count changes estimation information; changing the target allocation changes the coefficient being estimated. These two choices remain explicit.

Usage

```

gt_pilot_plan(fit, grid, nsim, seed, coefficient = "Erho2", outcome = NULL,
  score = NULL, kind = "outcome", width_target = NULL, level = 0.95,
  target_counts = NULL, components = NULL, means = NULL,
  control = fit$control, max_rows = 100000L)

```

Arguments

fit	An accepted Gaussian fit supported by gt_simulate .
grid	A numeric data frame with one proposed pilot per row. The required <code>n_items</code> column gives the object count. Optional facet columns give the pilot's facet counts. All counts must be integers of at least two. At most 100 rows are accepted. Unspecified facets retain the original counts. A facet called <code>n_items</code> must first be renamed to avoid ambiguity.
nsim	Number of attempted replicates per candidate. The total across all candidates must not exceed 10000. Small values are software illustrations, not precise planning evidence.
seed	Required integer seed. Each attempted replicate receives a recorded deterministic data seed and a separate <code>retry_seed</code> for its refit. An explicitly supplied Gaussian retry seed is preserved. Otherwise a separate seed is assigned and recorded for each refit. The caller's random-number state and kind are restored.
coefficient	"Erho2" or "Phi".
outcome	One reported outcome. Required for multivariate outcome results.
score	Optional explicit composite weights from gt_score .

kind	"outcome" or "composite"; select one coefficient.
width_target	Optional desired total interval width in $(0, 1]$. A precision success requires an available interval no wider than this value.
level	Interval confidence level passed unchanged to gt_reliability .
target_counts	Named positive integer instrumentation counts defining the eventual annotation protocol whose coefficient is estimated in every replicate. Unspecified counts, including all counts when NULL, use the original fit's counts. They do not change with the pilot candidate. Counts of one are allowed here even though fitting a pilot needs at least two levels.
components, means	Optional parameter scenario overrides as documented for gt_simulate .
control	Refitting controls from gt_control ; default is the original fit's controls. No numerical acceptance rule or limit is relaxed.
max_rows	Maximum generated rows per replicate, as for gt_simulate .

Details

The original estimator and resolved source covariance structures are preserved on every refit. New items and all instrumentation levels are sampled independently. The initial scope uses fully random facets, with no batch declaration. No claim is made about incomplete layouts, discrete outcomes, or reuse of specific fixed evaluator realizations.

Every planned replicate has a record, including simulation errors, preparation or fitting errors, numerical refusals, reliability errors, and unavailable intervals. Warnings are retained in that record. Error, refusal, and unavailability are not silently deleted from rate denominators. Width summaries condition on an available interval and show their denominator. Precision success rates use all attempted replicates, so failures and unavailable intervals are non-successes; they do not imply a true coefficient or width of zero.

Reported Monte Carlo standard errors for rates use $\sqrt{p(1-p)/B}$, where B is the number of attempted replicates for unconditional rates. The explicitly conditional precision rate and its Monte Carlo standard error instead use the number of available intervals. The mean-width Monte Carlo standard error uses the sample standard deviation divided by the square root of the number of available intervals. A zero plug-in rate standard error after no failures or all failures is not evidence that the underlying probability is known exactly. Increase replicate counts and examine parameter scenarios before making a collection decision.

Intervals retain the fitting engine's asymptotic delta-method interpretation. Conditional boundary intervals are explicitly flagged. This procedure evaluates model-based plug-in precision, not calibrated coverage, a parametric-bootstrap confidence interval, or guaranteed performance in the actual annotation study. It does not select an optimal pilot automatically.

Value

A list with allocations, one row per proposed pilot; replicates, one row per attempted simulation/refit; summary, rates, denominators, Monte Carlo standard errors, and conditional width summaries; scenario, the parameters, target protocol, estimator, coefficient, seeds, and controls; and interpretation.

accepted counts accepted fits even when coefficient extraction fails. `interval_unavailable` counts successful coefficient extraction without a finite interval. `reliability_errors` counts extraction failures separately. The `interval_available` count is the denominator of conditional width summaries. `conditional_intervals` reports how many available intervals hold boundary components fixed. The complete replicate table is authoritative for further stratified summaries.

Two character columns retain diagnostics from each refit. The `acceptance_failures` column contains failure codes separated by |. The `selected_attempt` column identifies the returned optimizer attempt. These columns survive ordinary CSV export; whole fit objects are not retained. A missing code or identifier is NA, including when no fit was returned. Accepted fits usually have no failure codes, while their selected attempt is retained when recorded. Reliability errors and unavailable intervals retain diagnostics from the completed refit. Missing diagnostics do not imply acceptance; the `accepted` column remains authoritative.

To replay a replicate, generate data with its recorded seed, copy `scenario$control`, replace the copied Gaussian `retry_seed` with the recorded refit seed, and refit with the recorded estimator and covariance structures. Earlier replicate failures or optimizer retries do not alter later replicates' random streams.

See Also

[gt_simulate](#), [gt_reliability](#), [gt_dstudy](#)

Examples

```
## Not run:
# fit is a numerically accepted Gaussian fit with an evaluator facet.
# Pilot allocation varies, but each pilot estimates a three-evaluator protocol.
plan <- gt_pilot_plan(fit,
  grid = expand.grid(n_items = c(40, 80), evaluator = c(3, 5)),
  nsim = 200, seed = 2026, coefficient = "Phi", width_target = 0.15,
  target_counts = c(evaluator = 3))
plan$summary
plan$replicates

## End(Not run)
```

gt_plan

Rank a Bounded Set of D-Study Allocations by User-Supplied Cost

Description

Exhaustively evaluates a declared set of balanced facet counts under one fitted model. It retains every candidate, screens a reliability target, identifies minimum-cost ties and a cost–performance Pareto set, and explains changes in error contributions relative to a declared baseline candidate.

