A History of glmnet

Trevor Hastie Balasubramanian Narasimhan

April 30, 2026

Contents

1. Introduction

`glmnet` is an R package for fitting the lasso and elastic-net regularization paths of generalized linear models and related regression problems. It has been on CRAN since 2008. Over the intervening years it has grown from a single coordinate-descent Fortran kernel for Gaussian, binomial, multinomial, Poisson, and Cox responses into a layered system in which *any* GLM family can be fitted, in which large-scale genomic problems are tractable, and in which every numerical kernel is now written in modern C++. The package reached its current shape in identifiable steps, each due to contributions from several people.

This vignette is a history of those steps and an acknowledgement of the contributions. While some historical information is in `NEWS.md`, it misses the details. We hope that this document goes beyond a listing of enhancements and bug fixes to explain why the package has its current structure, and attributes contributions accurately with references to published work where appropriate.

2. Origins: coordinate descent and the first release (2008–2010)

The statistical backbone of `glmnet` is older than the package. The lasso was introduced in Tibshirani (1996) and the elastic-net penalty, which combines ℓ_1 and ℓ_2 regularization and smooths the variable selection behavior of the lasso in the presence of correlated features, was introduced in Zou and Hastie (2005). The *algorithmic* insight, worked out in Friedman et al. (2007), that cyclical coordinate descent is extraordinarily effective when combined with warm starts along a decreasing sequence of regularization parameters $\lambda_1 > \lambda_2 > \cdots > \lambda_K$ was implemented in its CRAN debut (version 1.1-1). The path is computed once; each λ_k solution starts from the previous, and both the set of active variables and the number of inner passes stay small.

Coordinate descent has a long history, and our team was not the first to use it in conjunction with the lasso. Tibshirani's Ph.D student Wenjiang Fu at U. Toronto invented a version of coordinate descent called the *shooting algorithm* in 1997. Ingrid Daubechies used coordinate descent for ℓ_1 image deblurring in a talk at Stanford in 2002. In her Ph.D. thesis with Jacquie Meulman at U. Leiden in 2006, Anita van der Kooij

used coordinate descent in fitting elastic net problems. Others have used it is well. The *pathwise* use of coordinate descent in `glmnet` leads to great efficiency, and allows for a fairly seamless transition to different loss functions.

The canonical reference for the package itself is Friedman, Hastie, and Tibshirani (2010), published in the *Journal of Statistical Software* the year after the initial release. That paper covers Gaussian, two-class logistic, multinomial, and Poisson regression. The Cox proportional-hazards model for right-censored survival data was added in version 1.2 (2010) by Noah Simon, then a PhD student at Stanford, and documented in Simon et al. (2011). At that point the available families were exactly those exposed through the `family` argument today: `"gaussian"`, `"binomial"`, `"multinomial"`, `"poisson"`, `"mgaussian"`, and `"cox"`. Each one had its own Fortran subroutine, and the dispatch in `glmnet.R` was a chain of `if/else` branches keyed on the family string:

