

Package: retroglyph (via r-universe)

October 2, 2026

Title A Trusted Mini-Agent to Reconstruct Individual Patient Data from Kaplan-Meier Images

Version 0.0.9

Description The enclosed agent reconstructs individual patient data from published Kaplan-Meier curve images. Relative to other Kaplan-Meier reconstruction tools, it excels at untangling egregiously occluded Kaplan-Meier curves, and the digitization phase achieves a high level of pixel precision.

License MIT + file LICENSE

URL <https://wlandau.github.io/retroglyph/>

BugReports <https://github.com/wlandau/retroglyph/issues>

Depends R (>= 4.5.0)

Imports base64enc, bslib, diffviewer, dplyr, ellmer (>= 0.4.1),
htmltools, htmlwidgets, igraph, IPDfromKM, later, magick,
markdown, promises, R6, R.utils, shiny, shinychat (>= 0.5.0),
stats, survival, tesseract, tibble, tools, utils

Suggests shinytest2, testthat (>= 3.0.0)

Config/testthat/edition 3

Language en-US

Encoding UTF-8

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake libglpk-dev make libmagick++-dev gsfonts libleptonica-dev libjpeg-dev libpng-dev libtiff-dev libuv1-dev libxml2-dev libssl-dev libpoppler-cpp-dev poppler-data libtesseract-dev tesseract-ocr-eng zlib1g-dev

Repository <https://r-multiverse.r-universe.dev>

Date/Publication 2026-10-02 15:31:31 UTC

RemoteUrl <https://github.com/wlandau/retroglyph>

RemoteRef 0.0.9

RemoteSha aa3cf134c14dd7d9d7cd0c570881ed0c610904ff

Contents

retro_agent	2
retro_agent_class	3
retro_app	9
retro_chat_mock	10
retro_chat_replay_simulation	11
Index	12

retro_agent	<i>‘retroglyph’ agent constructor.</i>
-------------	--

Description

Create a ‘retroglyph’ agent object.

Usage

```
retro_agent(chat, state = new.env(parent = emptyenv()))
```

Arguments

chat	An ‘ellmer’ chat object with no registered tools. It is best to use a pragmatic low-cost model (e.g. “sonnet”) for the chat, since sending images to a frontier model can be expensive.
state	An empty environment or empty Shiny ‘reactiveValues’ list to hold agent state. The state must be completely empty when passed to ‘retro_agent()’, and it should be reserved for the agent’s use only.

Value

A ‘retro_agent’ ‘R6’ object.

Examples

```
if (nzchar(Sys.getenv("ANTHROPIC_API_KEY"))) {  
  chat <- ellmer::chat_anthropic()  
  agent <- retro_agent(chat)  
  agent$chat$chat("ping")  
}
```

retro_agent_class	<i>retro_agent R6 class</i>
-------------------	-----------------------------

Description

R6 class for a ‘retroglyph’ agent with an ‘ellmer’ chat and associated state.

Public fields

chat The inner ‘ellmer’ chat object.

state Environment or Shiny ‘reactiveValues’ list holding the state of the agent.

Methods

Public methods:

- `retro_agent$new()`
- `retro_agent$register()`
- `retro_agent$reconstruct()`
- `retro_agent$legend()`
- `retro_agent$risk_table()`
- `retro_agent$events_table()`
- `retro_agent$data()`
- `retro_agent$hazard_ratios()`
- `retro_agent$quantiles()`
- `retro_agent$probabilities()`
- `retro_agent$counts()`
- `retro_agent$compare()`
- `retro_agent$export()`
- `retro_agent$clone()`

`retro_agent$new()`: Create a new ‘retro_agent’ object.

Usage:

```
retro_agent$new(chat, state)
```

Arguments:

chat An ‘ellmer’ chat object.

state An environment for agent state.

`retro_agent$register()`: Register a source image for reconstruction: normalize it to an opaque PNG, store it in ‘state\$image_source’, and optionally clear the chat’s turn history. This is the half of reconstruction that needs no model call, so a Shiny app can call it as soon as the user uploads a file - before the chat loop (driven by ‘shinychat’'s or ‘ellmer’'s own round trip to the model, triggered separately by the user typing into the chat) ever starts. `[retro_agent_class]$reconstruct()` calls this method itself, so most callers never need to call it directly. Resets ‘state\$image_quantized’ and ‘state\$data_label’ to ‘NULL’ so the four-tool workflow (quantize → label → distill → data) starts fresh.

