From ad-hoc scripts to institutional infrastructure: Building a reproducible data pipeline for confidential early childhood administrative records

JoonHo Lee[1*], Alison Hooper[1]

1 College of Education, The University of Alabama, Tuscaloosa, Alabama, United States of America

* jlee296@ua.edu

Abstract

Processing administrative data for research presents a trilemma: workflows must be transparent for methodological credibility, consistent across years and personnel, and privacy-preserving for regulatory compliance. Current practice relies on ad-hoc scripts maintained by individual researchers—an arrangement vulnerable to personnel turnover, undocumented decisions, and silent errors. We present `ALprekDB`, an open-source R package that standardizes the processing of confidential administrative records from Alabama's First Class Pre-K program using a "public code, private data" architecture. All processing logic, validation rules, and documentation are publicly available, while confidential data never leave the secure environment. Key features include data-driven CSV codebooks readable by non-programmers, automatic format detection across file versions, a 37-check validation framework, synthetic data generators for public documentation without privacy risk, and integration with the `targets` pipeline for reproducible orchestration. We evaluate the system on four years of production data (5,867 classroom-years; 92,507 student-year records) and map each architectural decision to the competing goals of transparency, consistency, and privacy. Our results demonstrate that careful software engineering—particularly the combination of externalized codebooks, systematic validation, and synthetic data—can resolve this trilemma for confidential administrative records.

Introduction

Administrative records generated by government programs represent one of the most valuable yet underutilized resources for social science research. Compared with survey data, administrative records offer population-level coverage without sampling bias, longitudinal tracking of individuals across years, low marginal cost because the data are already collected for program management, and measurement of real-world behaviors rather than self-reported attitudes [1]. In education, state agencies routinely collect detailed records on enrollment, attendance, student assessments, teacher qualifications, and program expenditures. The early childhood education sector generates particularly rich data because publicly funded programs such as Head Start and state prekindergarten have detailed reporting requirements tied to accountability mandates. Yet the pathway from these raw records to research-ready datasets remains fragile and largely undocumented.

Despite its promise, the typical workflow for processing administrative data follows a familiar and precarious pattern: a state agency transmits Excel files to a university

research partner, an individual researcher writes ad-hoc cleaning scripts, and those scripts evolve informally over successive years of data. This arrangement creates three institutional vulnerabilities. First, *personnel dependency*: when the responsible researcher leaves the project, their scripts may be lost, poorly documented, or incomprehensible to successors. Second, *silent errors*: without systematic validation, data quality issues—miscoded variables, duplicate records, broken cross-references—propagate undetected through downstream analyses. Third, *undocumented decisions*: the myriad choices involved in data cleaning (how to handle missing values, how to classify teacher credentials, how to allocate shared costs) are buried in code that may never be shared or explained [2]. These are not failures of individual diligence but structural consequences of an incentive system that rewards publications over infrastructure.

Efforts to improve this situation encounter a fundamental tension among three competing goals: transparency, consistency, and privacy. *Transparency* requires that processing decisions be visible and auditable; journals and funders increasingly mandate code sharing as a condition of publication. *Consistency* requires that identical cleaning and transformation rules be applied across years and by different personnel; without it, longitudinal comparisons are confounded by processing artifacts. *Privacy* requires that education records protected by the Family Educational Rights and Privacy Act (FERPA) and state data use agreements remain confidential; even processing code may inadvertently reveal data structure. These three goals create pairwise tensions: making code public may expose data structure (transparency vs. privacy), documenting every decision in code creates rigidity that resists necessary adaptation (transparency vs. consistency), and enforcing identical processing requires detailed logs that may accumulate personally identifiable information (consistency vs. privacy). Previous work has addressed these tensions in pairs—Graham, Gooden, and Martin [3] examine the transparency–privacy paradox in public sector data, and Vilhuber [4] reconciles reproducibility with confidentiality—but the three-way tension has not been addressed as an integrated design problem.

We present `ALprekDB`, an open-source R package that resolves this trilemma through a "public code, private data" architecture. The package standardizes the processing of administrative records from Alabama's First Class Pre-K (FCPK) program across three data modules—budgets, classroom characteristics, and student outcomes—implementing five key architectural features: data-driven codebooks stored as external CSV files, automatic detection of file format changes across years, a 37-check validation framework, synthetic data generators for public documentation, and integration with the `targets` pipeline toolkit [5] for reproducible orchestration. The package is publicly available on GitHub (`https://github.com/joonho112/ALprekDB`) with full API documentation (`https://joonho112.github.io/ALprekDB/`). This paper contributes both a practical tool and a case study in infrastructure design for confidential administrative data.

Alabama's FCPK program provides an ideal context for this case study. Established in 2000, the program serves approximately 24,640 children annually in roughly 1,500 classrooms across all 67 Alabama counties, with $181.5 million in annual state spending ($7,368 per child). The program meets all ten quality benchmarks established by the National Institute for Early Education Research (NIEER) and has been rated first in the nation for 19 consecutive years [6]. The program's longevity and data richness make it both a valuable research resource and a challenging data management problem.

The remainder of this paper proceeds as follows. We first review related literature on administrative data, the transparency–privacy paradox, reproducibility, and code packaging. We then describe `ALprekDB`'s design, architecture, and implementation before evaluating the system using four years of production data through the trilemma framework. We close with a discussion of practical implications, comparisons with

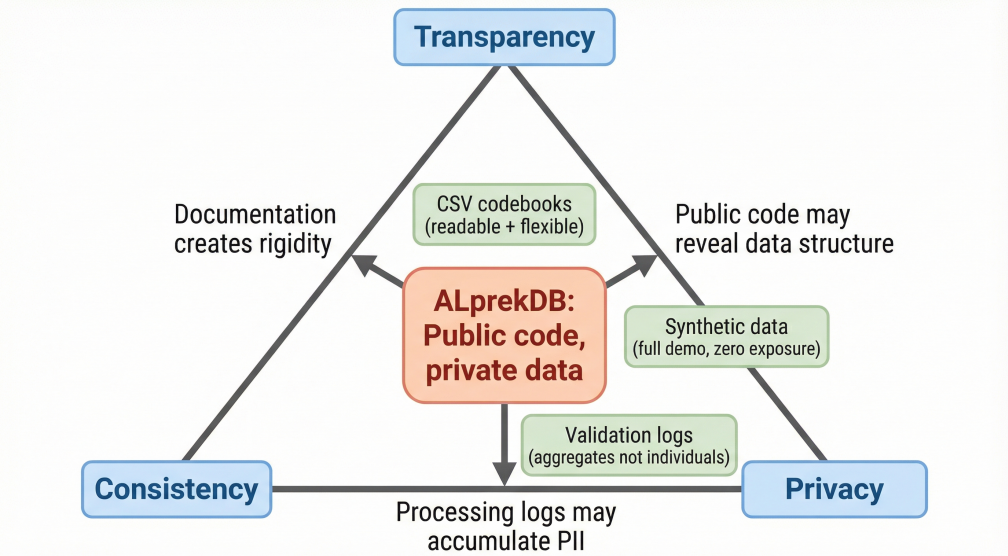

Fig 1. The transparency–consistency–privacy trilemma. Three competing goals in administrative data processing create pairwise tensions. Transparency and privacy conflict because public code may reveal data structure. Transparency and consistency conflict because documentation creates rigidity that resists adaptation. Consistency and privacy conflict because processing logs may accumulate personally identifiable information. The `ALprekDB` package resolves these tensions through a "public code, private data" architecture: code is fully public while data never leaves the secure environment; data-driven codebooks are both readable and flexible; and validation logs describe aggregates, not individuals.

alternatives, and limitations.

