Supplementary Information

r2py: AI-Assisted Conversion of
R Statistical Packages to Python

Yufei Cai Jun Li

Department of Applied and Computational Mathematics and Statistics,
University of Notre Dame

Contents

S1 Scope of the supplementary material **2**
S1.1 Phase numbering in the two Notes 2
S1.2 Where each claim is evidenced 3

S2 Complete inventory of reconstructed R C API symbols **4**
S2.1 Resolution of Category E 4
S2.2 Defects found in the generated headers 5

S3 Composition of the language-dependency catalogs **6**
S3.1 Conversion order 6
S3.2 Uniform treatment of constructs without Python equivalents 6

S4 Validation policy **8**
S4.1 Test composition 8
S4.2 Tolerance policy 8
S4.3 Divergences recorded rather than failed 10

S5 Skill and sub-agent specifications **11**

S6 Build and distribution configuration **12**

S7 Deposited data and code **13**

S1 Scope of the supplementary material

Three documents accompany the main text. They serve different purposes and are intended to be read differently.

- **This document** carries evidence for specific claims made in the main text: the complete inventory of reconstructed R C API symbols, the composition of the language-dependency catalogs, the validation policy, and the build configuration. It is intended to be consulted claim by claim, using the index in Table S2.
- **Supplementary Note 1** is the conversion record for `KernSmooth`.
- **Supplementary Note 2** is the conversion record for `rpart`.

The two Notes are contemporaneous engineering records written during the conversions themselves, not summaries composed afterwards. They document every phase as it was carried out, including the diagnostic paths taken through defects that were later fixed. They are supplied so that the process reported in the main text can be audited rather than taken on trust; they are not intended to be read end to end.

S1.1 Phase numbering in the two Notes

The Notes number phases in the order the work was *executed*. The main text numbers them by *structure*: an invariant seven-phase core, preceded by a prologue whose depth is determined by the foreign-function interface. The two schemes describe the same phases and are related as follows.

Supplementary Note 2 (`rpart`, execution order)	**Supplementary Note 1** (`KernSmooth`)	**Main text** (structural)
Phases 1–5	—	Prologue P1–P5
Phases 6–12	Phases 1–7	Core phases 1–7
Phase 13	—	Coda (presentation surface)

Table S1: Reconciliation of phase numbering between the conversion records and the main text. `KernSmooth` required no prologue, so its record numbers the core phases directly.

S1.2 Where each claim is evidenced

Claim in the main text	**Evidence**
The prologue is bounded, and its size is countable by static scan before conversion begins	Section S2 of this document (complete 51-symbol inventory); Note 2, Phases 1–2
Forty-six of the 51 symbols have direct standalone equivalents; only three require an R interpreter	Table S3; Note 2, Phases 3–4
Every base-R construct used by each package was cataloged before any code was generated	Section S3; Note 1, Phase 3; Note 2, Phase 8
The catalogs overlap without nesting	Table S4
Correctness is numerical equivalence at tolerances declared per assertion	Section S4; Note 1, Phase 7; Note 2, Phase 12
Four verification methods were applied, each finding defects the previous missed	Section S4; Note 1, Phases 5–7; Note 2, Phases 10–12
30 of 35 original C files required no modification whatsoever	Note 2, Phase 5
Skills and sub-agents are structured natural-language documents	Section S5; both Notes, Agent and Skill Specification appendices
Wheels are built for five platform families and CPython 3.10–3.14	Section S6

Table S2: Index from claims in the main text to the supplementary material that evidences them.

S2 Complete inventory of reconstructed R C API symbols

The main text reports that `rpart`'s compiled sources reference 51 distinct identifiers from R's C API across 235 reference sites, drawn from 10 R headers, with `Rinternals.h` alone accounting for 165 of the references. Local wrapper macros defined in the package's own headers were excluded from the inventory even where they wrap R API calls, since those travel with the source.