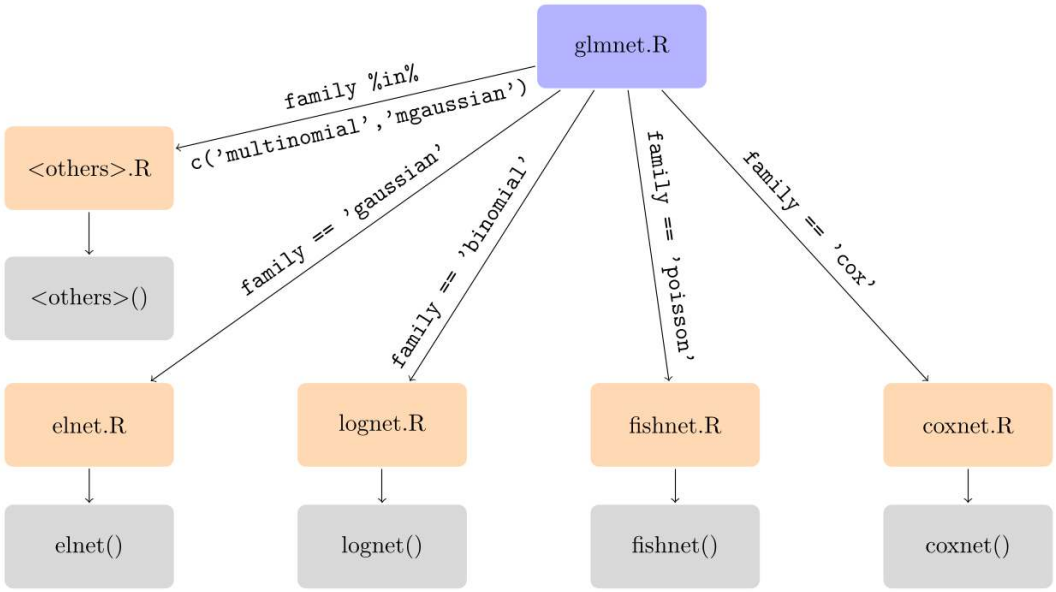

Figure 1: Family dispatch in `glmnet` before v4.0. User-facing R in blue, internal R in orange, Fortran subroutines in grey.

A word on the implementation language is warranted, because it explains most of what came later. The Fortran sources were not written by hand in Fortran but generated from *Mortran*, a macro language layered over Fortran 77. In the spirit of open source, the full Mortran sources live in `inst/mortran/` and were maintained by Jerome Friedman for over a decade. The combination was extremely fast but it was difficult to extend and hard for newcomers to read. Every new feature that touched the inner loop had to be expressed in Mortran or directly in Fortran, regenerated to Fortran, and then debugged under the limitations of that toolchain. Furthermore, Fortran was dropping in popularity and C/C++ was being used more and more for writing fast code to interface with R. So too with `glmnet`.

3. Filling out the matrix (2010–2018)

The years between the first JSS paper and the 2019 v3.0 release were spent in steady, broadening of what the package could accept as input and what it could do with it. We summarize the milestones thematically; the version numbers come from `NEWS.md` and CRAN archive.

More responses. The `mgaussian` family for multivariate Gaussian regression with a grouped-lasso penalty, and a `grouped` option for the multinomial family, arrived in v1.8 (2012). Sparse `x` via `Matrix::dgCMatrix` landed even earlier, in v1.6 (2011). This was a major step making it possible to use `glmnet` with large, sparse design matrices.

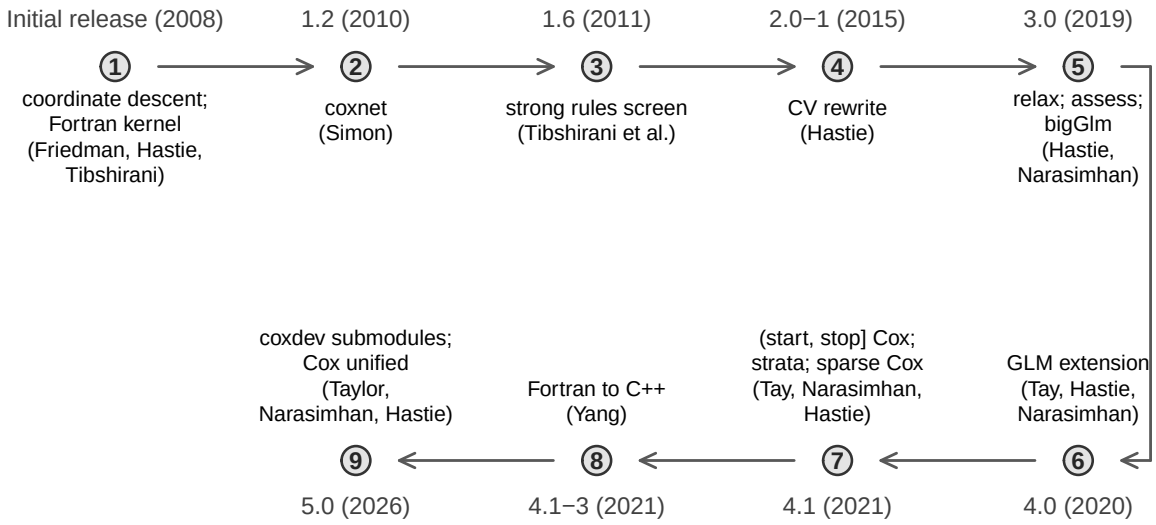

Figure 2: glmnet release milestones. Top row runs left-to-right; the path folds down at the right and the bottom row runs right-to-left, ending at v5.0. Evenly spaced for legibility rather than to reflect calendar time.