Usage:

```
retro_agent$register(file, clear = TRUE)
```

Arguments:

file Character scalar, path to the source image file. Only PNG, SVG, and JPEG files are supported.

clear Logical scalar. If 'TRUE', wipe the chat's turn history, so the next chat turn begins with a fresh conversation and no trace of prior images or turns. The system prompt and registered tools are untouched. Set to 'FALSE' to keep the prior conversation turns, e.g. to reconstruct a new image within an ongoing conversation.

Returns: 'NULL', invisibly.

retro_agent\$reconstruct(): Run the full reconstruction workflow on a source image: register the file (see [retro_agent_class]\$register()), then chat with the model to view the image, label the numbers, distill the image down to its curves, and read the data off them.

Usage:

```
retro_agent$reconstruct(file, prompt = "", clear = TRUE, timeout = 600)
```

Arguments:

file Character scalar, path to the source image file. Only PNG, SVG, and JPEG files are supported.

prompt Character scalar, a user prompt to send along with the image. Contains optional user-provided instructions or context. The most useful thing to put here is each arm's total number of events, if the publication reports it in the text rather than in the figure: the model is instructed to trust a number you state over anything it reads off the image, and a total events count sharpens the estimated censoring in each arm's final interval (see [retro_agent_class]\$events_table()).

clear Logical scalar. If 'TRUE', wipe the chat's turn history before reconstructing, so this call begins with a fresh conversation and no trace of prior images or turns. The system prompt and registered tools are untouched. Set to 'FALSE' to keep the prior conversation turns, e.g. to ask follow-up questions about an already-reconstructed image.

timeout Numeric scalar, the number of seconds of wall clock time to allow the whole conversation before giving up. The limit exists so a model that loses its way cannot keep taking turns and spending tokens indefinitely. Set to 'Inf' to let the conversation run as long as it likes. 'ellmer' has no equivalent setting: 'getOption("ellmer_timeout_s")' bounds a single HTTP request, not the whole tool-calling loop.

Returns: 'NULL', invisibly.

retro_agent\$legend(): Access the color/series legend produced by the distill tool.

Usage:

```
retro_agent$legend(friendly = TRUE)
```

Arguments:

friendly Logical scalar. If 'TRUE', the 'color' column holds the nearest friendly 'grDevices::colors()' name (e.g. "firebrick") instead of the raw hex code, via the same lookup used elsewhere in the package ('grDevices::colors()'). Set to 'FALSE' to keep the raw hex codes.

Returns: A tibble with columns 'series', 'color', and 'reference' (from 'state\$data_legend'), or 'NULL' if the distill tool has not run yet.

`retro_agent$risk_table()`: Access the risk table read by the data tool, in one of three layouts.

Usage:

```
retro_agent$risk_table(mode = "transposed")
```

Arguments:

`mode` Character scalar, one of "transposed", "wide", or "long". "long" is the layout 'state\$data_risk' already uses: one row per series/time entry, columns 'series', 'x', 'patients'. "wide" pivots that into one row per unique 'x' and one column per series, holding 'patients' ('NA' where a series has no entry at that 'x'). "transposed" transposes the wide layout again into one row per series and one column per time point, the layout most published risk tables use directly beneath their Kaplan-Meier figure.

Returns: A tibble in the layout named by 'mode' (see [retro_table_wide()] and [retro_table_transpose()] for the "wide" and "transposed" shapes), or 'NULL' if the data tool has not run yet.

`retro_agent$events_table()`: Access the per-arm total events counts read by the data tool. Optional throughout: unlike the risk table, total events is not required, and coverage may be partial, so a 'NULL' return does not mean the data tool failed to run. Worth reviewing when it is present - the model reads the number off the source image (or takes it from your own prompt), and it changes the reconstruction by sharpening the estimated censoring in each arm's final interval.

Usage:

```
retro_agent$events_table()
```

Returns: A tibble with columns 'series' and 'events', at most one row per series (from 'state\$data_events', the return value of [retro_data_events()]), or 'NULL' if the data tool has not run yet or no arm reported a total.

`retro_agent$data()`: Access the reconstructed survival data. This is an interpreted survival reconstruction: individual patient time-to-event and censoring status, inferred from the digitized, scaled curve data produced internally by the data tool together with the risk table counts via the Guyot et al. (2012) algorithm (see [retro_data_survival()]).

Usage:

```
retro_agent$data()
```

Returns: A tibble with columns 'series', 'time', and 'status' (from 'state\$data_survival', the return value of [retro_data_survival()]), or 'NULL' if the data tool has not run yet.