Table S3 gives the inventory in full, grouped into the five categories used in Figure 4 of the main text. Each of the 51 items is implemented in one self-contained header; the directory holds 53 files in total, the two additional ones being the arena allocator `fake_arena.h` and the master include `fake_R.h`, which replaces R's API headers in the package sources and pulls in the allocator ahead of every symbol header. Supplementary Note 2 refers to the 53-file count throughout. The categories are ordered by implementation difficulty rather than by size: A and B are mechanical, C and D require a design decision that is made once, and E is the only category that reaches beyond what standalone C++ can provide.

Category	Symbols
A — Types, enum constants and registration no-ops (19)	`SEXP`, `INTSXP`, `REALSXP`, `STRSXP`, `VECSXP`, `DL_FUNC`, `DllInfo`, `R_CallMethodDef`, `Rboolean`, `TRUE`, `FALSE`, `R_VERSION`, `R_Version`, `R_NilValue`, `R_NamesSymbol`, `R_UnboundValue`, `R_forceSymbols`, `R_registerRoutines`, `R_useDynamicSymbols`
B — Accessors (16)	`INTEGER`, `REAL`, `CHAR`, `PROTECT`, `UNPROTECT`, `LENGTH`, `PRINTNAME`, `ISNAN`, `R_FINITE`, `asInteger`, `asReal`, `isReal`, `ncols`, `nrows`, `SET_STRING_ELT`, `SET_VECTOR_ELT`
C — Allocation and memory management (7)	`allocVector`, `allocMatrix`, `R_alloc`, `R_chk_calloc`, `R_Free`, `mkChar`, `setAttrib`
D — Diagnostics (4)	`error`, `warning`, `Rprintf`, `R_CheckUserInterrupt`
E — R interpreter bridge (5)	`eval`, `findVar`, `findVarInFrame`, `install`, `R_getVar`

Table S3: The 51 R C API identifiers referenced by `rpart`'s compiled sources, by implementation category. Categories A–D (46 symbols) have direct standalone equivalents.

S2.1 Resolution of Category E

Category E is the honest limit of the approach, and its internal structure is what makes the approach viable. Of its five members:

- `R_getVar` is a **compatibility macro**: under the faked R version it expands to a static shim in `rpart`'s own source that calls `findVar` or `findVarInFrame` according to its `inherits` argument. All four call sites pass `inherits=FALSE`, so only `findVarInFrame` is ever reached, and the macro introduces no new requirement.
- `eval` is the only member that **genuinely requires an interpreter**: its callback evaluates the user's splitting and evaluation functions. It is resolved by a function-pointer bridge: Python registers a callback and the C code invokes it when needed.
- `findVar` and `findVarInFrame` are bridged but **require no interpreter**. `findVarInFrame` resolves to a lookup in a registry of buffers materialized before the fit begins; `findVar` is never reached, and its stub returns the unbound sentinel unconditionally.
- `install` **does not require an interpreter**, but is routed through the same bridge. Its callback only has to return a stable handle for each symbol name, which the Python side mints and uses as a registry key. A self-contained interning map is retained in the headers as an alternative, but the bridged path is the one the shipped package takes.

The bridge therefore carries four of the five items, which is why four function-pointer variables underlie it. Registration is not performed at import: all four callbacks are registered on the user-split path itself, so a fit that never supplies a custom splitting rule never registers any of them. In `rpart` the bridge is reachable only when a user supplies a custom splitting rule. All four built-in methods — analysis of variance, Poisson, classification and exponential — execute without ever entering it.

S2.2 Defects found in the generated headers

An audit of the generated headers found seven defects in three classes: include-guard mismatches, in which a header defining a symbol failed to set the flag a later header tested before emitting its own fallback definition; conflicts between `inline` and `static inline` linkage for the same function; and one defect of a different kind, described in the main text, in which the four function-pointer variables underlying the interpreter bridge were declared `static` at namespace scope. Note 2, Phase 4 records each defect and its resolution.