Background

Administrative data in education research

The use of administrative data in education research has expanded dramatically over the past two decades. Figlio, Karbownik, and Salvanes [1] provide a comprehensive review of the advantages: population-level coverage eliminates sampling bias, longitudinal tracking enables within-individual analyses, and the marginal cost of data collection is near zero because records are already generated for program management. Card, Chetty, Feldstein, and Saez [7] argue forcefully for expanding research access to administrative records, noting that several countries have built successful research infrastructure around government data. In early childhood education specifically, publicly funded programs generate unusually rich data because accountability requirements mandate detailed reporting on expenditures, workforce qualifications, and child outcomes. A recent survey by the Data Quality Campaign [8] found that 260 early childhood administrators across the United States strongly value data for decision-making but lack the technical infrastructure to use it effectively.

However, administrative data are generated for program management, not research, creating systematic challenges for secondary use. Variables reflect operational categories (budget line items, compliance fields) rather than theoretical constructs. Reporting templates and variable definitions change without advance notice when agencies update their data systems. Codebooks may be incomplete, outdated, or nonexistent. Data

quality varies across sites because different staff enter data according to different local conventions [2]. Spears, Bradburn, Schroeder, Tester, and Forry [9] describe similar challenges with child care subsidy administrative data, where inconsistent definitions across jurisdictions complicate longitudinal analysis.

Despite these challenges, most research teams lack systematic infrastructure for processing administrative data. Fischer et al. [2] describe the Cuyahoga County CHILD integrated data system as an exception, but such systems require substantial investment—often exceeding $200,000 per year with 1.5 full-time equivalent staff. For smaller programs and university–agency partnerships, a lighter-weight approach is needed.

The transparency–privacy paradox

Even when infrastructure exists, a deeper conceptual challenge remains. Graham, Gooden, and Martin [3] identify a fundamental paradox in public sector data: the same records that should be transparent under Freedom of Information Act principles must also be private under FERPA and similar regulations. These two legal frameworks create genuine conflict: processing code may need to be public for accountability and reproducibility, but the data it processes cannot be shared. The paradox is not merely legal but practical: how does one make data processing reproducible when the data cannot accompany the code? Different institutional arrangements—physical data enclaves, remote desktop access, licensed data agreements—each balance these competing demands differently, but none fully resolve the tension.

Vilhuber [4], writing as the American Economic Association's Data Editor, offers a crucial reframing. His key concern, he argues, is not that data may be confidential but rather that some confidential data may be accessible only to a single researcher; when access is so restricted, reproducibility depends entirely on that individual's diligence. If the processing code is public, anyone subsequently granted data access through the same institutional channels can reproduce the analysis. Because code contains processing logic rather than data values, sharing it does not increase disclosure risk. This "share syntax, not data" principle—separating what is shared (code, documentation, validation results) from what is protected (raw and processed data)—is the foundation of the architecture we present.

Recent policy work has begun to formalize these principles. The Federation of American Scientists [10] proposes privacy-preserving research models for education research and development, emphasizing tiered access structures, synthetic data for exploration, and secure computing environments for analysis. Cole, Dhaliwal, Sautmann, and Vilhuber [11] provide comprehensive guidance on using administrative data for research while respecting privacy constraints. These frameworks provide the policy context within which technical solutions like `ALprekDB` operate. Our contribution is a concrete implementation of these principles at the program level.

Reproducibility with confidential data

The transparency–privacy paradox has direct implications for reproducibility. Playford, Gayle, Connelly, and Gray [12] were among the first to directly address the reproducibility challenge for administrative social science data. Their key argument is that the traditional approach to reproducible research—sharing data alongside code—cannot work when data are confidential. Their proposed solution is to share research code through version-controlled repositories alongside published output, enabling others to inspect and verify the processing logic even if they cannot execute it on the original data. A limitation they acknowledge is that without data access, one cannot detect data-dependent bugs or confirm specific output values.

Howison and Goggins [13] present SIRAD (Secure Infrastructure for Research with Administrative Data), a more ambitious approach that integrates multiple agencies' data streams at the state level, providing deidentification, integration, and access management. SIRAD operates with significant infrastructure investment and complements but differs from our approach: SIRAD addresses the "where" of data access through secure computing environments, while `ALprekDB` addresses the "how" of data processing through packaged, validated, and documented code. Both share the principle that processing logic should be transparent even when data cannot be.

Despite growing consensus on code sharing, the quality and reproducibility of shared code remains highly variable. Most shared code consists of standalone scripts with hard-coded file paths, undocumented variable transformations, and implicit assumptions about data format. When agencies change their reporting templates—as they routinely do—such scripts break silently, producing output that appears valid but reflects stale processing logic. The challenge is not merely sharing code but sharing *maintainable* code that can withstand personnel turnover and data evolution. This motivates the packaging approach described next.

Packaging code for reproducible research

One approach to producing maintainable, shareable code is to organize it as formal software. Wickham [14] and Marwick, Boettiger, and Mullen [15] argue that the R package is a natural unit for reproducible research. The package structure enforces a set of practices that ad-hoc scripts typically lack: function documentation via `roxygen2`, unit tests via `testthat`, dependency management via the `DESCRIPTION` file, and semantic versioning. Marwick et al. [15] extend this idea to "research compendia"—entire research projects organized as R packages. The critical benefit is that `devtools::check()` provides automated verification that all components—code, documentation, tests, dependencies—work together correctly.

Recent case studies demonstrate the practical value of this approach. Botta, Lovelace, Gilbert, and Turrell [16] present `jtstats`, an R and Python package for processing Journey Time Statistics published in *Environment and Planning B*. Their work provides a close parallel: administrative data processing packaged as open-source software with documented methodology. The key difference is that `jtstats` processes *public* transport data, so the privacy dimension of the trilemma does not arise. Landau [5] describes the `targets` R package for pipeline management, relevant both as a tool that `ALprekDB` uses and as a model of software engineering for reproducible research. Donohue et al. [17] describe "Alzverse," a collection of R packages for clinical Alzheimer's research data, demonstrating the packaging approach in another sensitive data domain.

Despite these advances, no published case study demonstrates packaging *confidential administrative data processing* as open-source software. Botta et al. [16] use public data and therefore face no privacy challenge. SIRAD [13] is infrastructure, not a redistributable package. Fischer et al. [2] describe an integrated data system but do not release open-source code. The gap that `ALprekDB` fills is demonstrating how to package processing code for confidential data such that the code is fully transparent, tested, and documented while the data remain completely private. Table 6 in the Results section makes this comparison explicit.

Materials and methods

Context: Alabama First Class Pre-K data

Alabama's First Class Pre-K program, administered by the Alabama Department of Early Childhood Education (ADECE), provides voluntary prekindergarten education to four-year-olds across the state. Established in 2000 and expanded annually, the program served 24,640 children in approximately 1,500 classrooms across all 67 Alabama counties in the 2023–2024 academic year. Services are delivered through seven provider types: public schools, private child care centers, Head Start agencies, community organizations, faith-based organizations, universities, and private schools. Annual state spending totals $181.5 million ($7,368 per child). The program meets all ten NIEER quality benchmarks and has been rated first in the nation for 19 consecutive years [6]. The university research partnership that developed `ALprekDB` processes these administrative data for program evaluation and policy research.

