Draft version April 15, 2025
Typeset using LaTeX **twocolumn** style in AASTeX7

The LSST Science Pipelines Software: Optical Survey Pipeline Reduction and Analysis Environment

Rubin Observatory Science Pipelines Developers, James F. Bosch[2] Yusra AlSayyad[2] Tim Jenness[3]
Eric C. Bellm[4] Robert H. Lupton[2] Nate B. Lust[2] Ian S. Sullivan[4] Christopher Z. Waters[2]
Krzysztof Findeisen[4] Erfan Nourbakhsh[2] Agnès F. Ferté[5] Arun Kannawadi[6,2] Eli S. Rykoff[7]
Andrés A. Plazas Malagón[5,7] K. Simon Krughoff[3,*] John K. Parejko,[4] Lee S. Kelvin[2]
And Clare Saunders[2]

The Rubin Observatory Science Pipelines Team

[1]

[2]Department of Astrophysical Sciences, Princeton University, Princeton, NJ 08544, USA
[3]Vera C. Rubin Observatory Project Office, 950 N. Cherry Ave., Tucson, AZ 85719, USA
[4]University of Washington, Dept. of Astronomy, Box 351580, Seattle, WA 98195, USA
[5]SLAC National Accelerator Laboratory, 2575 Sand Hill Rd., Menlo Park, CA 94025, USA
[6]Department of Physics, Duke University, Durham, NC 27708, USA
[7]Kavli Institute for Particle Astrophysics and Cosmology, SLAC National Accelerator Laboratory, 2575 Sand Hill Rd., Menlo Park, CA 94025, USA

ABSTRACT

The NSF-DOE Vera C. Rubin Observatory will produce the Legacy Survey of Space and Time (LSST), producing 11 data releases over the ten-year survey. The LSST Science Pipelines Software, the Optical Survey Pipeline Reduction and Analysis Environment (OSPRAE), will be used to create these data releases and to perform the nightly alert production. This paper provides an overview of the LSST Science Pipelines Software, describing the components and how they are combined to form pipelines.

Keywords: Astrophysics - Instrumentation and Methods for Astrophysics — methods: data analysis — methods: miscellaneous

1. INTRODUCTION

The NSF-DOE Vera C. Rubin Observatory will be performing the 10-year Legacy Survey of Space and Time (LSST; Ž. Ivezić et al. 2019) starting in 2025. Rubin Observatory is located on Cerro Pachon in Chile and consists of the 8.4 m Simonyi Survey Telescope (S. J. Thomas et al. 2022) with the 3.2-gigapixel LSSTCam survey camera (A. Roodman et al. 2024) performing the main survey and the Rubin Auxiliary Telescope (P. Ingraham et al. 2020) providing supplementary atmospheric calibration data. The Data Management System (DMS; W. O'Mullane et al. 2022) is designed to handle the flow of data from the telescope, approaching 20 TB per night, in order to issue alerts and to prepare annual data releases. A central component of the DMS is the LSST Science Pipelines software that provides the algorithms and frameworks required to process the data from the LSST and generate the coadds, difference images, and catalogs to the user community for scientific analysis.

The LSST Science Pipelines software consists of the building blocks and pipeline infrastructure required to construct high performance pipelines to process the data from LSST. It has been under development since at least 2004 (T. Axelrod et al. 2004) and has evolved significantly over the years as the project transitioned from prototyping (T. Axelrod et al. 2010) and entered into formal construction (M. Jurić et al. 2017). The software is designed to be usable by other optical telescopes and this has been demonstrated with Hyper Suprime Cam on the Subaru Telescope in Hawaii (J. Bosch et al. 2018) and also with data from the Dark Energy Camera (DECam), the VISTA infrared camera (VIRCAM), the Wide Field Survey Telescope (WFST; M. Cai et al.

* Author is deceased

2025), and the Gravitational-wave Optical Transient Observer (GOTO; J. R. Mullaney et al. 2021).