S3 Composition of the language-dependency catalogs

Before any function is translated, every call site of every base-R construct the package uses is recorded, and one translation guide is generated per construct. Table S4 gives the composition of the two catalogs.

	`KernSmooth`	`rpart`
Distinct base-R constructs	46	172
Call sites recorded	509	1,298
Mean call sites per construct	11.1	7.5
Overlap between the two catalogs		
Common to both packages	34	
Peculiar to `KernSmooth`	12	
Peculiar to `rpart`	138	

Table S4: Composition of the language-dependency catalogs. The two sets overlap without nesting: converting `rpart` after `KernSmooth` reused 34 constructs and added 138, so the catalog was enlarged substantially rather than drawn upon unchanged.

Each guide has four parts: an overview of the construct and its translation hazards; a contextual usage analysis enumerating every call site grouped by usage pattern; a general conversion strategy; and step-by-step examples pairing R code with its Python translation for each distinct pattern. Because each guide enumerates the usage patterns observed in *its* source package, a construct already cataloged may still need extending when a new package exercises it differently.

The complete guide sets are deposited with the software (Section S7): 46 guides for `KernSmooth` and 172 for `rpart`.

S3.1 Conversion order

The dependency graph produced by phase 1, and the levels derived from it, are shown as Figure 2 of the main text. The underlying `dependency_levels.csv` artifacts—one per package, recording each function's level, immediate callers and immediate callees—are deposited with the software (Section S7).

S3.2 Uniform treatment of constructs without Python equivalents

R constructs with no direct Python counterpart were handled uniformly across both conversions, the decision being taken once and recorded in the guides: named lists became dictionaries; `missing()` became a module-level sentinel object tested by identity; S3 class

assignment became a reserved dictionary key; environments became dictionaries; 1-based indices were converted at every crossing; and matrices passed to compiled code were laid out in column-major order.

S4 Validation policy

S4.1 Test composition

	`r2py_kernsmooth`	`r2py_rpart`
Public functions tested	7	23
Tests in the suite	518	846
Passing	518	844
Expected failures	0	2
Historical R test scripts translated	2	14
Scripts that exposed defects	—	6 of 14
Cataloged behavioral divergences	94	

Table S5: Composition of the two test suites. The two expected failures record R's echo of the unevaluated call expression, which a Python object retaining only evaluated arguments cannot reproduce. `KernSmooth` ships two regression scripts (`bkfe.R` and `locpoly.R`); `rpart` ships fourteen.

S4.2 Tolerance policy

Table S6 reports the tolerance structure of both released suites, derived directly from the test sources. Three features are worth noting. Negative tests declare no numerical tolerance at all, because they assert that an invalid input is refused rather than that a value is reproduced. The median assertion that compares output against R is 10^{-6} for `r2py_kernsmooth` and 10^{-7} for `r2py_rpart`; the modal value differs by category, and is given per category in the table. And the extremes of each range are attached to identifiable classes of quantity rather than scattered arbitrarily.

Package	**Category**	**Files**	**Test functions**	**`rtol` declarations**	**Range**	**Modal**
`r2py_kernsmooth`	positive	7	240	50	10^{-10}–0.15	10^{-9}
	negative	7	92	0	—	—
	boundary	7	151	67	10^{-10}–10^{-1}	10^{-5}
`r2py_rpart`	positive	23	268	77	10^{-12}–10^{-4}	10^{-6}
	negative	23	201	0	—	—
	boundary	23	238	54	10^{-12}–10^{-4}	10^{-6}

Table S6: Declared relative tolerances by test category, counted from the released test suites. *Files* counts test modules, *Test functions* counts them before parametrisation — so the totals are smaller than the 518 and 846 tests collected from the suites (Table S5) — and ***`rtol`** declarations* counts explicit `rtol` arguments. Each public function contributes one module per category, giving 7×3 and 23×3 modules; a small number of additional modules carry integration tests and are omitted here.