ADECE provides three types of administrative records annually, each in Microsoft Excel format. *Budget data* contain classroom-level expenditures across eight categories (lead teacher salary, lead teacher benefits, auxiliary teacher salary, auxiliary teacher benefits, instructional support, operations and maintenance, equipment, and administration) from two funding sources (Office of School Readiness funds and other sources). *Classroom data* contain site-level information including lead and auxiliary teacher demographics, qualifications, experience, geographic coordinates, delivery type, and grant amounts. *Student data* contain individual-level records including demographics (age, gender, race, language), family characteristics (income, household composition, service receipt), attendance, and assessment scores from two instruments: Teaching Strategies GOLD and the Devereux Early Childhood Assessment (eDECA). The three modules share a common identifier—the classroom code (format: `NNNxNNNNNN.NN`)—which enables cross-module linkage.

Table 1. Data modules and scale. Summary of the three administrative data modules processed by `ALprekDB`, plus the derived linkage module. Raw variable counts reflect the legacy format (2021–2024); the new format (2024–2025) has different column counts. Records per year are approximate and vary by academic year.

Module	Source format	Records/year	Raw variables	Standardized variables	Years
Budget	Excel (.xlsx)	∼1,500	176 / 28	55	2021–2025
Classroom	Excel (.xlsx)	∼1,500	100 / 125	40+	2021–2025
Student	Excel (.xlsx)	∼24,000	202 / 270	80+	2021–2025
Linkage	(derived)	varies	—	208 / 445	2021–2025

Raw variable columns show legacy / new format counts.

These records are subject to multiple privacy protections. Student records contain personally identifiable information—ADECE identification numbers, dates of birth, and detailed demographics—protected by FERPA. A state data use agreement further restricts access to authorized personnel for specified research purposes. Classroom data include teacher names, email addresses, and demographic information. Budget data are less sensitive individually but link to identifiable classroom sites. As a consequence, raw data cannot leave the secure research environment or be shared publicly, but the processing code can. This asymmetry motivates the "public code, private data" design principle that structures the entire `ALprekDB` architecture.

Design principles

`ALprekDB`'s design is guided by five principles, each motivated by a specific failure mode of ad-hoc data processing.

Principle 1: Externalize decisions. When cleaning rules are buried in code, they are opaque to non-programmers and difficult to audit. `ALprekDB` stores all column mappings, category groupings, race and ethnicity classifications, and degree classification patterns in external CSV files—14 files containing 856 total entries. A program administrator can open these files in a spreadsheet application and verify the mappings without reading R code. This serves both transparency (rules are readable) and consistency (the same mappings are applied regardless of who runs the pipeline).

Principle 2: Validate at every stage. Data quality issues discovered late—or never—propagate silently through downstream analyses. `ALprekDB` executes 37 validation checks at each processing stage, returning structured results (check name, description, severity status, issue count, and details) as a queryable tibble. A strict mode option treats warnings as errors, blocking downstream processing until issues are resolved. This serves transparency (quality evidence is logged) and consistency (the same standards apply every time).

Principle 3: Separate code from data. Sharing code together with data violates privacy; not sharing code violates transparency. The `ALprekDB` GitHub repository contains all processing logic, codebooks, unit tests, documentation, and vignettes. Confidential data reside only on secure local machines, excluded from version control via `.gitignore`. This directly resolves the transparency–privacy tension.

Principle 4: Generate synthetic alternatives. Documentation and unit tests cannot demonstrate real data workflows without exposing real data. Three synthetic data generators produce realistic but entirely fabricated records with coordinated random seeds for cross-module linkage. All four package vignettes and all 313 unit tests use synthetic data exclusively. This serves transparency (full public demonstration) and privacy (zero real data exposure).

Principle 5: Automate orchestration. Manual sequencing of processing steps introduces human error and inconsistency. A `targets` pipeline [5] manages the full production workflow with content-hash-based caching, ensuring that only changed inputs trigger reprocessing. This serves consistency (no manual steps) and transparency (the pipeline definition is an inspectable specification of the entire workflow).

System architecture

`ALprekDB` processes three data modules in parallel before linking them into a unified dataset. The architecture consists of three parallel processing lanes—Budget, Classroom, and Student—each following the same general pattern: configuration, reading, cleaning or transformation, validation, and panel construction. The Budget lane includes an additional transformation step that converts individual budget line items into a wide-format master with derived variables. The Student lane includes a transformation step that computes assessment gains, kindergarten readiness flags, chronic absence indicators, and composite risk indices. After panel construction, a Linkage stage joins the three modules: classroom–budget (one-to-one on classroom code and school year), student–classroom (many-to-one), and a master dataset that provides both classroom-level and student-level analytic views. Final output is written to a DuckDB database and exported in five file formats: CSV, Excel, Parquet, RDS, and Stata.

Each processing stage is represented by a distinct S3 class that carries metadata and enforces correct function sequencing. The package defines 17 S3 classes across four modules plus configuration and validation types. The Budget module uses four processing classes (`alprek_budget_raw`, `alprek_budget_long`, `alprek_budget_master`, `alprek_budget_panel`), the Classroom module uses three (`alprek_classroom_raw`, `alprek_classroom_clean`, `alprek_classroom_panel`), the Student module uses three (`alprek_student_raw`, `alprek_student_clean`, `alprek_student_panel`), and the Linkage module uses three (`alprek_linkage_classroom`, `alprek_linkage_student`,