Usage

```
gt_plan(fit, candidates, cost, currency, cost_date, study_items, target,
        coefficient = "Erho2", outcome = NULL, kind = "outcome",
        screening = c("point", "lower"), baseline = 1L,
        constraints = NULL, scale = NULL, score = NULL,
        fixed = character(), level = 0.95, max_candidates = 10000L,
        cost_description = NULL)

## S3 method for class 'gt_plan'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'gt_plan'
print(x, ..., digits = 4L, max_rows = 12L)
```

Arguments

fit	An accepted <code>gt_fit</code> supporting analytic balanced reliability. Fits with a modelled Call component are refused by this planner.
candidates	A nonempty numeric data frame of instrumentation facet counts, or a named list of count vectors expanded to their full Cartesian product. Omitted facets retain their fitted counts. Counts must be positive integers below 2^{53} . Duplicate rows remain distinct candidates.
cost	A numeric data frame of nonnegative, finite monetary cost components, with one row per candidate, or a function called once with a planning data frame. The planning frame contains <code>design_id</code>, <code>allocation_</code> prefixed facet counts, <code>annotations_per_item</code>, <code>study_items</code>, and <code>total_annotations</code>. It has an <code>allocation_columns</code> attribute mapping column names to the original facets. The function returns a cost-component data frame or a list containing costs. Such a list can also be supplied directly. Its optional entries are: • <code>resources</code>: a nonnegative numeric data frame. • <code>assumptions</code>: character text. Components are summed: do not include an already summed total as an extra component. Prices, usage assumptions, overhead, retry costs, evaluator setup and human-review costs are entirely user supplied.
currency	One three-letter uppercase currency code; all cost components must already use that same currency. No exchange conversion is performed.
cost_date	The date of the supplied cost assumptions, as Date or "YYYY-MM-DD" text.
study_items	The number of distinct items to be costed. This is a positive integer held constant across candidates. It multiplies annotations and costs; it does not change reliability coefficients or estimate pilot-study precision.
target	A number from zero to one for the selected coefficient.
coefficient	"Erho2" for relative comparisons or "Phi" for absolute decisions.
outcome	The outcome to screen. Required when multiple outcomes are reported.
kind	"outcome" or "composite"; a composite requires score.

screening	"point" screens the projected point estimate. "lower" separately screens its available lower confidence bound at level. Missing bounds remain unavailable and do not pass the screen.
baseline	The one-based candidate row identifying the comparison protocol; default is row 1. Its complete allocation and cost are retained. The baseline can be infeasible: it remains a comparison, not a recommendation.
constraints	Optional function applied to the joined candidate, coefficient, cost and resource table, or a row-aligned logical vector/data frame supplied directly. A function returns a logical vector or a data frame of named logical checks. FALSE excludes a candidate; NA records unknown feasibility. The checks and their reasons are retained. Cost components use the prefix <code>cost_component_</code> ; resource columns use <code>resource_</code> .
scale, score, fixed, level	As in gt_reliability . The score and fixed-facet universe are declared once for all candidates. A fixed facet must retain its observed count; it is never selected or changed by the optimizer.
max_candidates	Positive integer upper limit checked before Cartesian expansion; default 10,000. The caller defines the search bounds.
cost_description	Optional explanatory text recording the source and units of the cost model. Supply additional assumptions through the cost record.
x	A <code>gt_plan</code> object.
row.names, optional	Standard data-frame conversion arguments.
...	Reserved for S3 compatibility.
digits, max_rows	Printing precision and maximum displayed rows.

Details

Three distinct sizes. `study_items` counts distinct study items. `annotations_per_item` is the product of facet counts and the fitted equal within-cell replication. `total_annotations` is their product. Items per request, request counts and retry counts may be supplied as cost resources; they are not statistically interchangeable with repetitions per item. Total annotations must remain exactly representable integers below 2^{53} .

What is optimized. All allocations use the same accepted fit, source covariances, score, outcome scale and declared facet universe. Counts represent exchangeable levels under that model. The function does not replace an expensive evaluator with a cheaper identity, refit an evaluator subset, change a random facet into a fixed facet, or alter a batching layout to improve a coefficient. A heterogeneous cost model must still describe costs under that same statistical sampling universe. Evaluator-identity selection needs separate model evidence. Adding a human-review cost does not by itself model a reliability benefit.

Search and missing values. A candidate is eligible when all constraints are satisfied and the selected screening value reaches the target. Minimum-cost flags retain every exact cost tie among eligible candidates. Pareto status compares cost with the selected performance measure among feasible candidates having an available screening value, independently of the target. A design is

dominated when another is no more expensive and performs at least as well, with at least one strict improvement. Exact duplicate cost–performance ties remain non-dominated. Unknown feasibility or performance has NA Pareto status; known infeasible rows have FALSE. An unavailable lower bound is not replaced by its point estimate or a zero standard error. A missing lower bound cannot qualify even if the point estimate exceeds the target.

Interpretation of intervals. The available Gaussian intervals retain the parent reliability calculation’s model and any conditional boundary qualification. They are pointwise estimates of uncertainty, not simultaneous coverage guarantees after searching many designs and not future-panel prediction bounds. Discrete Laplace fits currently provide latent point coefficients only, so lower-bound screening retains their explicit unavailable-uncertainty reason.

Batching boundary. Fits with a modelled Call effect are explicitly refused until this planner supports layout-specific reliability. Request batching can enter a cost callback for a fit without a call effect, but doing so does not account for shared-call dependence. No source is dropped or silently reinterpreted.

Marginal explanations. Changes are candidate minus baseline for cost and coefficients, and baseline minus candidate for relative/absolute error variance. Negative reductions therefore identify increased error. Per-source reductions add to their corresponding total error reductions. These are conditional projections, not causal effects, empirical accuracy, or a promise that future estimates will attain the target.

Value

A `gt_plan` list. `results` (also as `data.frame(x)`) retains all candidate allocations, coefficients and intervals, cost components/resources, constraint status/reasons, target status, minimum-cost and Pareto indicators, cost rank among feasible rows, and baseline cost/error/coefficient changes.

`source_contributions` reports universe, relative-error and absolute-error contributions and the error reductions per source. The input allocations and full coefficient tables remain in `allocations` and `all_coefficients`. Cost and resource records remain in `cost_breakdown` and `resources`. The `constraint_checks` element preserves every check. The `baseline` element records the chosen comparison.

`provenance` records the currency, date, cost assumptions and callback text. It also records the search scope, target, score, facet universe, interval qualifications, original fitted counts and selected baseline. `fit_diagnostics` retains the fitted numerical diagnostics. No request is executed and no provider price is queried.

See Also

[gt_reliability](#), [gt_dstudy](#), [gt_dstudy_target](#)

Examples

```
## Not run:
# fit is an accepted Gaussian item x evaluator x prompt model.
plan <- gt_plan(fit,
  candidates = list(evaluator = 2:4, prompt = 1:3),
  cost = function(d) list(
    costs = data.frame(annotation = 0.01 * d$total_annotiations,
      setup = 2 * d$allocation_evaluator),
```

```

resources = data.frame(request_count = d$total_annotatons),
assumptions = "Illustrative monetary units; one item per request."),
currency = "USD", cost_date = "2026-10-09", study_items = 1000,
target = 0.80, coefficient = "Phi", baseline = 1,
constraints = function(d) data.frame(
  budget = d$total_cost <= 150,
  requests = d$resource_request_count <= 10000),
cost_description = "User-supplied illustrative costs, not provider prices.")
plan
subset(as.data.frame(plan), minimum_cost
plan$source_contributions

## End(Not run)

```

gt_preflight

Inspect Data, Source Dimensions, and Backend Limits Before Fitting

Description

Reports observed design and outcome information, random-source dimensions, parameter counts, and the current backend's structural and resource restrictions without optimizing a model. Declared absent categories are reported as a failed check; fitting continues to reject them. Other category and source-resolution rules match fitting.