Strong rules. Version 1.6 (April 2011) also introduced *strong rules* screening (Tibshirani et al. 2012): at each λ along the path a cheap inner-product check predicts which variables are likely to stay inactive, and the coordinate-descent sweep skips them, with a KKT-condition re-check afterwards to catch any violations. For sparse problems this screens out the vast majority of predictors and was a step-change in speed for large p. Like several other `glmnet` milestones, the code landed in CRAN a year before the paper appeared.

Coefficient constraints. Upper and lower limits on individual coefficients, together with `glmnet.control()` for tuning numerical parameters, appeared in v1.9-1 (2013). The ability to fix a coefficient in an interval — or pin it to zero from outside via `exclude` — is now routinely relied upon by downstream packages.

Cross-validation maturity. The v2.0-1 (2015) release rewrote `cv.glmnet`: each fold fits on its own λ sequence and is then evaluated at the original path. The alignment subtlety that motivated this — paths from different folds do not in general hit the same λ values — was subsequently addressed again in v2.0-18 (2019) via the `alignment` argument.

Cox refinements. A bug affecting ties between the death set and the risk set was fixed in v2.0-19, and `coxgrad` and an improved `coxnet.deviance` were added in v2.0-20. These changes prepared the ground for the larger Cox work that followed in v4.1.

Fortran maintenance. Substantial effort was spent in making the Mortran generated Fortran build cleanly on every CRAN platform: portable on every Fortran compiler (v2.0-15), double-precision-consistent and `-Wall`-clean (v2.0-16, 2018), native Fortran registration (v2.0-8, 2017). These changes were behind the scenes, less visible to users, but essential for passing CRAN checks for another decade.

4. Usability matures: v3.0 (2019)

Version 3.0, released in November 2019, was the first release in which the feature set grew faster than the implementation underneath it. The headline additions, contributed by Trevor Hastie and Balasubramanian Narasimhan, were: the `relax` argument (Hastie, Tibshirani, and Tibshirani 2020) to both `glmnet` and `cv.glmnet`, which refits each active set in the path without regularization and blends the two fits through a `gamma` parameter (a dedicated `relax` vignette documents the usage); `assess.glmnet`, `roc.glmnet`, and `confusion.glmnet` for evaluating model performance; `makeX` for building input matrices from raw data frames with one-hot encoding of factors and NA handling; and `bigGlm` for unpenalized GLM fits using the same algorithmic machinery. A `trace.it` progress bar made long paths less opaque, and `print` methods for `cv.glmnet` and the new `relaxed` class were tidied up.

Importantly, the family dispatch at this point was still entirely character-based. A call `glmnet(x, y, family = "binomial")` passed through an `if` branch into `lognet.R`, which called a Fortran subroutine dedicated to logistic regression. Every family had its own subroutine. That was about to change.

5. GLM extension: v4.0 (2020)

Before Version 4.0, each family reference such as `"binomial"` was routed to a dedicated routine. Version 4.0 (May 2020) generalized the objective

$$\sum_{i=1}^{n} w_i \,\mathrm{NLL}_i(\beta_0, \beta) + \lambda\, P_\alpha(\beta),$$

where NLL_i is the negative log-likelihood contribution of observation i under *any* GLM family, and the elastic-net penalty is $P_\alpha(\beta) = \alpha\|\beta\|_1 + \frac{1-\alpha}{2}\|\beta\|_2^2$. Similar to classical GLMs, we use an iteratively reweighted lasso or elastic net algorithm at each value of `lambda` (along with cyclical coordinate descent) to fit this model, which is implemented pathwise down the sequence of `lambda` values. That is, at each value of `lambda`, we implement the *proximal Newton* via penalized iterative weighted lease squares

$$\frac{1}{2n}\sum_{i=1}^{n} w_i\big(y_i - \beta_0 - x_i^\top \beta\big)^2 + \lambda\, P_\alpha(\beta).$$

This coordinate-descent kernel that serves this inner loop is the weighted least squares routine `wls`, which becomes the engine for every family the package does not handle with a dedicated subroutine.