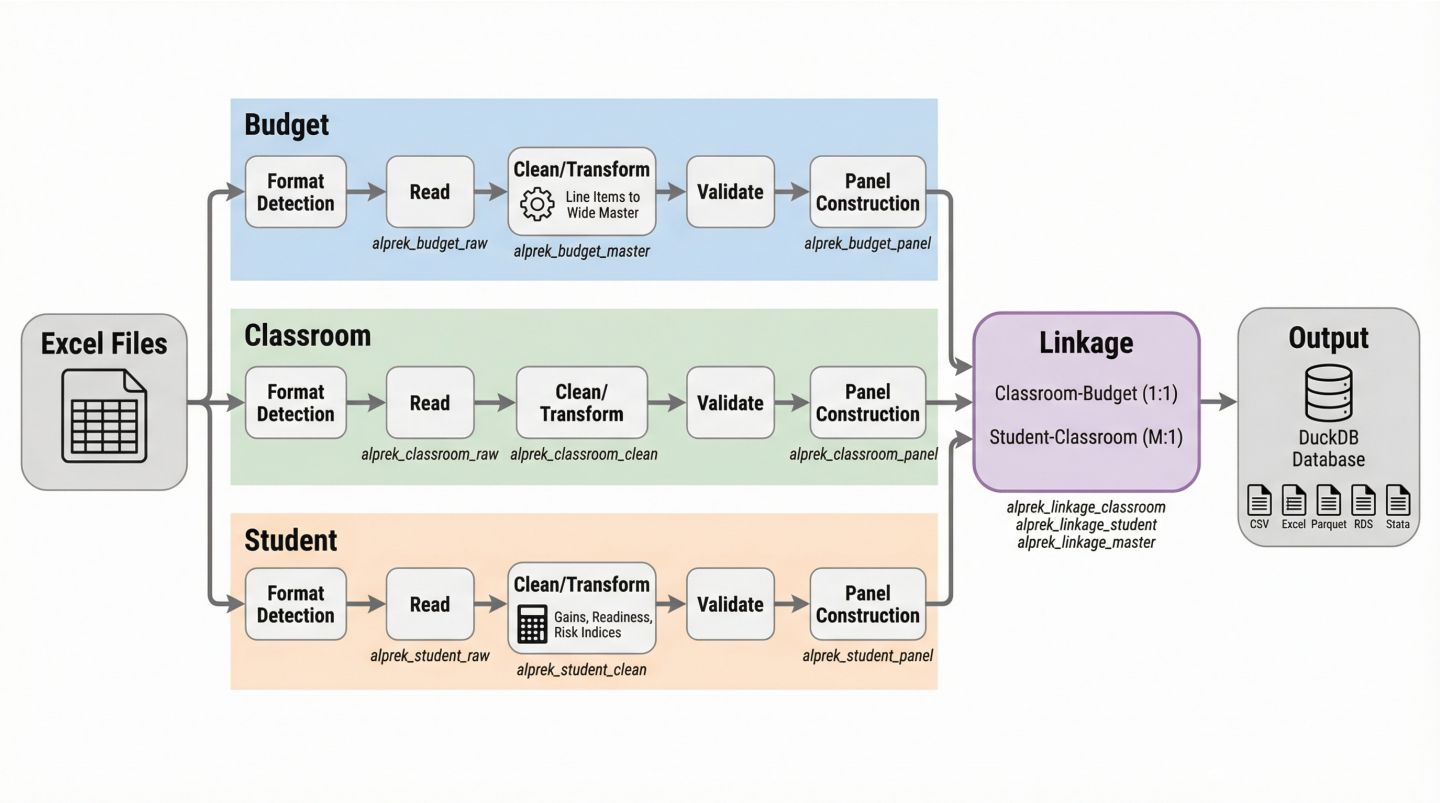

Fig 2. `ALprekDB` system architecture. Three parallel processing lanes (Budget, Classroom, Student) feed into a Linkage stage. Each lane follows the pattern: Excel files are read with automatic format detection, cleaned and standardized using CSV codebook mappings, validated against module-specific checks, and assembled into multi-year panels. The Linkage stage joins the three panels by classroom code and school year. The S3 class hierarchy enforces correct processing order: each function checks its input class and produces a typed output (e.g., `budget_read()` produces an `alprek_budget_raw` object that `budget_clean()` accepts and transforms into an `alprek_budget_long` object). Final output is written to a DuckDB database or exported as CSV, Excel, Parquet, RDS, or Stata files.

`alprek_linkage_master`). Each class has a `print()` method (23 methods total) that displays metadata including record counts, column counts, school year, detected format, and processing timestamp. Functions validate their input types: calling `budget_transform()` on raw data rather than cleaned data produces an informative error. This type system prevents users from accidentally skipping processing stages and serves both consistency (enforced ordering) and transparency (metadata visible at every stage).

Processing parameters are encapsulated in configuration objects (`budget_config()`, `classroom_config()`, `student_config()`) that specify the file path, school year, format (auto-detect or manual override), whether to include personally identifiable information in exports, and whether to run validation in strict mode. Configuration objects are passed through the pipeline and recorded in output metadata, enabling exact reproduction: the same configuration applied to the same input data yields identical output.

Codebook-driven configuration

All data cleaning rules are externalized in 14 CSV files stored in the package's `inst/extdata/` directory. These files fall into two categories. *Column mappings* (six files, 595 total entries) translate raw column names from the source Excel files into standardized variable names, separately for legacy and new formats across all three data modules. For example, the budget column mapping for legacy format maps verbose names such as "Lead Teacher Salary From OSR Funds" to a standardized triplet of variable name (`amount`), category (`lead_teacher_salary`), and source (`osr`). *Codebooks* (eight files, 261 total entries) define category groupings, delivery type codes, county codes, degree classification patterns, and race, ethnicity, and language standardization rules. The degree classification codebook, for instance, contains 27 regular expression patterns with priority ordering for classifying free-text teacher credential descriptions into standardized levels (e.g., the pattern `(?i)master` maps to "Master's" with priority 3).

The codebook approach provides concrete benefits for reproducibility and collaboration. When ADECE changes a column name in their reporting template, the fix is a single-line edit in a CSV file rather than a code change requiring R expertise. Non-programmers can contribute to data quality: an ADECE administrator might notice that a race category mapping is incorrect and edit the CSV directly. In this sense, the codebooks serve as "boundary objects" [18]—artifacts that are meaningful in multiple social worlds, bridging the gap between the world of R programming and the world of program administration. Version control tracks all changes to codebooks alongside code changes, so the complete history of data cleaning decisions is preserved and auditable.

Table 2. Codebook and column mapping inventory. All 14 CSV configuration files stored in `inst/extdata/`. Column mappings translate raw column names to standardized names for each format version. Codebooks define categorical groupings and standardization rules.

File	Category	Entries	Purpose
budget_column_map_legacy.csv	Mapping	7	Budget columns, 2021–2024 format
budget_column_map_new.csv	Mapping	13	Budget columns, 2024–2025 format
classroom_column_map_legacy.csv	Mapping	100	Classroom columns, 2021–2024 format
classroom_column_map_new.csv	Mapping	125	Classroom columns, 2024–2025 format
student_column_map_legacy.csv	Mapping	202	Student columns, 2021–2024 format
student_column_map_new.csv	Mapping	270	Student columns, 2024–2025 format
budget_category_groups.csv	Codebook	38	Budget line items to 8 categories
delivery_type_codes.csv	Codebook	7	Provider type codes (P, C, H, O, F, U, S)
county_codes.csv	Codebook	67	Alabama county codes and FIPS
classroom_degree_patterns.csv	Codebook	27	Regex patterns for degree classification
classroom_race_mapping.csv	Codebook	16	Teacher race/ethnicity standardization
classroom_language_mapping.csv	Codebook	39	Language field normalization
student_race_mapping.csv	Codebook	15	Student race/ethnicity standardization
student_delivery_type_mapping.csv	Codebook	16	Student delivery type cleaning
Total		**856**	

Format detection and migration

In the 2024–2025 academic year, ADECE migrated to a new data management system, fundamentally restructuring all three data files. The budget file changed most dramatically: the legacy format used approximately 176 columns in a wide layout with individual budget line items as separate columns, while the new format uses

approximately 28 columns in a partially aggregated structure. Classroom files changed from roughly 100 to 125 columns with renamed variables and added fields. Student files expanded from approximately 202 to 270 columns, adding consistently populated eDECA post-assessment scores and new data fields. The legacy format also required a special processing step—proportional allocation of payroll taxes to lead and auxiliary teacher benefits—that the new format renders unnecessary because taxes are already included in the benefits columns. Without automatic detection, this format change would have caused a pipeline failure requiring manual diagnosis and code modification.

`ALprekDB` automatically detects the format version by examining column name patterns in each source file. The detection strategy checks for "marker" columns unique to each format: for example, the budget module identifies the legacy format by the presence of any column name ending in "From OSR Funds" and the new format by the joint presence of "OSR" and "Proration" columns. Each module's detection function returns either "legacy" or "new," which selects the appropriate column mapping CSV file. The detection result is recorded in the output metadata and visible through the object's `print()` method. Users can override auto-detection through the configuration object if necessary. This design means that accommodating a future format change requires only two additions: a new column mapping CSV file and a new detection rule. No changes to processing logic are needed. The 2024–2025 format transition validated this architecture: despite the dramatic reduction from 176 to 28 budget columns, the standardized output panel retained its identical 55-column structure.

Validation framework

`ALprekDB` implements a systematic validation framework with 37 checks distributed across all four processing modules: seven for budget data, ten for classroom data, twelve for student data, and eight for linkage operations. Each check returns one of four severity levels: PASS (check satisfied), WARN (potential issue that does not block processing), ERROR (definite problem that blocks processing in strict mode), or INFO (informational diagnostic). Results are stored as a tibble—a structured data frame that can be filtered, aggregated, and exported—with columns for check name, description, status, number of issues identified, and a details field containing affected records.