Usage

```

gt_preflight(data, outcomes, design, family = gt_family("gaussian"),
  covariance = "unstructured", residual = NULL,
  control = gt_control(), max_examples = 10L)

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

## S3 method for class 'gt_preflight'
plot(x, type = c("cells", "sources", "outcomes"),
  max_sources = 20L, outcome = NULL, max_categories = 12L,
  max_variables = 8L, max_label_chars = 18L, ...)

```

Arguments

data	A nonempty data frame with unique column names.
outcomes	One or more outcome column names.
design	A design from gt_design .
family	A family from gt_family , or a named family list matching the outcomes.
covariance	The source covariance specification accepted by gt_fit .
residual	Gaussian residual covariance specification, or NULL for the default. Do not specify this argument for discrete outcomes.

control	Engine controls from gt_control . Resource limits are read from the selected engine's controls.
max_examples	A nonnegative finite integer, no greater than <code>.Machine\$integer.max</code> . Maximum rows in each panel-audit table and in each outcome/variable no-variation example table, and maximum invalid design-row indices in an error. Zero suppresses examples while retaining counts and truncation metadata. Defaults to 10.
x	An object returned by <code>gt_preflight</code> .
type	Plot mutually exclusive full-cell counts ("cells"), random-source dimensions ("sources"), or descriptive category coverage across design-variable groups ("outcomes").
max_sources	A positive finite integer limiting the source bars. Sources are sorted by decreasing random dimension, with ties in source-table order. The plot explicitly reports truncation. Ignored for cell bars, but still validated.
outcome	For <code>type = "outcomes"</code> , one outcome name. Required when preflight contains multiple outcomes. Gaussian outcomes have numeric profile tables, not a category heatmap.
max_categories, max_variables	Positive finite integers limiting heatmap rows and columns. Categories follow declared order; variables follow design order. Displayed and total counts, and truncation, are printed on the plot.
max_label_chars	A finite integer of at least four, limiting heatmap label text before a position number is added. Abbreviated labels map to the complete profile tables by that number.
...	For cell/source plotting, named arguments passed to barplot ; for category coverage, named arguments passed to image . Supplied arguments override plotting defaults. Reserved for print-method compatibility.

Details

The panel audit uses the Cartesian product of the levels actually observed in each design column. Unused factor levels are excluded. It does not infer unobserved levels, physical crossing, or permitted combinations from sampling intent. No data, source specification, or model is automatically corrected. `panel_audit` contains:

- `summary`: one row each for `complete`, `under_replicated`, `over_replicated`, and `missing`. These categories are mutually exclusive. Complete means an observed full cell has exactly the declared replication; under-replicated cells have at least one row but fewer than declared; missing cells have no rows. `cell_count_exact` stores exact decimal counts as character strings. `cell_count` is a numeric convenience column: counts above 2^{53} can be rounded, and counts beyond the finite numeric range are `Inf`. `expected_cells_exact` gives the exact full-product count.
- `missing_cells`: representative absent full cells, retaining the original design-column names, sampled values, and classes. Factor levels not present in an example table are dropped, including all levels in empty tables, so factor attributes do not expose omitted identifiers. Examples use first-observed level order, with the object column varying fastest.

- `replication_issues`: cells with incorrect replication, in input order. Design columns are followed by observed replication, declared replication, and status metadata.
- `row_examples`: input-row examples from cells with incorrect replication, in input order. Only design columns and audit metadata are copied; outcomes and unrelated data columns are excluded. The row metadata is the one-based input-row position, independent of data-frame row names.
- `metadata_columns`: maps metadata roles to table column names. The roles identify the input row, observed and declared replication, and status. The corresponding keys are:
 - `row`
 - `observed_replicates`
 - `declared_replicates`
 - `status`

Metadata normally uses the prefix `.gt_`; leading dots are added until none of the metadata names collide with input column names. Original column names are not repaired or renamed.

- `examples`: for each example table, the requested `limit`, number shown, total available (`total` and authoritative `total_exact`), and whether examples were truncated.

Examples are deterministic, not random samples. Their counts use all input rows. Missing-cell examples are generated incrementally, visiting at most the observed-cell count plus the requested example count; the full Cartesian grid is never materialized. All three example tables are capped separately.

Each `outcome_profile[[outcome]]` contains:

- `family`, `link`, and `summary`: the number of rows and distinct observed values, and declared/observed category counts for discrete outcomes. Gaussian outcomes instead have minimum, maximum, mean, and sample standard deviation; derived summaries outside the finite numeric range are NA. Gaussian values are not converted to categories. Globally constant, missing, or nonfinite Gaussian inputs still raise the fitting validation error.
- `categories`: declared category labels, counts, row proportions, and an observed flag. Safely resolved declared categories, including unused declared factor levels, appear with zero counts when absent. The binary numeric/logical category labels follow fitting's 0/1 convention. This table is NULL for Gaussian outcomes.
- `by_variable`: the number of observed groups, groups with exactly one distinct response (`no_variation_groups`), groups with one row (`single_row_groups`), and the no-variation fraction for each design variable. Single-row groups are included in no-variation counts, and counted separately to distinguish missing replication from repeated constant responses.
- `category_coverage`: for each variable/category pair, the number and proportion of observed groups containing at least one row in that category. The denominator is the observed group count, not the measurement-row count. A group may contain several categories, so these proportions need not sum to one. This table is NULL for Gaussian outcomes.
- `group_members`: the columns defining each variable's groups. Declared nested variables include all declared ancestors. Other variables use their marginal coded levels. Reused nested IDs are distinguished by their parents. No additional physical nesting is inferred. The `grouping_scope` column of `by_variable` records this choice:
 - `declared_parent_scoped`: groups include declared ancestors.

- `marginal_coded_levels`: groups use one variable's coded levels.
- `no_variation_examples`: one table per design variable containing at most `max_examples` grouping identifiers, row counts and distinct-value counts, in first-observed order. Sampled values, column classes, and order are preserved; unused factor levels are dropped. Outcome values and unrelated columns are excluded. `metadata_columns` maps `row_count` and `distinct_values` to collision-safe metadata names. `examples` reports total, shown, limit, and truncation for each variable.
- `scope and privacy`: interpretation and sharing notes. These tables are descriptive, with no minimum category count or variation threshold for adequacy. They do not certify identification, fit acceptance, or likelihood-approximation accuracy, and do not collapse categories. Category labels, small counts, and sampled grouping identifiers may be sensitive. `max_examples = 0` suppresses group identifiers in example tables, including unused factor levels, but leaves aggregate counts and category/outcome labels. The preflight object is not anonymized.