The programming was done by Kenneth Tay, then a PhD student at Stanford, with supervision from Hastie and Narasimhan. The user-visible payoff is that the `family` argument can now be an R `family()` object — any object of class `"family"` in the sense of `stats::glm`, including `MASS::negative.binomial`, `statmod::tweedie`, or a user-defined family. The published reference is Tay, Narasimhan, and Hastie (2023). Note that the paper appeared three years after the release; the software led the publication.

Two design decisions deserve explicit mention. First, the pre-v4 character-named families were retained and continue to go through their dedicated subroutines, because a hand-written kernel will always beat a generic IRLS loop. So the flexibility was added alongside the old code, not on top of it. Second, the v4.0 warm-start convention was tightened: the earliest form required passing a full `glmnetfit` object, which was awkward; in v4.0.1 this was relaxed so a warm start can be a simple list containing the intercept and coefficient vector. Consistency with the pre-v4 numerical results — verified through KKT-condition checks against the Fortran output — was the correctness criterion throughout.

6. Cox catches up: v4.1 (2021)

Version 4.1 (January 2021) brought the Cox model to something close to feature parity with `survival::coxph`, which has served as the reference implementation throughout `glmnet`'s Cox history. Four things were added in a single release:

1. *Left truncation and time-varying covariates* via $(start, stop]$ counting-process input, the same format `survival` has long accepted.
2. *Strata*, specified via `stratifySurv()`, with the interpretation of `survival::coxph`: a separate baseline hazard per stratum and a single shared coefficient vector.
3. *Sparse* x for Cox models, which had been a notable gap — the pre-v4.1 Cox path required a dense matrix even when the rest of the package was happy with `dgCMatrix`.
4. A `survfit` method for `coxnet` objects so that fitted survival curves could be produced without leaving `glmnet`.

Much of this work was also implemented by Tay, building on the v4.0 IRLS machinery but targeting the Cox deviance specifically. In v4.1-2, following a suggestion by Rob Tibshirani, the `exclude` argument was