The checks cover data integrity (required columns present, no duplicate keys), value range plausibility (budget amounts between $0 and $1 million per classroom, student ages between 3 and 7, teacher experience between 0 and 50 years, geographic coordinates within Alabama boundaries), cross-field consistency (budget category sums matching reported totals within a $1.00 tolerance), and coverage adequacy (teacher degree classification rates of at least 95% for lead teachers and 85% for auxiliary teachers, gender distributions within 40–60% as a population-level sanity check). A strict mode option converts all warnings to errors, blocking downstream processing until every issue is resolved. This framework addresses each dimension of the trilemma: transparency (every check and its results are documented and logged), consistency (the same 37 checks execute identically every time), and privacy (validation results describe aggregate patterns such as "3% of students have missing race data," not individual records).

Synthetic data and documentation

`ALprekDB` includes three synthetic data generators—`alprek_synthetic_budget()`, `alprek_synthetic_classroom()`, and `alprek_synthetic_student()`—that produce realistic but entirely fabricated records. Each generator accepts parameters for the number of records, number of years, and a random seed. A critical design feature is seed coordination: the generators share an internal function that produces consistent classroom codes across modules using `withr::with_seed()`, ensuring that synthetic

Table 3. Validation check summary (selected). Twelve representative checks from the 37-check framework. The complete list is available in the package documentation.

Module	Check	Type	Description	Threshold
Budget	required_columns	ERROR	Required columns present	—
Budget	key_uniqueness	ERROR	No duplicate keys	—
Budget	total_reconciliation	ERROR	Category sums match totals	±$1.00
Budget	budget_range	WARN	Values within plausible range	$1,000,000
Classroom	classroom_code_format	ERROR	Code matches expected pattern	regex
Classroom	degree_classification	WARN	Lead ≥95%, Aux ≥85%	95%/85%
Classroom	coordinate_range	WARN	Within Alabama boundaries	30–36°N
Student	student_unique	ERROR	No duplicate ID per year	—
Student	gender_distribution	WARN	Both genders within range	40–60%
Student	missing_rates	WARN	Key variables below threshold	varies
Student	age_range	WARN	Ages within expected range	3–7 years
Linkage	join_rate	WARN	Records successfully linked	≥95%

Twelve of 37 checks shown. Type indicates severity: ERROR blocks processing in strict mode; WARN is advisory.

budget, classroom, and student records can be linked just as real records can. The generated data follow realistic distributions calibrated to observed patterns: budget amounts range from $28,000 to $48,000 for lead teacher salaries, teacher demographics reflect Alabama's workforce composition, and assessment scores follow plausible normal distributions. All output objects carry proper S3 class attributes, making them structurally identical to real pipeline output.

Synthetic data enable full public documentation without any privacy risk. All four package vignettes—covering basic usage, system architecture, panel construction, and cross-module linkage—use synthetic data exclusively. All 313 unit tests operate on synthetic fixtures generated via a shared test helper. The pkgdown documentation website (`https://joonho112.github.io/ALprekDB/`) demonstrates the complete processing pipeline with synthetic examples. Users can install the package and run the entire workflow on synthetic data to verify their setup before connecting real confidential files. This approach simultaneously serves transparency (the full pipeline is publicly demonstrated), consistency (synthetic data use the same processing code as real data), and privacy (no real records appear in any public material).

Pipeline orchestration

The production workflow is orchestrated using the `targets` R package [5], which manages dependencies and caching automatically. The pipeline consists of 30 targets organized in eight stages: source file tracking (12 `tar_file` targets monitoring each Excel input file across three modules and four years), configuration (three targets, one per module), module processing (six targets for processing and panel extraction), student transformation (one target for derived variables), cross-module linkage (one target for the master dataset), validation summary (one target aggregating all check results), DuckDB database writing (one target), and file exports (four targets producing output in multiple formats for each module). Content-hash-based caching ensures that the pipeline re-executes only when inputs actually change: the first complete run takes approximately 90 seconds on a standard laptop, while a cached re-run with no changes completes in under one second.

The `targets` pipeline makes the entire workflow inspectable and reproducible. The function `tar_visnetwork()` generates an interactive dependency graph (Fig 3) showing exactly which targets depend on which inputs, enabling researchers to trace any output back to its source data and processing steps. The function `tar_outdated()` identifies

which targets need updating after data or code changes. The pipeline definition file (`_targets.R`) is itself a concise, readable specification of the full data workflow. Any researcher with legitimate data access can reproduce the entire processing pipeline by running `targets::tar_make()`. The complete pipeline definition, including a YAML-based configuration system that supports both synthetic and real data modes, is publicly available as a companion workflow repository at `https://github.com/joonho112/ALprekDB-workflow`. Users can clone this repository and run the full pipeline on synthetic data with zero configuration; switching to real data requires only editing a configuration file with local data paths.