The recorded call keeps its data argument only when that argument is a plain object name. A data frame supplied through `do.call()`, or an inline expression, is recorded as a marker rather than stored in the call.

For a design made with a `gt_batch` declaration, `batch_audit` contains the declared size, order source, facet and reaches, and membership: "inferred" when the batches are cut from the item order, "recorded" when the calls are read from the column named in the declaration's `id`. It counts the items, the distinct batches, the conditions and the calls. Recorded calls are counted. Inferred calls are implied: one for each batch, condition and declared replicate. `equal_sized`, `calls_below_size` and `smallest_observed_call` describe calls with fewer items present than declared. That is a property of the rows supplied, not a problem, and not evidence of how many items a call was sent: a call sent short and a call that lost rows after collection look the same, and a call with no row left is not counted. `position_scope` records that positions are ranks among the items present. `fixed_composition` and `fixed_order` say whether every condition holds the same batches, and each batch in one order. `consistent` is TRUE when the declaration can be laid over the data, with the reasons in problems when it cannot. At most `max_examples` rows list items with their batch, or their recorded call, and position. The object column retains its original name. `batch_audit$metadata_columns` maps the roles `batch` (inferred membership) or `call` (recorded membership), and `position`, to their actual columns in `batch_audit$examples`. These normally use the role names. If an object name collides, metadata names receive a `.gt_` prefix, extended with leading dots if needed; the object name is unchanged. Use this mapping when selecting example metadata by name. `model` says whether a fit of these data would estimate a shared call effect, with the reason when it would not, the name of the source it adds and that source's covariance parameters, which the total parameter count includes. No coefficient scale is offered for a design whose fit models a call effect; see `gt_batch`.

Inferred membership is a reconstruction from the rows supplied, and `consistent` is then no evidence about how the data were collected: items removed after collection, calls of other sizes and regrouping between conditions leave no trace. `membership_scope` says so in the object and the printed report repeats it; for recorded calls it states the limits just described. For inferred batches `stored_rows` reports, when every condition holds each item once, in how many conditions consecutive blocks of rows hold one declared batch, and in how many they also follow the declared order. That agreement means something only when rows are stored call by call. The audit estimates no dependence and does not enter checks.

For absent declared categories, the profile and structural/resource estimates remain available; free-covariance and source-kernel checks are skipped after the failed category check, and no reliability

scale is offered.

The default plot shows horizontal bars for the four exclusive cell categories. Direct count labels make zero and small counts visible beside large bars; labels above $1e12$ use abbreviated scientific notation marked with a tilde. Exact counts remain available in the audit table. The source plot shows the largest random dimensions, up to `max_sources`. Plots describe observed coded levels, not runtime, statistical identification, or fit acceptance. Numeric cell bars may be rounded above 2^{53} ; a cell plot raises an error when counts exceed the finite plotting range. Consult the exact count strings in that case.

The outcome heatmap shows group coverage for one discrete outcome, with a fixed zero-to-one color scale and bounded category/variable labels. An asterisk marks a variable whose groups include declared nesting ancestors. Small displays (at most eight variables and twelve categories) include percentage labels; near-zero and near-one values are abbreviated without labeling them exactly zero or one. The plot does not show per-group response rows or treat sparse coverage as an automatic model rejection. Larger displays use colors only.

Each random source has one predictor dimension per Gaussian, binary, or ordinal outcome and one dimension per nonreference category for a categorical outcome. Its random dimension is the product of that dimension count and its observed grouping count. These are conceptual random-effect dimensions for Gaussian fitting; the exact Gaussian backend instead uses factorial contrasts.

The source table counts free variance/covariance parameters under the requested covariance structure. Fixed discrete source covariance matrices contribute zero free covariance parameters. Gaussian observation parameters are the profiled outcome means; discrete observation parameters are intercepts or ordinal thresholds. The total includes these observation parameters and the Gaussian residual covariance parameters. It is not a residual degrees-of-freedom count or an information-criterion definition.

Gaussian checks require a complete coded Cartesian panel with one measurement per cell and resolve a full-cell source into the residual when appropriate. Discrete fitting can admit missing whole cells, but analytic reliability and D studies still require a complete balanced coded panel. Within configured size limits, preflight also applies the current discrete source-kernel checks. When an earlier structural or resource check fails, `kernel_check` explicitly reports that the kernel checks were skipped. Source counts then refer to the requested model if family-specific resolution failed.

The Gaussian allocation estimate covers preparation arrays only. The discrete byte count covers one dense random-design matrix and one random-effect Hessian only. Neither predicts peak memory or runtime. Fewer than five observed groups triggers a descriptive note, not a rejection rule. Starting values and optimizer-specific controls are validated by fitting.

Passing preflight does not establish statistical identification, reliable estimation, numerical acceptance, or Laplace-approximation adequacy. Supported scales remain conditional on a numerically accepted fit: observed Gaussian scores, or explicitly requested latent binary/ordinal responses. Latent coefficients describe averages of latent responses, not majority-vote decisions or observed label proportions. No scalar nominal coefficient is implemented.

Value

A `gt_preflight` list with observed counts and replication, source and parameter totals, resource estimates, and interpretation notes. It also contains:

- `sources`: a data frame describing the random sources a fit would estimate: the retained declared sources and, for a batch the fit would model, the call source. The parameter and dimen-

sion totals describe these.

- `panel_audit`: bounded panel diagnostics, described below.
- `outcome_profile`: a named list of descriptive outcome summaries, category counts, grouping coverage, and bounded no-variation examples.
- `batch_audit`: for a design that declares batches, the calls that declaration implies or records; absent otherwise. Described below.
- `checks`: a data frame of the structural and resource checks.
- `fitting_feasible`: whether those checks passed.
- `supported_reliability_scales`: conditional coefficient scales.

Feasibility is not a fitting or identification result. Invalid input types or undeclared observation categories raise an error. Declared but absent categories produce a failed `outcome_category_coverage` check and `fitting_feasible = FALSE`; no category is removed and `gt_fit()` still raises an error. Unsupported observed designs and exceeded resource limits are reported as failed checks. Both print and plot methods return `x` invisibly.