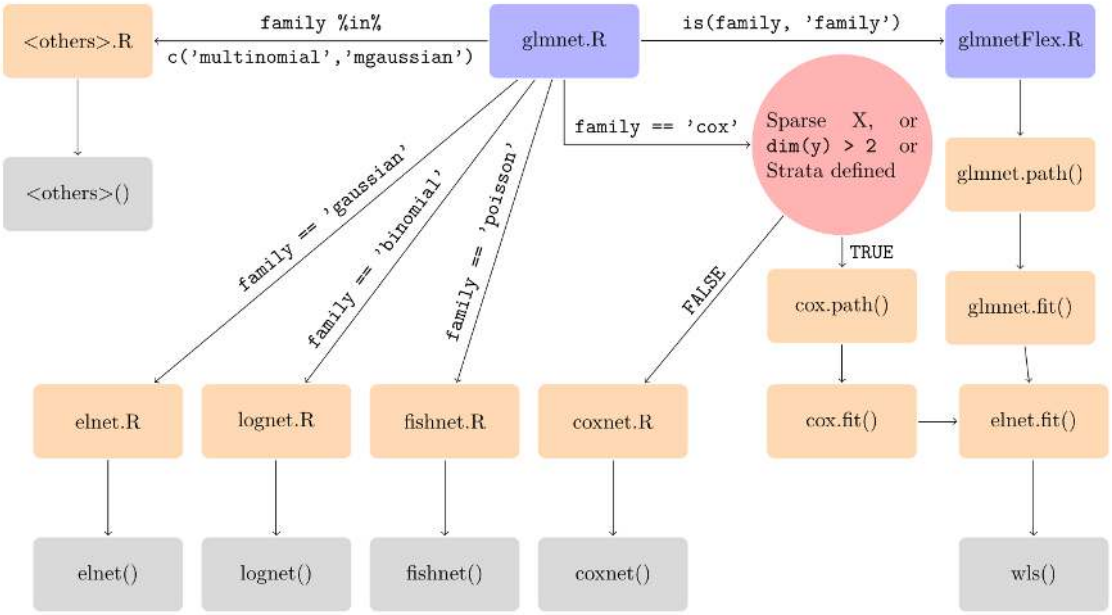

Figure 3: Family dispatch in `glmnet` v4.0. The new right-hand branch (`glmnet.path`, `glmnet.fit`, `wls`) handles any `family()` object; the original per-family subroutines remain for the built-in string-named families, which stay on the fast path.

extended from a fixed set of indices to *a function* that is handed `x`, `y`, and `weights` and returns the indices to drop. This is the user-level hook by which strong-rules screening can be plugged into the path; see Tibshirani et al. (2012) and, for its application at GWAS scale, Qian et al. (2020).

7. Fortran → C++ port (2021–2022)

During the summer of 2021, first year statistics Ph.D student James Yang was looking for a fun activity after completing the grueling qualiying exams. He decided to rewrite the fortran backbone in `glmnet` in C++! This caught the attention of Hastie (who became his advisor) and Narasimhan, because Yang is a very experienced programmer. Between v4.1-3 (November 2021) and v4.1-6 (November 2022), Yang replaced essentially all of the Mortran/Fortran in the package with a header-only C++ implementation built on `Eigen`. This implementation, `glmnetpp`, is a single template kernel that absorbs dense and sparse matrices and all the built-in families through parametrization rather than code duplication.

The staging was deliberate. In v4.1-3 the weighted least squares kernels (`wls` and its sparse counterpart) and the `elnet` family of routines were ported first — these were the engines that the new `glmnet.fit` path exercised the most, and porting them delivered roughly 4–8× speedups on the generic GLM path. In v4.1-4 the rest of the Fortran was rewritten, leaving only the Cox subroutine. In v4.1-6 the legacy Fortran that could be removed was removed.

Several thousand lines of template C++ replaced several thousand lines of Fortran, with comparable or better performance because arguments were no longer copied—R version 3.1+ changed this behavior and `dup = FALSE` had no effect. This provided a much lower barrier to future extension, and removed the dependence on a generated-Fortran toolchain. The Cox subroutine was left alone because its numerical structure — stratified risk sets, ties, left truncation — is harder to express cleanly as a template, and because James Yang's priority was to get the rest of the package onto the new footing. Resolving Cox would take another three years.