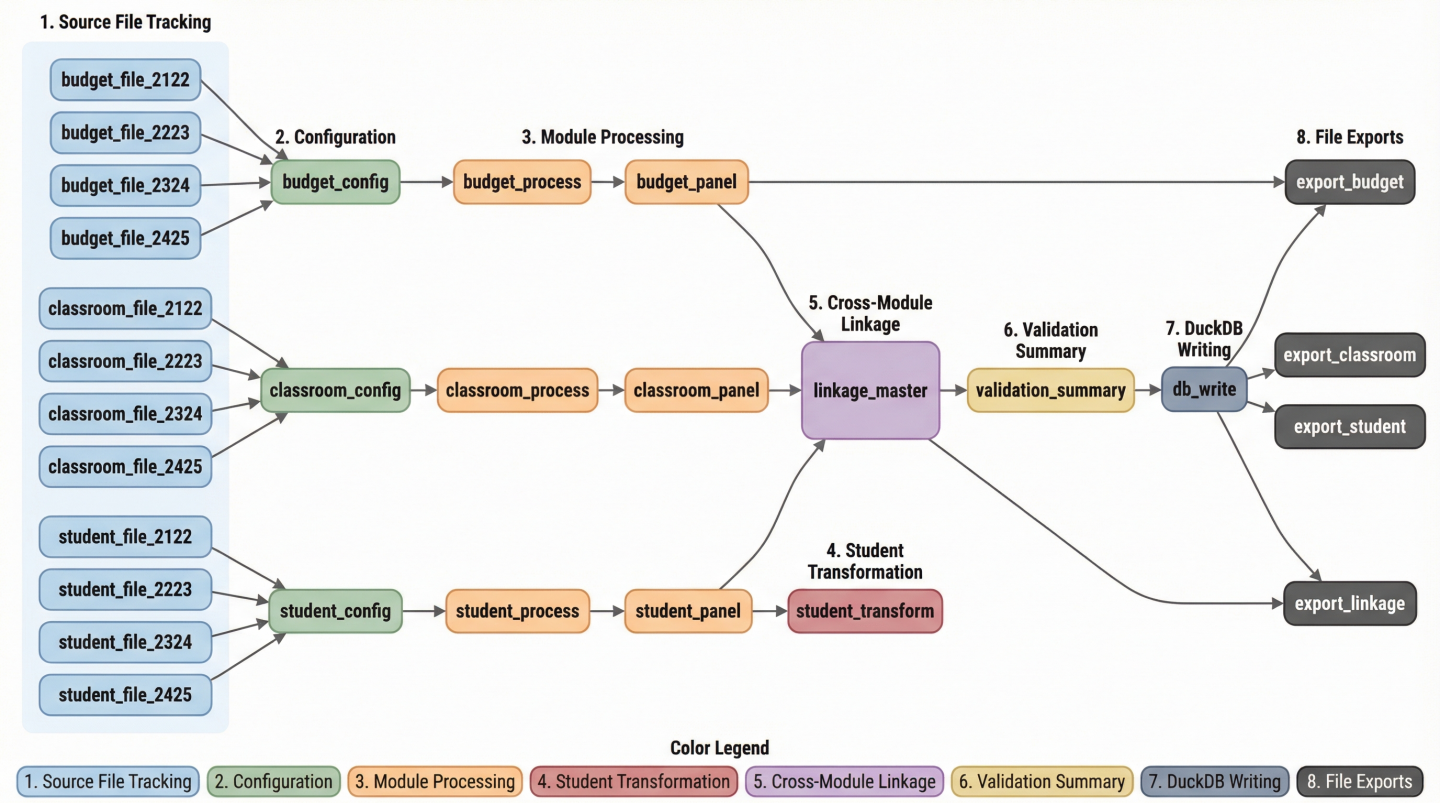

Fig 3. Pipeline dependency graph. Visualization of the `targets` pipeline showing 30 targets organized across eight processing stages. Source file targets (monitoring 12 Excel files) feed into module configuration targets, which flow through parallel processing lanes for Budget, Classroom, and Student data. These converge at the Linkage stage, followed by validation, database writing, and file export. Arrows indicate data dependencies: a target re-executes only when its upstream inputs change, as determined by content-hash comparison. Generated by `tar_visnetwork()`.

Results

Production data summary

We have used `ALprekDB` to process four complete years of Alabama FCPK administrative data, spanning the 2021–2022 through 2024–2025 academic years. The dataset encompasses 5,867 classroom-years and approximately 92,507 student-year records. Classroom counts by year are: 1,376 (2021–2022), 1,483 (2022–2023), 1,524 (2023–2024), and 1,484 (2024–2025). The increase from 2021–2022 to 2023–2024 reflects program expansion, while the modest decrease in 2024–2025 represents normal

year-to-year variation. The student-to-classroom ratio is approximately 16:1, consistent with the program's emphasis on small class sizes. The standardized budget panel contains 55 columns covering eight expenditure categories from two funding sources plus identifiers and derived variables; the classroom panel contains over 40 columns spanning demographics, qualifications, geography, and funding; and the student panel contains over 80 columns covering demographics, family characteristics, service receipt, attendance, assessment scores, and derived outcome measures.

The complete pipeline—from raw Excel files through processing, linkage, database storage, and file export—executes in approximately 90 seconds on a standard laptop computer. This includes reading all 12 source Excel files, applying format detection, executing all cleaning and transformation steps, running 37 validation checks, constructing three multi-year panels, performing cross-module linkage, writing to a DuckDB database, and generating exports in five file formats. When all inputs are unchanged, the `targets` pipeline verifies content hashes and completes in under one second. An incremental update adding one new year of data requires approximately 20 to 30 seconds, as `targets` reprocesses only the new files and downstream linkage operations.

The validation framework provides quantitative evidence of data quality across all four years. Overall, 36 of 37 checks pass consistently across the complete dataset, yielding a 97.3% pass rate. The single recurring issue is a budget reconciliation discrepancy in one classroom (code `139C031601.01`) during the 2023–2024 year: the sum of individual budget categories differs from the reported total by $4,230, exceeding the $1.00 tolerance. This discrepancy was automatically flagged by the `total_reconciliation` check and documented—not silently ignored. Warning-level findings, which are expected and monitored rather than treated as errors, include limited representation of some delivery types (e.g., university-operated classrooms), teacher degree classification coverage of 97% for lead teachers and 88% for auxiliary teachers, and varying rates of missing assessment data across years and instruments.

Table 4. Four-year production summary. Processing results across all four academic years. Budget range reports the median grand total per classroom. Validation counts reflect checks at the ERROR level (blocking) and WARN level (advisory).

Year	Format	Classrooms	Students	Budget (median)	Errors	Warnings
2021–2022	Legacy	1,376	∼22,000	$131K	0	3
2022–2023	Legacy	1,483	∼23,000	$137K	0	3
2023–2024	Legacy	1,524	∼24,000	$140K	1	3
2024–2025	New	1,484	∼23,500	$145K	0	4
Total	—	**5,867**	**∼92,500**	—	**1**	—

Student counts are approximate. The single error is a budget reconciliation discrepancy in one classroom during 2023–2024. Warnings include expected patterns: limited delivery type representation, degree classification coverage below target thresholds, and missing assessment data.

Mapping design features to the trilemma

To evaluate `ALprekDB`'s design systematically, we map each major architectural feature to the three dimensions of the trilemma, assessing whether the feature directly serves transparency, consistency, and/or privacy, and through what specific mechanism. Table 5 presents the complete mapping for ten key design features. This mapping constitutes the core analytical contribution of the paper: rather than evaluating the system against a single criterion, we assess how the combination of features addresses all three competing goals.

Several patterns emerge from this analysis. First, most features serve two of the three goals simultaneously. CSV codebooks, for instance, serve transparency (cleaning rules are readable by non-programmers) and consistency (the same mappings apply regardless of who runs the pipeline), while privacy is supported indirectly through the code–data separation that codebooks enable. Second, privacy is achieved not by restricting code access but by architectural separation: the code is fully public, and the data are completely absent from the public repository. This is more robust than access control mechanisms because there is nothing confidential to leak from the codebase. Third, consistency is achieved through automation: format detection, validation, and pipeline orchestration all eliminate human judgment at runtime, capturing decisions in code and codebooks once and applying them identically thereafter. Fourth, transparency operates at multiple levels—API documentation (pkgdown site), processing logic (R source code), cleaning rules (CSV codebooks), quality evidence (validation logs), and public demonstration (synthetic data with vignettes)—each serving a different audience. Finally, synthetic data emerge as the single feature that most comprehensively bridges all three dimensions: they enable full public documentation (transparency), use the identical processing pipeline as real data (consistency), and contain no real information whatsoever (privacy).

Table 5. Design feature–trilemma mapping. Ten key architectural features mapped to the three dimensions of the trilemma. Checkmarks indicate direct service to a goal; dots indicate indirect or neutral support. This mapping is the core analytical contribution: no single feature resolves all three tensions, but the architectural combination provides comprehensive coverage.

#	Feature	T	C	P	Mechanism
1	CSV codebooks (14 files)	✓	✓	·	Rules readable by non-programmers; same mapping every run
2	Format auto-detection	✓	✓	·	Logic documented in code; handles changes without manual intervention
3	S3 class pipeline (17 classes)	✓	✓	·	Metadata at each stage; type system prevents incorrect sequencing
4	37 validation checks	✓	✓	✓	Logged quality evidence; same standards; aggregates not individuals
5	Synthetic data generators (3)	✓	✓	✓	Full public demo; shared seed; zero real data exposure
6	DuckDB persistence	✓	✓	·	SQL queryable; type reconstruction; access-controlled file
7	targets pipeline (30 targets)	✓	✓	·	Inspectable graph; content-hash caching; public definition
8	313 unit tests	✓	✓	✓	Behavior documented; regression prevention; synthetic data only
9	pkgdown website	✓	·	✓	Full API docs; no confidential data in public materials
10	`include_pii` parameter	✓	✓	✓	Explicit opt-in; consistent handling; clear warnings

T = Transparency, C = Consistency, P = Privacy. ✓ = directly serves this goal; · = indirectly supports or neutral. Key insight: the resolution is *architectural*—the combination of features, not any single feature, addresses all three tensions.