See Also

[gt_fit](#), [gt_diagnostics](#), [gt_reliability](#), [gt_dstudy](#)

Examples

```
# Entirely synthetic evaluator/prompt data; no API or study data are used.
d <- expand.grid(item = 1:8, evaluator = 1:3, prompt = 1:2)
d$score <- sin(d$item) + d$evaluator / 5 + cos(d$item * d$prompt) / 4
design <- gt_design("item", c("evaluator", "prompt"),
  random = ~ item + evaluator + prompt + item:evaluator)
report <- gt_preflight(d, "score", design)
print(report)
report$sources

# Missing whole cells block Gaussian fitting and analytic reliability.
incomplete <- gt_preflight(d[-1, ], "score", design)
incomplete$checks[!incomplete$checks$passed, ]
incomplete$panel_audit$missing_cells
plot(incomplete)
plot(report, type = "sources", max_sources = 5L)
```

gt_reliability

Compute Balanced G and Phi Coefficients

Description

Forms universe-score, relative-error, and absolute-error covariance matrices under a declared balanced averaging design.

Usage

```
gt_reliability(fit, scale = NULL, score = NULL, design = NULL, counts = NULL,
  fixed = character(), level = 0.95)
## S3 method for class 'gt_reliability'
print(x, ..., digits = 4L, max_rows = 12L)
## S3 method for class 'gt_reliability'
as.data.frame(x, row.names = NULL,
  optional = FALSE, ...)
```

Arguments

fit	A numerically accepted gt_fit with a complete balanced coded panel.
scale	"observed" for Gaussian outcomes (default); explicitly request "latent" for binary/ordinal outcomes.
score	Optional gt_score weights. Per-outcome results are always returned.
counts	Optional named positive integer instrumentation counts overriding the fitted allocation. Unspecified counts retain their fitted values.
design	Compatibility alias for counts. Supply only one of these arguments.
fixed	Instrumentation facets whose universe of generalization is exactly their observed levels. Defaults to none, which is the fully random model. A fixed facet's count cannot be changed, and at least one facet must remain random.
level	Confidence level for the reported coefficient intervals.
x	A gt_reliability object.
digits	Significant digits used when printing.
max_rows	Maximum number of outcome rows to print.
row.names	Optional row names for the exported data frame.
optional	Accepted for generic compatibility. Exported column names are always preserved without syntactic name repair.
...	Reserved for additional print or data-frame arguments.

Details

Tabular export. `as.data.frame()` returns an ordinary data frame with one row per outcome and an additional row for a requested weighted composite. `design_id` is 1 for this single allocation. `kind` is "outcome" or "composite"; use it together with `outcome`, because a measured outcome may itself be named "composite". Coefficients retain the names `Erho2` and `Phi`, with their `_se`, `_lower`, and `_upper` columns, including missing values when uncertainty is unavailable.

Allocation columns prepend `allocation_` to each exact facet name. The mapping attribute is named `allocation_columns`. It maps exported names back to the original facets, protecting coefficient fields from similarly named facets. The table also includes:

- `scale` and `interval_level`: coefficient interpretation.
- `uncertainty_available`: covariance availability.
- `uncertainty_reason`: the recorded reason.

- `uncertainty_conditional`: the conditional-inference flag.
- `uncertainty_conditioning`: what was held fixed.
- `Erho2_interval_available`: relative interval availability.
- `Phi_interval_available`: absolute interval availability.
- `fixed_facets`: the declared fixed universe.
- `composite_weights`: weights for composite rows.
- `measurements_per_object`: the allocation total.
- `measurement_count_exact`: integer precision status.
- `extrapolated`: extension beyond fitted facet counts.
- `batch_status`: whether the design declared batches.

`uncertainty_available` describes availability of the fitted covariance used for propagation; individual intervals can still be missing, for example at a coefficient boundary. The conditional flag records inference restricted to the interior parameter block, and the conditioning text identifies what was held fixed, distinguishing zero from singular nonzero covariance. The per-coefficient availability flags indicate that the estimate and both interval bounds are finite in that particular row. Fixed facets are listed with quoted, escaped labels, separated by semicolons; an empty string means none. Composite rows carry similarly quoted outcome labels and numerical weights; `composite_weights` is NA for measured-outcome rows. `measurement_count_exact` is TRUE for finite allocation totals below 2^{53} , the range in which consecutive integers have exact double representations. Larger totals are retained for reporting but not treated as exact counts when comparing candidate minima. `batch_status` is "not_declared", or "declared_not_modelled" for a design made with `gt_batch` whose batches the fit did not model: the coefficients in that row treat items annotated in one call as independent. Printing and plotting say the same. A fit that does model a call effect is refused by these ordinary methods. Use `gt_batch_reliability` and `gt_batch_dstudy` for explicit targets conditional on the fitted fixed Gaussian batching layout; no universal batch-adjusted G is assumed.

Columns are atomic vectors suitable for CSV output. Additional attributes retain the fixed-facet declaration, a compact uncertainty record, the structured score, the batch status and the interpretation. Their names are `fixed_facets`, `uncertainty`, `score`, `batch` and `interpretation`. CSV does not preserve R attributes; keep these alongside a CSV when reporting the full analysis context. Conversion does not refit the model or create new uncertainty estimates.

Balanced coded panel. This is the complete Cartesian product of the observed coded levels of the object and of every instrumentation facet, carrying exactly the declared number of rows in each of those cells, with no missing outcome. For a nested facet, *coded* means the child's within-parent code rather than a globally unique label, so the child's identity is the child-by-parent source; `gt_design` works both codings through. A panel that is not balanced in this sense has no analytic coefficient here, and none is approximated.

In the fully random model, the object main-effect covariance defines universe-score variation. Every object-containing error component contributes to relative error, and every non-object-main component contributes to absolute error. Each component is divided by the product of the instrumentation counts in its grouping term; the residual is also divided by declared cell replication.

Declaring a facet fixed selects the mixed model of Brennan (2001). The universe of generalization then contains exactly that facet's observed levels: an object interaction containing only fixed instrumentation facets is averaged over those levels and added to universe-score variance; a source built

only from fixed instrumentation facets shifts every object equally and leaves the comparison; every remaining source keeps its usual divisor. With no fixed facet this reduces exactly to the fully random decomposition. `$source_roles` records which role each source took. Fixed versus random treatment follows the intended universe of generalization, not a facet name or whether its levels were researcher-selected. The option changes coefficient aggregation after fitting; it does not repair a misspecified G-study covariance model.

Intervals use the delta method on the logit scale, propagating the fitted parameter covariance matrix through the source covariance entries. They are asymptotic Wald intervals for estimation uncertainty under the declared random-effects model and allocation. Finite sampling of facet levels is part of that model; the intervals are not prediction intervals for the realized scores of a new panel and do not quantify misspecification of the facet populations. Wald theory does not generally apply to a source resting on a variance boundary. When inference uses an interior parameter block, the remaining intervals condition on the listed fixed components, and the fit says which way: a source whose fitted covariance is entirely zero is held at zero, and a singular but nonzero source is held fixed at its fitted covariance, which is a different statement. Unconditional coverage after that selection has not been established. The discrete engine supplies no parameter covariance matrix, so its coefficients remain point estimates.