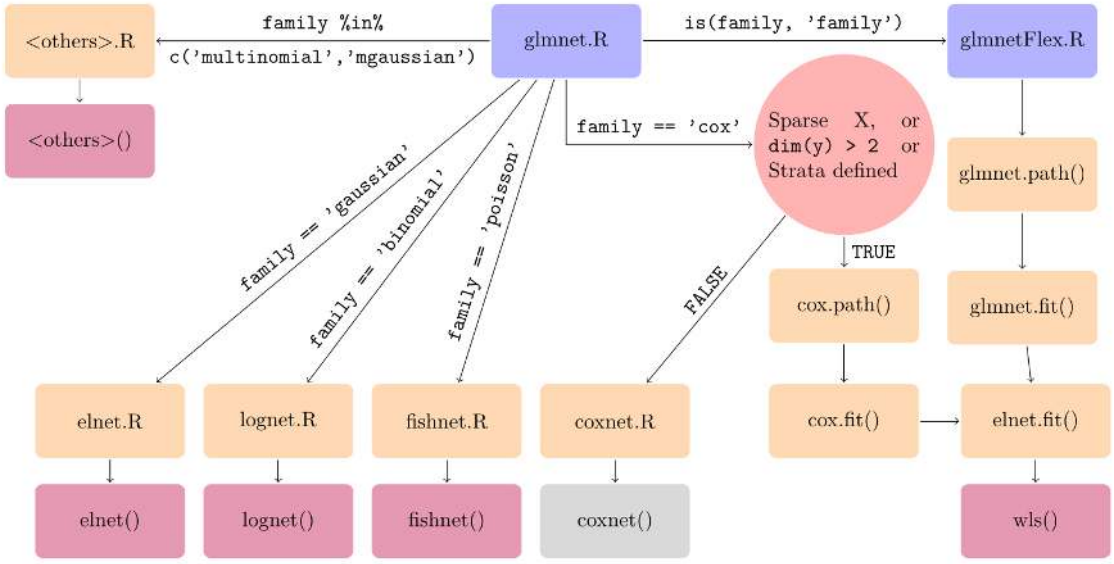

Figure 4: Family dispatch after the C++ port (v4.1-3 onwards). Coloured boxes indicate kernels rewritten in C++; only `coxnet()` remained in Fortran.

8. Applications feeding back into the package

`glmnet` is also a piece of infrastructure on which other statistical software is built, and some of those applications have exerted pressure on the package's interface. We mention three briefly.

SNPnet (Qian et al. 2020) applies the lasso at the scale of modern genome-wide association studies — hundreds of thousands of samples by hundreds of thousands of variants — by aggressively screening predictors using the strong rules of Tibshirani et al. (2012). The work motivated the v4.1-2 generalization of `exclude` to accept a function, so that strong-rules screening can be plugged into the standard `glmnet` call rather than requiring a fork.

The data-shared lasso (Gross and Tibshirani 2016) uses `glmnet` as its optimization backend to share information across related datasets while retaining per-dataset coefficients, an idiom close to multi-task learning.

Cooperative learning (Ding et al. 2022) is a framework for multi-view regression in which multiple data modalities on the same subjects (genomics, proteomics, imaging, and so on) are jointly regressed against an outcome under a regularization term that penalizes disagreement between the views. The companion CRAN package `multiview` dispatches its fits to `glmnet`.

9. Streamlined Cox and v5.0 (2026)

On the eve of v5.0 the package was in an unusual state: every kernel except the Cox subroutine had been rewritten in C++. The Cox kernel had survived because of the combinatorial complexity its users expect of it — tie handling, stratified risk sets, left truncation — and because producing a numerically stable deviance, gradient, and Hessian in all of these cases in a single clean implementation is harder than it looks.

The v5.0 release resolves this. Jonathan Taylor joined the team, and contributed `coxdev`, a standalone C++ library that computes the Cox partial log-likelihood deviance together with its gradient and Hessian, handling Breslow and Efron tie methods, strata, and (*start*, *stop*] data through a small number of well-tested classes. `coxdev` is separately developed (a nested git submodule of `glmnetpp`, which is itself a submodule of `glmnet`), and the computational details are the subject of a companion vignette. At the same time, Narasimhan replaced the Fortran code for the Cox regularization path with C++, making use of the `coxdev` library for the gradient and Hessian computations, which automatically takes care of all the special cases. From the

point of view of `glmnet`, the essential change is that Cox is now a first-class GLM type inside `glmnetpp` — `glm_type::cox` in the type enumeration — and the same IRLS loop that drives the Gaussian, binomial, and Poisson paths drives the Cox path, with a thin `CoxDevAdapter` handing the deviance and gradient queries off to `coxdev`. The last of the Fortran leaves the package in this release.