Format migration experience

The 2024–2025 academic year provided a natural test of `ALprekDB`'s format resilience when ADECE migrated to a new data management system. The budget file underwent the most dramatic change, from approximately 176 columns in a wide layout with

individual budget line items as separate columns to approximately 28 columns in a partially aggregated structure. Classroom and student files changed column names extensively and added new data fields. The format auto-detection correctly identified all three new-format files without any manual intervention. The appropriate column mapping CSVs were loaded automatically, and the standardized output remained structurally identical: the budget panel retained its 55-column structure with the same variable names, enabling seamless longitudinal analysis across the format boundary. The only preparation required was creating the new column mapping CSV files—a one-time task completed in approximately two hours. No R code changes were necessary.

The validation framework also revealed a genuine data quality issue through this process. As noted above, one classroom in 2023–2024 exhibited a reconciliation discrepancy of $4,230 between the sum of individual budget categories and the reported grand total. This discrepancy likely represents a data entry error at the source. The package identifies and documents such issues but does not attempt to "fix" them, preserving the original data while providing clear diagnostic information. The fact that this single discrepancy is the only reconciliation mismatch across 5,867 classroom-years speaks both to the overall quality of the source data and to the sensitivity of the validation framework in detecting genuine anomalies.

Comparison with alternative approaches

To contextualize `ALprekDB`'s contribution, we compare it with three alternative approaches to administrative data processing. *Ad-hoc scripts* represent the baseline: individual researchers write cleaning code for each project without formal structure, testing, or documentation. This approach is flexible and requires no upfront investment, but it provides no systematic validation, is not reproducible across researchers, and is entirely personnel-dependent. *SIRAD* [13] represents the infrastructure end of the spectrum: a state-level system for multi-agency data integration with formal security, deidentification, and access management. SIRAD handles challenges that `ALprekDB` does not address (cross-agency record linkage, role-based access control) but requires investment that is impractical for most university–agency partnerships. *jtstats* [16] represents the closest methodological parallel: administrative data processing packaged as open-source R software with documented methodology. However, `jtstats` processes public transport data, so the privacy dimension of the trilemma is absent from their design considerations. `ALprekDB` occupies a distinct position: it provides the reliability and reproducibility of packaged software (like `jtstats`) for confidential data (unlike `jtstats`), at a cost accessible to university–agency partnerships (unlike SIRAD). Table 6 summarizes this comparison.

Table 6. Comparison with alternative approaches. Four approaches to administrative data processing compared across key dimensions relevant to the trilemma framework.

Dimension	Ad-hoc scripts	SIRAD	jtstats	`ALprekDB`
Transparency	Low	Medium	High	High
Consistency	Low	High	High	High
Privacy handling	Variable	High (formal)	N/A (public data)	High (architectural)
Scope	Single project	Multi-agency	Single source	Single program
Investment	Minimal	$200K+/year	Moderate	Moderate
Data type	Any	Integrated admin	Public transport	Confidential education
Validation	None standard	System-level	Basic	37 checks, 4 levels
Open source	Rarely	No	Yes	Yes

SIRAD = Secure Infrastructure for Research with Administrative Data [13]. jtstats = Journey Time Statistics R package [16].

Discussion

Resolving the trilemma

The central finding of this case study is that the transparency–consistency–privacy trilemma is resolvable through architectural design, not merely through policy or intent. The underlying principle—share code publicly while keeping data private [4, 12]—is straightforward, but effective implementation requires supporting mechanisms along each dimension. Transparency demands not just open code but *readable* code (CSV codebooks), *tested* code (313 unit tests), and *demonstrated* code (synthetic data powering four public vignettes). Consistency demands not just shared code but *enforced* processing sequences (S3 type system), *automated* validation (37 checks), and *reproducible* orchestration (content-hash caching). Privacy demands not just data exclusion but *explicit* opt-in decisions (`include_pii` parameter), *synthetic* alternatives for all public content, and *aggregate* reporting in validation outputs. As Table 5 shows, no single feature addresses all three tensions; the resolution is architectural.

CSV codebooks serve as "boundary objects" [18] that facilitate communication between researchers and program administrators. Star and Griesemer define boundary objects as artifacts that maintain a common identity across different social worlds while remaining plastic enough to adapt to local needs and constraints. A codebook file such as `budget_category_groups.csv` can be opened in Microsoft Excel by an ADECE staff member who verifies that budget line items are correctly categorized, and simultaneously loaded in R by a data analyst who uses it to drive automated processing. This dual readability bridges the gap between technical documentation and operational documentation. When the 2024–2025 format change occurred, the codebook approach allowed us to accommodate the new data structure by creating new mapping files without modifying any R code—a practical demonstration of the boundary object's flexibility across contexts.

The validation framework demonstrates how transparency can be achieved at the aggregate level without exposing individual records. Results such as "97% of lead teachers have classified degrees" or "median classroom budget is $140,000" provide meaningful evidence of data quality that can be shared publicly—including in this paper—even though the underlying individual records cannot be disclosed. This is a practical implementation of an intuition from the differential privacy literature: sharing carefully chosen statistics about data, rather than the data themselves, can satisfy transparency requirements while preserving individual confidentiality.

Practical implications

For state agencies like ADECE, packaged data processing infrastructure offers three practical benefits. First, *lower maintenance burden*: when a researcher or graduate student leaves the project, the package remains fully functional. The next person can install it, read the vignettes and API documentation, and begin processing data immediately without reverse-engineering predecessor scripts. Second, *personnel-independent quality*: the 37 validation checks execute regardless of who runs the pipeline, providing consistent data quality monitoring that is embedded in the tool rather than dependent on individual expertise. Third, *auditable processing*: if a legislator, funder, or journalist questions how program data were cleaned, the answer is in publicly accessible, version-controlled code and codebooks—not in an email chain or personal notebook. These benefits are especially valuable for programs with high staff turnover or where the research partnership involves rotating graduate students and postdoctoral researchers [19].

For university–agency research partnerships, shared infrastructure creates a common

technical platform that reduces coordination costs. Multiple researchers can work on the same program data using identical processing, eliminating the discrepancies that arise when different team members apply different cleaning conventions. A second researcher can independently verify data processing by running the same package on the same raw files and comparing output. New research questions can build on the standardized panel data without re-cleaning from raw files. And the package structure—with its functions, documentation, and tests—provides a natural framework for dividing labor and maintaining quality as teams change over time.

For funders and policymakers, packaged data infrastructure provides documented, auditable evidence of data quality that can accompany research findings and program evaluations. The Data Quality Campaign [8] reports that early childhood administrators want better technology for data management; `ALprekDB` provides a concrete model for how open-source software can serve this need. Federal and state funders who require evidence of data quality in evaluation reports can point to the publicly available validation framework and processing documentation as indicators of methodological rigor.

Limitations

Several limitations of this work should be acknowledged. First, `ALprekDB` is designed for a single program's administrative data, not for multi-agency integration. It does not address cross-agency record linkage, probabilistic matching, or deidentification—tasks for which tools like SIRAD [13] are more appropriate. However, `ALprekDB`-style processing could serve as a component within a larger integrated data system, providing clean, validated input to a cross-agency linkage process.

Second, the package requires R and its ecosystem, which may limit adoption by agencies without existing R expertise. The DuckDB export partially mitigates this constraint: processed data stored in DuckDB can be queried from Python, SQL, or other environments. CSV and Stata exports provide non-R access to final datasets, and the codebook CSV files are entirely platform-independent. A Python wrapper or web-based interface could further broaden accessibility in future work.