Coefficients for discrete outcomes are identified latent-response coefficients. They describe the average latent response under the declared measurement design, not binary proportions, majority-vote labels, or observed ordinal scores. Scalar reliability for unordered categories, observed discrete reliability, and unbalanced decision-study integration are not implemented. Instrumentation facets are averaged as random sources unless fixed names them; the function never infers a fixed-facet score universe on its own.

Value

A `gt_reliability` list containing per-outcome Erho_2 and Φ with their standard errors and interval bounds, optional composite coefficients, universe/relative-error/absolute-error covariance matrices, the role each source played in the decomposition, coefficient scale, allocation, fixed facets, extrapolation flag, the uncertainty record, and fit diagnostics. Standard errors and bounds are NA whenever the fit carries no usable parameter covariance matrix. Printing shows a concise coefficient table, omits all-missing interval columns, and returns the complete object invisibly.

References

Brennan, R. L. (2001). *Generalizability Theory*. Springer. doi:10.1007/9781475734560

See Also

[gt_score](#), [gt_dstudy](#)

Examples

```
d <- expand.grid(item = 1:12, rater = 1:3)
d$score <- sin(d$item) + d$rater / 5 + cos(d$item * d$rater) / 4
design <- gt_design("item", "rater", random = ~ item + rater)
fit <- gt_fit(d, "score", design,
  control = gt_control(gaussian = list(check_hessian = FALSE, retry_seed = 42)))
gt_reliability(fit)$per_trait
```

```
gt_reliability(fit, counts = c(rater = 6))
as.data.frame(gt_reliability(fit))
```

gt_report

Create and Export a Portable Aggregate Analysis Report

Description

Collects retained analysis metadata and verified coefficient projections in a versioned snapshot, then exports a self-contained HTML report with aggregate tables, inline figures, limitations, and provenance. No model fitting, network access, external assets, or additional packages are needed.

Usage

```
gt_report(fit, preflight = NULL, reliability = NULL, dstudy = NULL)
gt_export_report(report, path, overwrite = FALSE)
## S3 method for class 'gt_report'
print(x, ...)
```

Arguments

fit	A <code>gt_fit</code> . Rejected fits can be reported as diagnostic objects, but their reliability or decision-study coefficients are refused.
preflight	Optional preflight report with matching family, source, grouping and counts. Outcome and category coding must also match.
reliability	Optional <code>gt_reliability</code> whose projections agree with the supplied fit under its declared allocation, scale, fixed facets, score, and uncertainty settings.
dstudy	Optional <code>gt_dstudy</code> checked against the supplied fit.
report, x	A <code>gt_report</code> snapshot.
path	Destination <code>.html</code> or <code>.htm</code> file. The parent directory must already exist. The report embeds its styles, tables and SVG figures.
overwrite	Replace an existing destination only when explicitly <code>TRUE</code> . Defaults to <code>FALSE</code> .
...	Reserved for print-method compatibility.

Details

Association checks. Supplied coefficient objects are checked by recalculating their declared projections from the supplied fit. This reuses stored covariance estimates and uncertainty; it does not rerun an optimizer. Coefficient values, intervals, scales, allocations, fixed facets, composite weights and uncertainty context must agree. Legitimately reordered result rows are accepted. This verifies numerical compatibility, not historical object identity or a shared original data file.

Preflight correspondence checks the available model specification, category coding, sources, free covariance-parameter counts when recorded, observed counts, replication and grouping scopes. Requested controls, fixed covariance values and resource-budget correspondence are not verified. Legacy preflight objects lacking grouping metadata are labelled as not verified for nesting. Even

matching aggregate specifications cannot prove that preflight and fitting used the same observations; the report states that limitation prominently. No preflight is inferred from a fit and no coefficient section is calculated when its input is omitted.

Privacy scope. Only explicit aggregate fields are copied. Observations, facet-level identifiers, raw category labels, calls, models, grouping examples, arbitrary controls, diagnostic messages and filesystem paths are excluded. In particular, calls containing literal data frames and unused factor levels in empty example tables are not copied. Variable, outcome and source names, aggregate counts, and numerical estimates remain. These names and small aggregate counts can themselves be sensitive, so this is not anonymization; review a report before sharing it.

Categorical profiles use `category_1`, `category_2`, and so on in declared category order. The private label correspondence is not exported. Category counts and proportions of groups with each category are descriptive, not adequacy or identification thresholds. Gaussian minimum/maximum values, which are individual observed responses, are not copied.

Provenance. The fitting-session record contains only its retained R, platform and selected package versions. The report-generation environment is a separate record and never fills a missing fitting-session version. Session versions describe loaded packages, not necessarily the executing sources when code was sourced separately. Neither record is an archive hash, source commit, or release qualification. Missing session or retention records stay unknown. Model and data retention are not required when the fit retains the compact panel summary and source covariances needed for coefficient checks.

Figures and export. HTML text and SVG labels are escaped; figures use numeric coordinates and contain no scripts or remote references. Reliability figures show at most 30 coefficient rows; D-study figures show at most 600 rows. Coverage figures show at most 20 categories and 12 variables. Figure captions disclose these limits; complete aggregate tables remain available. Long graphical labels may be shortened, with complete labels in the tables. Source variance rows flag user-fixed covariance components so they cannot be mistaken for estimated zeros or estimated variance boundaries. Allocations are unconnected points; intervals appear only where recorded and available. Discrete latent coefficients have no intervals. The report cannot establish observed-label accuracy, a globally optimal allocation, statistical identification, or qualification of a private backend.

The exporter validates the supported schema version, prepares the complete HTML in a temporary file, and protects existing destinations unless `overwrite = TRUE`. No R objects or executable serialization are embedded in the HTML. Saving an RDS snapshot is a separate explicit operation.

Value

`gt_report()` returns a plain-list snapshot of class `gt_report`, with `schema_version = "1.0"`. It supports `saveRDS()` and `readRDS()` without retaining the fitted object. The main fields are:

- `generated_at`: report creation time.
- `analysis`: estimator, backend, numerical status and batch status.
- `design` and `families`: model declarations.
- `random_sources` and `nesting`: grouping declarations.
- `settings`: selected scalar controls.
- `covariance_settings`: recorded covariance structures.
- `diagnostics`: staged numerical status.