`retro_agent$hazard_ratios()`: Fit a proportional hazards model to the reconstructed individual patient survival data ([retro_agent_class]\$data()) and report the hazard ratio of every other series relative to a chosen reference series (see [retro_survival_hazard_ratios()]).

Usage:

```
retro_agent$hazard_ratios(
  reference = {
    leg <- self$legend(friendly = FALSE)
    leg$series[leg$reference]
  },
  confidence = 0.95
)
```

Arguments:

reference Character scalar, the series to treat as the reference (denominator) level.

confidence Numeric scalar strictly between 0 and 1, the confidence level for the 'lower'/'upper' interval.

Returns: A tibble with columns 'series', 'estimate', 'lower', 'upper', and 'p_value' (see [retro_survival_hazard_ratios()]), one row per series other than 'reference'.

`retro_agent$quantiles()`: Compute Kaplan-Meier survival quantiles (e.g. median survival) for each series in the reconstructed individual patient survival data ([retro_agent_class]\$data()), at the probabilities requested. The digitized, scaled curve data produced internally by the data tool is deliberately never used for this, even as a fallback: it makes no statistical claims, and reconstruction does not record whether its y-axis was survival or cumulative incidence, so a quantile read off it would not be statistically defensible.

Usage:

```
retro_agent$quantiles(
  probabilities = c(0.25, 0.5, 0.75),
  type = c("survival", "incidence"),
  confidence = 0.95
)
```

Arguments:

probabilities Numeric vector of probabilities between 0 and 1, interpreted according to 'type': as target survival probabilities or as target cumulative incidence probabilities. Order does not matter, and duplicates (after converting to a common scale) are dropped - the returned tibble always reports both 'survival' and 'incidence' for each requested probability, sorted chronologically. A series with a low overall event rate may never reach a given probability within the observed follow-up - see 'time' below.

type Character string, either "survival" or "incidence". Controls how 'probabilities' is interpreted: "survival" treats each value as a target survival probability (fraction still alive); "incidence" treats each value as a target cumulative incidence probability (fraction with the event), i.e. '1 - survival'.

confidence Numeric scalar strictly between 0 and 1, the confidence level for the 'time_lower'/'time_upper' interval.

Returns: A tibble with columns 'series', 'survival', 'incidence' ('1 - survival'), 'time', 'time_lower', and 'time_upper', one row per series/probability combination, sorted chronologically (increasing 'time', i.e. decreasing 'survival') within each series. 'time' (and 'time_lower'/'time_upper') is 'NA' for any probability the reconstructed survival curve never reaches.

`retro_agent$probabilities()`: Compute Kaplan-Meier survival probabilities (e.g. 12-month survival) for each series in the reconstructed individual patient survival data ([retro_agent_class]\$data()), at the times requested. This is the inverse of [retro_agent_class]\$quantiles(): instead of asking "at what time is a target survival probability reached", it asks "what is the survival probability at a target time". The digitized, scaled curve data produced internally by the data tool is deliberately never used for this, even as a fallback: it makes no statistical claims, and reconstruction does not record whether its y-axis was survival or cumulative incidence, so a probability read off it would not be statistically defensible.

Usage:

```
retro_agent$probabilities(quantiles, confidence = 0.95)
```

Arguments:

quantiles Numeric vector of non-negative time points, e.g. 'c(6, 12, 24)' for the survival probabilities at 6, 12, and 24 months. Named 'quantiles' (not 'time') to mirror [retro_agent_class]\$quantiles()'s 'probabilities' argument - despite the name, these are time points, not probabilities. Order does not matter, and duplicates are dropped - the returned tibble is always sorted chronologically. A requested time beyond a series' observed follow-up still returns the Kaplan-Meier estimate at that time, extended flat from the last observation.

confidence Numeric scalar strictly between 0 and 1, the confidence level for the 'survival_lower'/'survival_upper' interval.

Returns: A tibble with columns 'series', 'time', 'survival', 'survival_lower', 'survival_upper', 'incidence' ('1 - survival'), 'incidence_lower' ('1 - survival_lower'), and 'incidence_upper' ('1 - survival_upper'), one row per series/time combination, sorted chronologically (increasing 'time') within each series.

retro_agent\$counts(): Count the number of patients and events per series in the reconstructed individual patient survival data ([retro_agent_class]\$data()). One row per legend row, in legend order, plus a final total row summing across all series (see [retro_survival_counts()]).