Third, despite substantial automation, the package requires ongoing technical maintenance. New format versions demand new column mapping CSV files and possibly new detection rules. R package dependencies (nine required, ten suggested) may require updates as the R ecosystem evolves. Validation thresholds may need adjustment as program characteristics change. This maintenance still requires technical skill, though the package structure makes it more systematic and less error-prone than maintaining ad-hoc scripts.

Fourth, `ALprekDB`'s privacy protection is organizational, not cryptographic. The system relies on `.gitignore` rules and data use agreement compliance rather than encryption, differential privacy, or secure multi-party computation. A careless or malicious user with legitimate data access could still mishandle confidential records. The package provides guardrails—the `include_pii` parameter, export warnings, synthetic data defaults—but cannot enforce data governance. For applications requiring stronger privacy guarantees, institutional safeguards such as secure computing environments and access logging remain necessary.

Fifth, generalizability is limited by the scope of testing. `ALprekDB` has been developed and evaluated using data from one program in one state. While the design principles (externalized codebooks, systematic validation, synthetic data, pipeline orchestration) are general, the specific implementation is tailored to Alabama FCPK data structures. Adapting the package to another state's prekindergarten program would require new codebooks, new column mappings, and potentially new validation

checks. We hope that the design patterns described here, rather than the specific code, represent the most transferable contribution.

Conclusion

We have presented `ALprekDB`, an open-source R package that standardizes the processing of confidential administrative records from Alabama's First Class Pre-K program through a "public code, private data" architecture. Five design principles—externalized codebooks, stage-level validation, code–data separation, synthetic data alternatives, and automated orchestration—together resolve the transparency–consistency–privacy trilemma that characterizes confidential administrative data processing. Evaluation on four years of production data (5,867 classroom-years; 92,507 student-year records) confirmed that the architecture handles major format migrations without code changes and that a 37-check validation framework maintains consistent data quality across years.

The trilemma we describe is not unique to Alabama or to prekindergarten programs. Health services, social welfare programs, criminal justice agencies, and workforce development initiatives face analogous tensions between methodological openness, processing reliability, and individual privacy. The architectural patterns we present—codebooks as boundary objects, validation as aggregate transparency, and synthetic data as a documentation bridge—apply broadly. `ALprekDB` is freely available under the MIT license, and we welcome community contributions and adaptations for other programs. The package source code, production pipeline, and analysis report templates are all publicly available (see Data Availability).

Data availability

The `ALprekDB` R package source code is publicly available at `https://github.com/joonho112/ALprekDB` under the MIT license. Full API documentation is available at `https://joonho112.github.io/ALprekDB/`. The production `targets` pipeline and Quarto analysis report are available in a companion repository at `https://github.com/joonho112/ALprekDB-workflow`, which can be run on synthetic data without access to confidential records. The administrative data analyzed in this study contain personally identifiable information protected by the Family Educational Rights and Privacy Act (FERPA) and a state data use agreement, and cannot be shared publicly. The package includes synthetic data generators that reproduce the full pipeline without confidential records.

Acknowledgments

The authors thank the Alabama Department of Early Childhood Education (ADECE) for providing administrative data and for their ongoing partnership in program evaluation.

References

1. Figlio D, Karbownik K, Salvanes KG. Education Research and Administrative Data. In: Hanushek EA, Machin S, Woessmann L, editors. Handbook of the Economics of Education. vol. 5. Amsterdam: Elsevier; 2016. p. 75-138. doi:10.1016/B978-0-444-63459-7.00002-6.

2. Fischer RL, Richter FGC, Anthony E, Lalich N, Coulton C. Leveraging Administrative Data to Better Serve Children and Families. Public Administration Review. 2019;79(5):675-83. doi:10.1111/puar.13047.

3. Graham FS, Gooden ST, Martin KJ. Navigating the Transparency–Privacy Paradox in Public Sector Data Sharing. The American Review of Public Administration. 2016;46(5):569-91. doi:10.1177/0275074014561116.

4. Vilhuber L. Reproducibility and Transparency Versus Privacy and Confidentiality: Reflections from a Data Editor. Journal of Econometrics. 2023;235(2):2285-94. doi:10.1016/j.jeconom.2023.05.001.

5. Landau WM. The targets R Package: A Dynamic Make-Like Function-Oriented Pipeline Toolkit for Reproducibility and High-Performance Computing. Journal of Open Source Software. 2021;6(57):2959. doi:10.21105/joss.02959.

6. National Institute for Early Education Research. The State of Preschool 2024: Alabama State Profile [Annual report]. National Institute for Early Education Research, Rutgers University; 2025 [cited 2026-01-15]. Available from: https://nieer.org/yearbook/2024/state-profiles/alabama.

7. Card D, Chetty R, Feldstein MS, Saez E. Expanding Access to Administrative Data for Research in the United States [White paper]. American Economic Association; 2010 [cited 2026-01-15]. Available from: https://eml.berkeley.edu/~saez/card-chetty-feldstein-saezNSF10dataaccess.pdf.

8. Data Quality Campaign. Early Childhood Administrators Value Data for Decisionmaking [Survey report]. Data Quality Campaign; 2024 [cited 2026-01-15]. Available from: https://files.eric.ed.gov/fulltext/ED674738.pdf.

9. Spears JV, Bradburn I, Schroeder AD, Tester DM, Forry N. New Data on Child Care Subsidy Programs. Policy & Practice. 2012 Aug:18-21.

10. Prado Y, Wei X. Privacy-Preserving Research Models Essential for Large Scale Education R&D Infrastructure [Policy memo]. Federation of American Scientists; 2025 [cited 2026-01-15]. Available from: https://fas.org/publication/privacy-preserving-research-models-ed-rd/.

11. Cole S, Dhaliwal I, Sautmann A, Vilhuber L, editors. Handbook on Using Administrative Data for Research and Evidence-Based Policy. Cambridge, MA: Abdul Latif Jameel Poverty Action Lab; 2020. Available from: https://admindatahandbook.mit.edu/.

12. Playford CJ, Gayle V, Connelly R, Gray AJG. Administrative Social Science Data: The Challenge of Reproducible Research. Big Data & Society. 2016;3(2). doi:10.1177/2053951716684143.

13. Howison M, Goggins M. SIRAD: Secure Infrastructure for Research with Administrative Data. Software Impacts. 2022;12:100245. doi:10.1016/j.simpa.2022.100245.

14. Wickham H. R Packages: Organize, Test, Document, and Share Your Code. 1st ed. Sebastopol, CA: O'Reilly Media; 2015.

15. Marwick B, Boettiger C, Mullen L. Packaging Data Analytical Work Reproducibly Using R (and Friends). The American Statistician. 2018;72(1):80-8. doi:10.1080/00031305.2017.1375986.

16. Botta F, Lovelace R, Gilbert L, Turrell A. Packaging Code and Data for Reproducible Research: A Case Study of Journey Time Statistics. Environment and Planning B: Urban Analytics and City Science. 2025;52(4):1002-13. doi:10.1177/23998083241267331.

17. Donohue MC, Hussen K, Langford O, Gallardo R, Jimenez-Maggiora G, Aisen PS. Alzheimer's Clinical Research Data via R Packages: The Alzverse. Alzheimer's & Dementia. 2026;22(2):e71152. doi:10.1002/alz.71152.

18. Star SL, Griesemer JR. Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907–39. Social Studies of Science. 1989;19(3):387-420. doi:10.1177/030631289019003001.

19. Meyer JL, Waterman C, Coleman GA, Strambler MJ. Whose Agenda Is It? Navigating the Politics of Setting the Research Agenda in Education Research-Practice Partnerships. Educational Policy. 2023;37(1):122-46. doi:10.1177/08959048221131567.