- source_variances: component values and fixed flags.
- uncertainty: interval scope and availability.
- retention and provenance: retained context.
- Optional preflight, reliability and dstudy tables.
- association, privacy and limitations: scope text.

Tables have atomic columns; absent optional sections are NULL. `gt_export_report()` returns the normalized output path invisibly. The print method returns the snapshot invisibly.

See Also

[gt_preflight](#), [gt_reliability](#), [gt_dstudy](#), [gt_diagnostics](#)

Examples

```
d <- expand.grid(item = 1:12, rater = 1:3)
d$score <- sin(d$item) + d$rater / 5 + cos(d$item * d$rater) / 4
design <- gt_design("item", "rater", random = ~ item + rater)
fit <- gt_fit(d, "score", design,
  control = gt_control(gaussian = list(check_hessian = FALSE)))
report <- gt_report(fit, gt_preflight(d, "score", design),
  gt_reliability(fit), gt_dstudy(fit, data.frame(rater = c(2, 3, 6))))
report
path <- tempfile(fileext = ".html")
gt_export_report(report, path)
unlink(path)
```

gt_score

Declare Composite-Score Weights

Description

Stores explicit weights without normalizing or choosing a score scale.

Usage

```
gt_score(weights)
```

Arguments

weights	Finite numeric vector with unique outcome names and at least one nonzero entry. Every modeled outcome must be weighted exactly once when used in a reliability calculation.
---------	---

Details

Weights must be substantively meaningful on the chosen outcome scale. A latent binary/ordinal composite is not an observed-score composite. Negative and non-unit-sum weights are allowed.

Value

A gt_score object containing weights and the "as_supplied" normalization policy.

See Also

[gt_reliability](#), [gt_dstudy](#)

Examples

```
gt_score(c(efficacy = 0.5, safety = 0.5))
```

gt_simulate	<i>Simulate a New Balanced Gaussian Annotation Panel</i>
-------------	--

Description

Generates fresh independent Gaussian random effects for objects, instrumentation levels, source interactions, and residuals under a fitted or specified parameter scenario. Distinct item count and instrumentation allocation are separate inputs.

Usage

```
gt_simulate(fit, n_items, counts = NULL, seed, components = NULL,
  means = NULL, max_rows = 1000000L)
```

Arguments

fit	An explicitly numerically accepted Gaussian gt_fit object, retaining its covariance structures and either modelled data or panel summary.
n_items	Number of new distinct objects, an integer of at least two.
counts	Optional named integer counts of at least two for instrumentation facets. Unspecified counts retain the fitted panel counts. Counts for a nested child describe its local coded levels within each parent combination.
seed	Required integer seed between zero and .Machine\$integer.max. The caller's random-number generator kind and state are restored, including when no random seed existed before the call.
components	Optional complete named list of covariance matrices, one for every retained source including Residual. Each matrix must be symmetric, positive semidefinite, and preserve its fitted diagonal, unstructured, or pooled structure. Fully unnamed axes use outcome order; labelled axes must name every outcome exactly once. The fitted components are used when NULL.
means	Optional finite numeric vector naming every outcome exactly once. Defaults to the fitted outcome means.
max_rows	Maximum generated rows, between one and one million. A separate estimated 512 MiB workspace limit also applies. Neither limit changes fitting controls or permits an unsupported model.

Details

The initial scope is a complete balanced coded panel with one outcome vector per cell and no batch declaration. Univariate and multivariate Gaussian models, explicit random sources, and declared parent-scoped grouping terms are supported. Discrete outcomes, incomplete panels, within-cell replication, and batch layouts are refused. These restrictions match the supported Gaussian preparation model; globally unique physical child labels are not generated.

Zero and singular positive semidefinite components are allowed. Only negative eigenvalues within a relative numerical tolerance of 10^{-10} (using a unit minimum scale) are set to zero. No positive interior jitter is added. A degenerate scenario can generate data that fitting subsequently refuses.

Every facet is sampled afresh; the function does not reuse estimated random effects or condition on observed evaluator or item realizations. This is design simulation. A fitted-model parameter scenario can also provide data for an explicitly designed parametric-bootstrap analysis, but this function performs no refitting, interval construction, boundary calibration, or coverage assessment.

Value

A data frame with integer coded object and facet columns and numeric outcome columns. Its simulation attribute records the seed, counts, means, covariance matrices, covariance structures, parameter source, and interpretation. The generated covariance matrices, including numerical tolerance adjustments, are recorded explicitly.

See Also

[gt_pilot_plan](#), [gt_fit](#), [gt_dstudy](#)

Examples

```
set.seed(91)
panel <- expand.grid(item = 1:20, evaluator = 1:3)
panel$score <- rnorm(20)[panel$item] + rnorm(3)[panel$evaluator] +
  rnorm(nrow(panel))
fit <- gt_fit(panel, "score", gt_design("item", "evaluator"))
if (isTRUE(fit$numerically_accepted)) {
  simulated <- gt_simulate(fit, n_items = 30,
    counts = c(evaluator = 4), seed = 2026)
  head(simulated)
}
```

plot.gt_reliability *Plot Reliability Coefficients and Available Intervals*

Description

Draws a forest plot of the already calculated relative and absolute coefficients, with available estimation intervals and explicit outcome labels.

Usage

```
## S3 method for class 'gt_reliability'
plot(x, coefficient = c("Erho2", "Phi"),
     interval = TRUE, target = NULL, ...)
```

Arguments

<code>x</code>	A result from <code>gt_reliability()</code> .
<code>coefficient</code>	One or both of "Erho2" and "Phi", without duplicates.
<code>interval</code>	Whether to draw available interval bounds. Missing bounds are never replaced by a zero-width interval.
<code>target</code>	Optional reference line at one number between zero and one. It is a descriptive target, not a confidence statement.
<code>...</code>	Named graphical arguments passed to <code>graphics::plot()</code> , such as <code>col</code> , <code>pch</code> , <code>main</code> , <code>xlim</code> , or <code>sub</code> , replacing defaults. The coefficient coordinates are supplied by the method.

Details

Outcome rows and a weighted composite have distinct labels. The horizontal axis names the observed or latent coefficient scale. The default subtitle identifies unavailable or hidden intervals, conditional uncertainty, and extrapolation. A caller overriding the subtitle should preserve the interpretation relevant to their analysis. For a design that declared batches with [gt_batch](#), the plot also states above the plotting region that they are not modelled; that statement is not a subtitle and a caller's sub does not replace it. Unavailable estimates keep their label but have no plotted point. Changing the graphic does not refit the model or alter any acceptance or inference decision.

On narrow devices, long labels are shortened with an ellipsis and a bracketed row number referring to `as.data.frame(x)`. Full outcome names remain in that table. Margins are restored after plotting.