The eight loosest assertions in `r2py_kernsmooth`, all with rtol $\geq 10^{-2}$, were read individually: each bounds a self-consistency property of the plug-in bandwidth selectors—convergence in grid size, scale invariance, monotonicity in the `level` argument, and agreement between alternative scale estimators—rather than agreement with R. The ranges in Table S6 are raw counts over all `rtol` declarations and therefore include both kinds of assertion.

Where the extremes fall. In `r2py_rpart` every assertion at 10^{-12} is in the `meanvar` suite, which reads means and variances directly off the fitted tree, and all four assertions at 10^{-4} are in the `xpred.rpart` suite, whose values are the output of one tree refit per fold, with fold membership supplied explicitly and identically to both implementations; the looseness therefore reflects error accumulated in refitting and combining across folds, not a differing partition. The ordering the main text describes—tightest on quantities read from the tree, loosest on cross-validated quantities—is therefore a property of the suite as written, not a post-hoc characterization.

In `r2py_kernsmooth` the two loosest *agreement* bounds, 10^{-3} and 10^{-4}, are both two-dimensional binned density estimates at extreme scales or with asymmetric bandwidths. Eight further assertions use bounds between 10^{-2} and 0.15; none of these compares against R. They bound properties of the plug-in bandwidth selectors—convergence in grid size, scale invariance, monotonicity in the `level` argument, and agreement between scale estimators—for which a loose bound is the substantive claim rather than a weakened one.

Tolerances are fixed at the point of comparison rather than set globally for the suite, because the appropriate tolerance differs by quantity; where an assertion states none, it inherits the default of the comparison routine used (10^{-7} for NumPy's `assert_allclose`, 10^{-5} for `allclose`). The relative tolerances are:

- `r2py_kernsmooth`: from 10^{-10} to 10^{-3}. The tightest apply to grid coordinates, which are determined exactly by the inputs; 10^{-9} to the binned kernel functional; 10^{-6}–10^{-5} to density and local-polynomial regression estimates; and 10^{-5} to the three scalar plug-in selectors.
- `r2py_rpart`: from 10^{-12} to 10^{-4}. The tightest apply to quantities read directly from the fitted tree; the loosest to cross-validated quantities, whose value depends on the partition of the data into folds.

Reference values are obtained from the original R packages at test time through `rpy2`, so the comparison is against R actually running rather than against values transcribed at some earlier date. Where a suite is required to run without an R installation — as for `rpart`'s translated historical suite — reference values and datasets were captured once from a live R session into checked-in files.

S4.3 Divergences recorded rather than failed

Not every difference between the two languages is a defect in the translation. Where R and Python diverge for reasons intrinsic to the languages, the test records the divergence rather than failing, so that it is documented instead of silently accommodated. Ninety-four such divergences were cataloged, in which both languages correctly reject the same invalid input but describe the rejection in their own native vocabulary.

S5 Skill and sub-agent specifications

Each phase except two — build infrastructure and the equivalence audit — is carried out by a *skill* — an orchestrator that assembles the inputs a unit of work requires and dispatches it — and by the *sub-agents* the skill invokes, each a separate language-model invocation with its own context.

Both are structured natural-language documents, not code. Each is a Markdown file with a YAML front-matter block declaring a name and a one-line description, followed by a body giving execution steps and an output schema. Skills reside in `.claude/commands/` and sub-agents in `.claude/agents/` in each project tree. They are reproduced in full in the Agent Specification and Skill Specification appendices of Supplementary Notes 1 and 2, and are deposited with the software (Section S7).