Usage:

```
retro_agent$counts()
```

Returns: A tibble with columns 'series', 'patients', and 'events' (see [retro_survival_counts()]), one row per legend row in legend order, plus a final "total" row.

retro_agent\$compare(): Visually compare the source image against a freshly generated impression, with axes drawn back in and one series brought to the front. Two sources are available for the impression (see the 'data' argument): the reconstructed survival data ('state\$data_survival') refit to a Kaplan-Meier curve per series, which validates the thing the package actually exists to produce rather than the raw digitized pixels; or the raw digitized trace ('state\$data_scaled') before reconstruction, which isolates whether a disagreement traces back to retroglyph's own digitization or to 'IPDfromKM's reconstruction. Each series is drawn out to its own last reconstructed observation, so arms with shorter follow-up end earlier in the impression than arms with longer follow-up, exactly as they do in the source figure. Assumes 'reconstruct()' has already completed successfully.

Usage:

```
retro_agent$compare(front = 1L, data = "survival")
```

Arguments:

front Integer scalar, the row of the legend (1 to 'nrow(state\$data_legend)') whose series is drawn on top in the impression; the remaining series are layered beneath it in legend order.

data Character scalar, either "survival" to render the reconstructed survival data ('state\$data_survival') refit to a Kaplan-Meier curve (see [retro_image_layer_survival()]), or "trace" to render the raw digitized trace ('state\$data_scaled') with no refit (see [retro_image_layer_trace()]) - useful for telling apart a retroglyph digitization problem from an 'IPDfromKM' reconstruction problem.

Returns: An HTML widget (from 'diffviewer::visual_diff()') comparing 'state\$image_source' (old, ground truth) to the generated impression (new).

`retro_agent$export()`: Save the reconstruction to disk under the ‘output’ directory (created if it doesn’t already exist), split into two subdirectories: * ‘compare/’: one self-contained HTML comparison widget per legend row, each bringing that row’s series to the front (see `[retro_agent_class]$compare()`). Files are named ‘<series>_<color-name>.html’, where ‘<color-name>’ is the nearest built-in R color name (from ‘`grDevices::colors()`’) to the series’ hex color. * ‘data/’: ‘risk_table.csv’ (see `[retro_agent_class]$risk_table()`), ‘events_table.csv’ (see `[retro_agent_class]$events_table()`), ‘data.csv’ (see `[retro_agent_class]$data()`), ‘counts.csv’ (see `[retro_agent_class]$counts()`), ‘quantiles.csv’ (see `[retro_agent_class]$quantiles()`), ‘probabilities.csv’ (see `[retro_agent_class]$probabilities()`), and ‘hazard_ratios.csv’ (see `[retro_agent_class]$hazard_ratios()`) whenever those are non-‘NULL’. ‘events_table.csv’ is absent whenever no arm reported a total events count, which is common and not an error. ‘probabilities.csv’ is absent unless ‘quantiles’ (below) is supplied. Assumes ‘reconstruct()’ has already completed successfully.

Usage:

```
retro_agent$export(
  output,
  data = "survival",
  probabilities = c(0.25, 0.5, 0.75),
  type = c("survival", "incidence"),
  quantiles = NULL,
  confidence = 0.95
)
```

Arguments:

`output` Character scalar, path to a directory to write the ‘compare/’ and ‘data/’ subdirectories into. ‘`export()`’ deletes ‘output’ before writing to it, so be careful about the choice of output directory.

`data` Character scalar, either “survival” or “trace”, forwarded to `[retro_agent_class]$compare()` for every legend row - see its ‘data’ argument.

`probabilities` Numeric vector, forwarded to `[retro_agent_class]$quantiles()` for ‘quantiles.csv’.

`type` Character string, forwarded to `[retro_agent_class]$quantiles()` for ‘quantiles.csv’.

`quantiles` Numeric vector, forwarded to `[retro_agent_class]$probabilities()` for ‘probabilities.csv’. Defaults to ‘NULL’, which skips ‘probabilities.csv’ - there is no dataset-agnostic default set of time points.

`confidence` Numeric scalar, forwarded to both `[retro_agent_class]$quantiles()` (for ‘quantiles.csv’) and `[retro_agent_class]$probabilities()` (for ‘probabilities.csv’).

Returns: ‘NULL’, invisibly.

`retro_agent$clone()`: The objects of this class are cloneable with this method.