Discrete fits have point estimates only. Their latent-response coefficients do not describe observed label proportions or majority votes. Gaussian interval and boundary qualifications are described in [gt_reliability](#). The plotting method uses base graphics; PDF and PNG devices can save the same figure without additional plotting dependencies.

Value

Returns `x` invisibly.

See Also

[gt_reliability](#), [gt_dstudy](#), [gt_preflight](#)

print.summary.gt_fit *Print a Concise Generalizability Model Summary*

Description

Displays model scope, numerical acceptance, likelihood, source variances, and fixed estimates without printing the fitted backend model or observation data.

Usage

```
## S3 method for class 'summary.gt_fit'
print(x, ..., digits = max(3L,
  getOption("digits") - 3L))
```

Arguments

x	A summary.gt_fit object returned by summary(fit).
...	Additional arguments; currently unused.
digits	Number of significant digits for printed estimates, from 1 to 22.

Details

The printed summary reports the selected attempt, recorded failed attempts, acceptance failures, natural covariance boundaries, artificial parameter bounds, and likelihood-approximation status separately. If the native Gaussian transcript does not identify a unique selected trial, the output says “not identified”. An accepted fit can have earlier failed attempts or a natural covariance boundary. A rejected fit’s estimates are printed for diagnosis only; it remains ineligible for reliability or decision-study calculations.

The printed variance table shows at most 20 rows and the attempt-failure table at most 6 rows. Complete tables and covariance matrices remain in the summary.

Use `gt_diagnostics(fit)` for full diagnostic details.

Fixed location or contrast estimates are printed on the fitted family’s scale. Numerical acceptance and first-order Laplace approximation adequacy remain separate; the summary makes no additional claim of statistical validation.

Value

Returns x invisibly. The summary object remains a list with outcome and family specifications, source covariance matrices, a source-variance table, means or coefficients, thresholds, likelihood, and diagnostics.

See Also

[gt_fit](#), [gt_diagnostics](#), [gt_components](#)

Examples

```
d <- expand.grid(item = 1:12, rater = 1:3)
d$score <- sin(d$item) + d$rater / 5 + cos(d$item * d$rater) / 4
design <- gt_design("item", "rater", random = ~ item + rater)
fit <- gt_fit(d, "score", design,
  control = gt_control(gaussian = list(check_hessian = FALSE, retry_seed = 42)))
s <- summary(fit)
print(s, digits = 3)
s$variances
```

Index

- * **datasets**
 - gtheory_datasets, 4
 - gtheory_drug_review, 8
 - gtheory_hate_speech, 9
 - gtheory_mental_health, 11
- as.data.frame.gt_batch_dstudy (gt_batch_dstudy), 16
- as.data.frame.gt_batch_reliability (gt_batch_reliability), 17
- as.data.frame.gt_dstudy (gt_dstudy), 29
- as.data.frame.gt_plan (gt_plan), 42
- as.data.frame.gt_reliability (gt_reliability), 51
- barplot, 47
- coef.gt_fit (gt_fit_methods), 38
- Dataset overview, 9, 11, 13, 34
- dataset overview, 3, 8–12, 33
- Drug-Review, 3, 33
- Drug-Review data, 7, 34
- Drug-Review data topic, 6
- drug_review (gtheory_drug_review), 8
- drug_review_4aspect (gtheory_drug_review), 8
- glm, 4
- gt_batch, 13, 19, 26, 31, 36, 49, 53, 60
- gt_batch_dstudy, 15, 16, 19, 53
- gt_batch_reliability, 3, 15–17, 17, 53
- gt_component_vcov (gt_fit_methods), 38
- gt_components, 20, 39, 61
- gt_control, 21, 29, 37, 41, 47
- gt_design, 4, 14, 15, 25, 31, 37, 46, 53
- gt_diagnostics, 20, 24, 27, 37, 39, 51, 57, 61
- gt_dstudy, 4, 15, 17, 29, 42, 45, 51, 54, 57–60
- gt_dstudy_target, 45
- gt_dstudy_target (gt_dstudy), 29
- gt_example, 4, 7–9, 11, 13, 32
- gt_export_report (gt_report), 55
- gt_family, 9, 11, 13, 26, 34, 34, 46
- gt_fit, 4, 6, 20, 24, 26, 29, 34, 35, 35, 39, 46, 51, 59, 61
- gt_fit_methods, 38
- gt_pilot_plan, 4, 40, 59
- gt_plan, 3, 42
- gt_preflight, 4, 14, 15, 46, 57, 60
- gt_reliability, 4, 6, 15, 19, 31, 32, 39, 41, 42, 44, 45, 51, 51, 57, 58, 60
- gt_report, 15, 55
- gt_score, 13, 19, 32, 40, 54, 57
- gt_simulate, 4, 40–42, 58
- Gtheory4LLM (Gtheory4LLM-package), 3
- Gtheory4LLM-datasets (gtheory_datasets), 4
- Gtheory4LLM-package, 3
- gtheory_datasets, 4
- gtheory_drug_review, 8
- gtheory_hate_speech, 9
- gtheory_mental_health, 11
- Hate-Speech, 3, 33
- Hate-Speech data, 7, 34
- Hate-Speech data topic, 6
- hate_speech (gtheory_hate_speech), 9
- image, 47
- logLik.gt_fit (gt_fit_methods), 38
- Mental-Health, 3, 33
- Mental-Health data, 7, 34
- Mental-Health data topic, 6
- mental_health_3group (gtheory_mental_health), 11
- mental_health_3L (gtheory_mental_health), 11
- mental_health_6flag (gtheory_mental_health), 11

mental_health_7L
 (gtheory_mental_health), 11
mental_health_nominal
 (gtheory_mental_health), 11

nobs.gt_fit(gt_fit_methods), 38

plot.gt_dstudy(gt_dstudy), 29
plot.gt_preflight(gt_preflight), 46
plot.gt_reliability, 59
print method, 37
print.gt_batch_dstudy
 (gt_batch_dstudy), 16
print.gt_batch_reliability
 (gt_batch_reliability), 17
print.gt_diagnostic_stage
 (gt_diagnostics), 27
print.gt_diagnostic_stages
 (gt_diagnostics), 27
print.gt_diagnostics(gt_diagnostics),
 27
print.gt_dstudy(gt_dstudy), 29
print.gt_fit(gt_fit), 35
print.gt_plan(gt_plan), 42
print.gt_preflight(gt_preflight), 46
print.gt_reliability(gt_reliability),
 51
print.gt_report(gt_report), 55
print.summary.gt_fit, 61

summary.gt_fit(gt_fit), 35

vcov.gt_fit(gt_fit_methods), 38