Dispatch mode follows from the structure of the work rather than from convenience:

- **Sequential**, where an invocation must read an artifact an earlier one produced — function conversion proceeds one function at a time in dependency order, because an agent translating a caller reads the already-translated interfaces of its callees.
- **Parallel**, where invocations are mutually independent — no translation guide depends on any other, so the 46 guides for `KernSmooth` were produced in a single batch and the 172 for `rpart` in nine.
- **Iterative**, where the work is a fix-and-recheck cycle — the suite is re-run, the first remaining failure is handed to a repair agent, and the cycle repeats, which is what allows a defect masked by an earlier one to become visible at all.

S6 Build and distribution configuration

Both packages use `meson-python` as the PEP 517 build backend, but they reach their compiled code differently.

`r2py_kernsmooth` builds a Python extension module from an `f2py` custom target, with NumPy's C and `f2py` include directories resolved at configure time. BLAS is located through a five-stage chain: `pkg-config`, then library searches for `openblaso`, `openblas` and `blas`, then a guarded absolute path for a development cluster whose distribution omits the unversioned symbolic link. If none succeeds the build fails at configure time with an actionable message rather than at link time. Two LINPACK routines providing QR decomposition (`dqrdc`, `dqrsl`) are called by the package but satisfied under R from R's own internal LINPACK copy; because OpenBLAS does not supply them, they were obtained from Netlib and added to the Python package's sources.

`r2py_rpart` builds a shared library loaded at import through `cffi` in ABI mode, requires no external numerical library, and is compiled as C++14 rather than C++17. In C++17 mode the standard math header on the build system declares overloads referencing compiler built-ins that its C library does not provide; the inline-variable declarations in the reconstructed headers compile in C++14 as an accepted extension.

Platform	**Architectures**
Linux (`glibc`)	`x86_64`, `aarch64`
Linux (`musl`)	`x86_64`, `aarch64`
macOS	`x86_64`, `arm64`
Windows	`x86_64`

Table S7: Binary wheel coverage. Wheels are built in CI with `cibuildwheel` across five runners for CPython 3.10 through 3.14, and published on release through OIDC trusted publishing. Windows builds obtain their toolchain from MSYS2, and dependent libraries are vendored into the wheel.

S7 Deposited data and code

Every intermediate artifact of both conversions is deposited publicly, not only the resulting packages. The deposits include the structural-analysis JSON, the language-dependency catalogs and the complete guide sets, the reconstructed C API headers and their implementation blueprints, the per-function conversion artifacts, the complete test suites, and the skill and sub-agent specifications.

Resource	**Location**
Conversion projects	`https://github.com/r2py-project/python-KernSmooth` `https://github.com/r2py-project/python-rpart`
Released packages	`https://github.com/r2py-project/r2py_kernsmooth` `https://github.com/r2py-project/r2py_rpart`
Archived snapshots	10.5281/zenodo.21853796 (`KernSmooth`) 10.5281/zenodo.21853814 (`rpart`)
Distribution	`pip install r2py_kernsmooth` `pip install r2py_rpart`

Table S8: Where each class of artifact is deposited. The archived snapshots are version DOIs corresponding to the state described in the main text.

Table S9 gives the counts of each artifact class, taken directly from the deposited repositories.

Artifact	`KernSmooth`	`rpart`
Skill specifications	10	20
Sub-agent specifications	8	17
Translation guides (one per construct)	46	172
C API implementation blueprints	—	51
Reconstructed headers	—	53
Python test modules	23	84

Table S9: Deposited artifacts by class. Two of `rpart`'s translation guides are dot-prefixed—`.getXlevels.md` and `.checkMFClasses.md`, after the R constructs they document—and are therefore invisible to shell globbing; a directory listing is required to see all 172. The reconstructed headers include two headers whose names differ only in case, so a checkout on a case-insensitive filesystem will show 52 rather than 53.

The converted packages follow the licensing of their originals: `r2py_kernsmooth` is distributed under the Unlimited license of `KernSmooth` 2.23-26, and `r2py_rpart` under GPL-2, since it reuses `rpart` 4.1.27's GPL-licensed C source.