Usage:

```
retro_agent$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

retro_app	<i>Launch a ‘retroglyph’ Shiny app.</i>
-----------	---

Description

Create and return a Shiny app that wraps a [retro_agent()] in a browser-based UI for uploading Kaplan-Meier images, chatting with the model, and reviewing the reconstructed image, risk table, hazard ratios, and survival quantiles/probabilities.

Usage

```
retro_app(
  chat,
  theme = bslib::bs_theme(version = 5, preset = "flatly", primary = "#2C3E50"),
  authentication_ui = NULL,
  authentication_server = NULL,
  onStart = NULL,
  options = list()
)
```

Arguments

chat	An ‘ellmer’ chat object with no registered tools, as in [retro_agent()]. Cloned once per session.
theme	A ‘bslib’ theme object from ‘bslib::bs_theme()’, passed to ‘bslib::page_sidebar()’. The default is the Bootstrap 5 “flatly” preset with a dark navy primary color. The app puts a light/dark mode toggle (‘bslib::input_dark_mode()’) in the top right of the main panel and starts in light mode. Bootstrap 5 color modes mean a single theme covers both modes.
authentication_ui	‘NULL’, or a Shiny tag/taglist to insert into the app’s UI, outside the main navigation panels. Use this to layer on authentication UI (e.g. an SSO login gate) without ‘retroglyph’ itself depending on any authentication package.
authentication_server	‘NULL’, or a function with arguments ‘input’, ‘output’, ‘session’, called first inside the app’s server function. Use this together with ‘authentication_ui’ to layer on server-side authentication logic.
onStart	See ‘shiny::shinyApp()’.
options	See ‘shiny::shinyApp()’.

Details

‘chat’ is cloned once per Shiny session (via ‘chat\$clone(deep = TRUE)’), so the same ‘chat’ object can be reused across multiple concurrent app sessions without one session’s conversation or registered tools leaking into another’s.

Value

A Shiny app object from `'shiny::shinyApp()'`.

Examples

```
if (identical(Sys.getenv("RETROGLYPH_EXAMPLES"), "true")) {  
  retro_app(retro_chat_replay_simulation())  
}
```

retro_chat_mock	<i>Mock chat constructor</i>
-----------------	------------------------------

Description

Create an ‘ellmer’ chat object that does not authenticate or connect to any real provider. Useful for testing and development without network access.

Usage

```
retro_chat_mock()
```

Details

The returned chat supports registering tools, setting system prompts, and all other local operations. It will error only if you attempt to send a message to the model.

Value

An ‘ellmer’ chat object.

See Also

Other chats: [retro_chat_replay_simulation\(\)](#)

Examples

```
chat <- retro_chat_mock()  
chat$get_system_prompt()
```

retro_chat_replay_simulation
Replay chat constructor

Description

Create an ‘ellmer’ chat object that replays a fixed, real tool-call transcript instead of talking to a model. Useful for exercising [retro_agent()]/[retro_app()] end to end, deterministically and without a network call.

Usage

```
retro_chat_replay_simulation()
```

Details

The returned chat is only good for one image: the ‘inst/simulation.png’ example shipped with the package. Its ‘chat()’/‘stream_async()’ methods replay the exact tool-call sequence a live model once took to reconstruct that specific image - quantize, label, distill, then data, each called with the arguments that model read off ‘inst/simulation.png’ - then return a canned summary of the result. ‘chat()’ returns that summary directly, and ‘stream_async()’ returns a promise of it, so [retro_app()] streams the replay the same way it streams a real model’s response. Because the replayed calls run the real tools [retro_agent()] registers (not canned tool results), the state ends up populated exactly as it would from a real reconstruction of ‘inst/simulation.png’, but any other image will not track this chat’s hard-coded axis calibration and risk table, so results are only sensible for ‘inst/simulation.png’.

Value

An ‘ellmer’ chat object.

See Also

Other chats: [retro_chat_mock\(\)](#)

Examples

```
if (identical(Sys.getenv("RETROGLYPH_EXAMPLES"), "true")) {  
  chat <- retro_chat_replay_simulation()  
  agent <- retro_agent(chat)  
  agent$register(system.file("simulation.png", package = "retroglyph"))  
  agent$chat$chat("Reconstruct the data from the registered plot.")  
  agent$data()  
  if (interactive()) {  
    agent$compare()  
  }  
  # Run the Shiny app against the replay chat instead of a real model:  
  shiny::runApp(retro_chat_replay_simulation())  
}
```

Index

- * **agents**
 - retro_agent, [2](#)
- * **apps**
 - retro_app, [9](#)
- * **chats**
 - retro_chat_mock, [10](#)
 - retro_chat_replay_simulation, [11](#)

retro_agent, [2](#)
retro_agent_class, [3](#)
retro_app, [9](#)
retro_chat_mock, [10](#)
retro_chat_mock(), [11](#)
retro_chat_replay_simulation, [11](#)
retro_chat_replay_simulation(), [10](#)