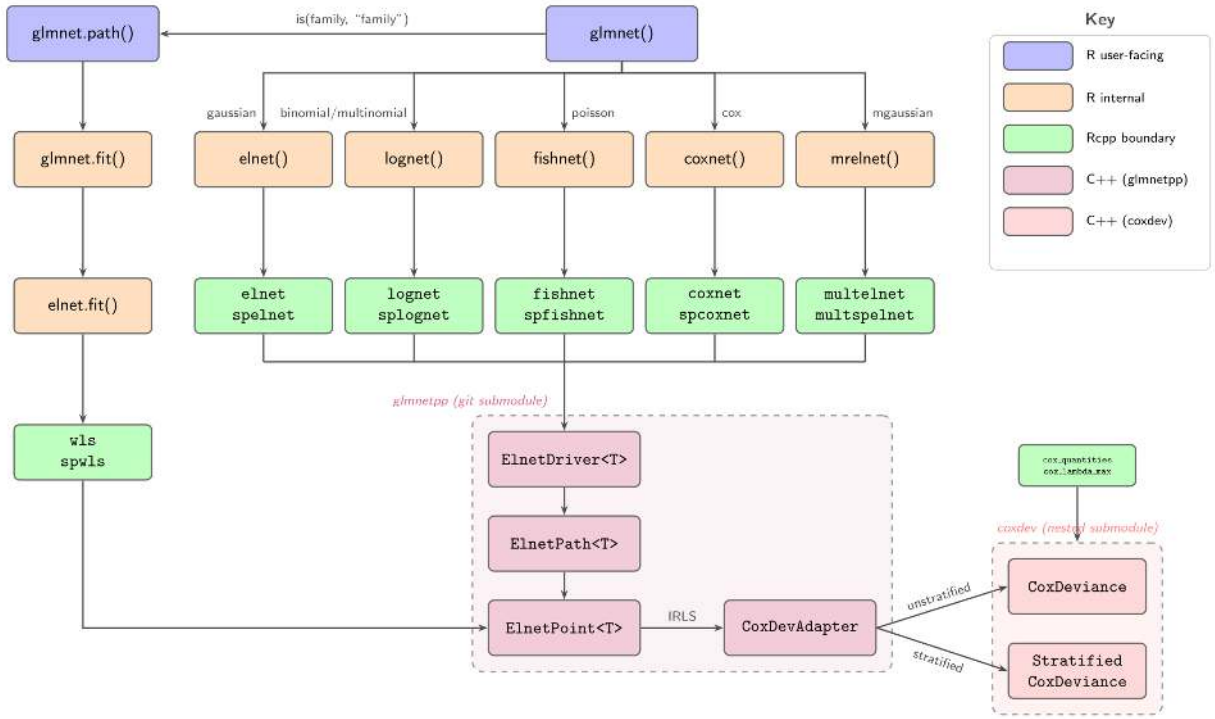

Figure 5: Full architecture of `glmnet` v5.0 with the `glmnetpp` and `coxdev` submodules, the Rcpp boundary, and the `CoxDevAdapter` bridge from the `glmnetpp` IRLS loop to the `coxdev` deviance and gradient routines.

Two user-visible consequences of this unification are worth stating plainly. First, the Cox model now takes a `cox.ties` argument with choices `"breslow"` or `"efron"`. The v5.0 default is `cox.ties = "breslow"`, preserving the pre-v5.0 numerical behavior for upgraders; v5.1 will switch the default to `"efron"` to match `survival::coxph`. Calls to `glmnet()` with `family = "cox"` that do not set `cox.ties` explicitly emit a transition warning until the switch occurs. The performance picture is favourable. On the benchmark set in `tests/benchmark_cox.R`, dense Cox paths are roughly 3 times faster than the legacy Fortran, sparse Cox paths roughly 90 times faster, and stratified Cox paths — which had been the most penalized case in the old implementation — roughly 300 times faster. Full benchmark numbers appear in the NEWS entry for v5.0.

10. Looking forward

Development continues along several lines. A Python `glmnet` is being prepared by Jonathan Taylor that shares a common C++ codebase with the R package, so that the same kernels serve both languages. The shared codebase opens the possibility of compiling `glmnet` to WebAssembly for in-browser use. Upstream work in `coxdev` is aimed at reducing per-call allocation in the IRLS loop, which is the last remaining place where the Cox path pays a measurable overhead for its extensibility.

11. Contributors and acknowledgements

`glmnet` is the work of many hands. Jerome Friedman wrote the original coordinate-descent kernel and the Mortran infrastructure, and is a co-author of Friedman, Hastie, and Tibshirani (2010), Simon et al. (2011), and Friedman et al. (2007). Trevor Hastie has been co-author throughout, as well as the package maintainer.

He also wrote the bulk of the R interface code, as well as methods for plotting, printing, predicting and cross-validation, and the corresponding code for the relaxed models (with help from Narasimhan).

Robert Tibshirani introduced the lasso (Tibshirani 1996) and co-authored the strong-rules work on which the package depends. Noah Simon led the original Cox path work and is the first author of Simon et al. (2011). Kenneth Tay is responsible for the v4.0 GLM extension and much of the v4.1 Cox expansion, and is first author of Tay, Narasimhan, and Hastie (2023). Balasubramanian Narasimhan has maintained the package through the v4.x and v5.0 architectural transitions and is a co-author of Tay, Narasimhan, and Hastie (2023). James Yang carried out the v4.1-3 through v4.1-6 port from Fortran to C++ and is the author of `glmnetpp`. Junyang Qian is first author of Qian et al. (2020), whose screening machinery influenced the v4.1-2 `exclude`-as-function interface. Jonathan Taylor contributed `coxdev` and with Narasimhan drove the v5.0 unification of the Cox path onto the C++ engine. Many bug reports and fixes contributed by users over the years are recorded in `NEWS.md`; the work of Tomas Kalibera, David Keplinger, and others is gratefully acknowledged.

References

Ding, Daisy Yi, Shuangning Li, Balasubramanian Narasimhan, and Robert Tibshirani. 2022. "Cooperative Learning for Multiview Analysis." *Proceedings of the National Academy of Sciences* 119 (38): e2202113119. https://doi.org/10.1073/pnas.2202113119.

Friedman, Jerome, Trevor Hastie, Holger Höfling, and Robert Tibshirani. 2007. "Pathwise Coordinate Optimization." *The Annals of Applied Statistics* 1 (2): 302–32. https://doi.org/10.1214/07-AOAS131.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2010. "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software, Articles* 33 (1): 1–22. https://doi.org/10.18637/jss.v033.i01.

Gross, Samuel M., and Robert Tibshirani. 2016. "Data Shared Lasso: A Novel Tool to Discover Uplift." *Computational Statistics & Data Analysis* 101: 226–35. https://doi.org/10.1016/j.csda.2016.02.015.

Hastie, Trevor, Robert Tibshirani, and Ryan Tibshirani. 2020. "Best Subset, Forward Stepwise or Lasso? Analysis and Recommendations Based on Extensive Comparisons." *Statistical Science*. Institute of Mathematical Statistics. https://doi.org/10.1214/19-STS733.

Qian, Junyang, Yosuke Tanigawa, Wenfei Du, Matthew Aguirre, Chris Chang, Robert Tibshirani, Manuel A. Rivas, and Trevor Hastie. 2020. "A Fast and Flexible Algorithm for Solving the Lasso in Large-Scale and Ultrahigh-Dimensional Problems." *PLOS Genetics* 16 (10): e1009141. https://doi.org/10.1371/journal.pgen.1009141.

Simon, Noah, Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2011. "Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent." *Journal of Statistical Software, Articles* 39 (5): 1–13. https://doi.org/10.18637/jss.v039.i05.

Tay, J. Kenneth, Balasubramanian Narasimhan, and Trevor Hastie. 2023. "Elastic Net Regularization Paths for All Generalized Linear Models." *Journal of Statistical Software* 106 (1): 1–31. https://doi.org/10.18637/jss.v106.i01.

Tibshirani, Robert. 1996. "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society: Series B (Methodological)* 58 (1): 267–88. https://doi.org/10.1111/j.2517-6161.1996.tb02080.x.

Tibshirani, Robert, Jacob Bien, Jerome Friedman, Trevor Hastie, Noah Simon, Jonathan Taylor, and Ryan Tibshirani. 2012. "Strong Rules for Discarding Predictors in Lasso-Type Problems." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 74 (2): 245–66. https://doi.org/10.1111/j.1467-9868.2011.01004.x.

Zou, Hui, and Trevor Hastie. 2005. "Regularization and Variable Selection via the Elastic Net." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67 (2): 301–20. https://doi.org/10.1111/j.1467-9868.2005.00503.x.