In this paper we provide an overview of the components of the software system. This includes a description of the support libraries and data access abstraction, the pipeline task system, and an overview of the algorithmic components. We do not include details of the science validation of the individual algorithms. The other components of the LSST DMS, such as the workflow system (M. Gower et al. 2022; E. Karavakis et al. 2024), the Qserv database (D. L. Wang et al. 2011; F. Mueller et al. 2023) and the Rubin Science Platform (M. Jurić et al. 2019; W. O'Mullane et al. 2024), are not covered in this paper.

2. FUNDAMENTALS

The LSST Science Pipelines software is written in Python with C++ used for high-performance algorithms and for core classes that are usable in both languages. We use Python 3 (having ported from python 2, T. Jenness 2020, currently with a minimum version of Python 3.12), and the C++ layer can use C++17 features with pybind11 being used to provide the interface from Python to C++. Additionally, the C++ layer uses `ndarray` to allow seamless passing of C++ arrays to and from Python `numpy` arrays. This compatibility with `numpy` is important in that it makes LSST data structures available to standard Python libraries such as Scipy and Astropy (T. Jenness et al. 2016; Astropy Collaboration et al. 2018).

Although all the software uses the `lsst` namespace, the code base is split into individual Python products in the LSST GitHub organization[8] that can be installed independently and which declare their own dependencies. These dependencies are managed using the "Extended Unix Product System" (EUPS; N. Padmanabhan et al. 2015; T. Jenness et al. 2018) where most of the products are built using the SCons system (S. Knight 2005) with LSST-specific extensions provided in the `sconsUtils` package enforcing standard build rules and creating the necessary Python package metadata files.

For logging we always use standard Python logging with an additional `VERBOSE` log level between `INFO` and `DEBUG` to provide additional non-debugging detail that can be enabled during batch processing. This verbose logging is used for periodic logging where long-lived analysis tasks are required to issue a log message every 10 minutes to indicate to the batch system that they are still alive and actively performing work. For logging from C++ we use Log4CXX wrapped in the `lsst.log` package to make it look more like standard Python logging, whilst also supporting deferred string formatting such that log messages are only formed if the log message level is sufficient for the message to be logged. These C++ log messages are forwarded to Python rather than being issued from an independent logging stream. Finally, we also provide some LSST-specific exceptions that can be thrown from C++ code and caught in Python.

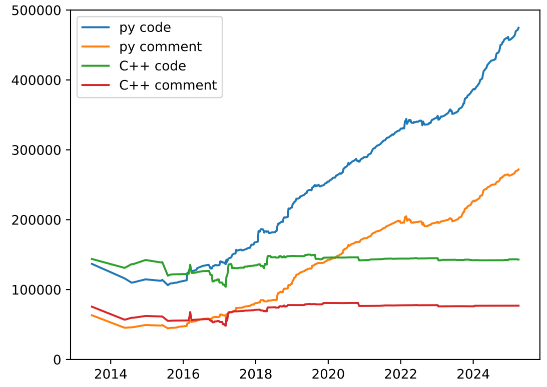

Figure 1. The number of lines of code comprising the LSST Science Pipelines software as a function of year. Line counts include comments but not blank lines. Python interfaces are implemented using `pybind11` and that is counted as C++ code. For the purposes of this count Science pipelines software is defined as the `lsst_distrib` metapackage and does not include code from third party packages.

As of April 2025, the Science Pipelines software is approximately 700,000 lines of Python and 225,000 lines of C++. The number of lines in the pipelines code as a function of time is given in Fig. 1.

2.1. *Python environment*

An important aspect of running a large data processing campaign is to ensure that the software environment is well defined. We define a base python environment using conda-forge via a meta package named `rubin-env`[9]. This specifies all the software needed to build and run the science pipelines software. A Docker container is built for each software release and the fully-specified versions of all software are recorded to ensure repeatability.

2.2. *Unit Testing and Code Coverage*

Unit testing and code coverage are critical components of code quality (T. Jenness et al. 2018). Every package comes with unit tests written using the standard `unittest` module. We run the tests using `pytest` (H. Krekel 2017) and this comes with many advantages in

[8] https://github.com/lsst

[9] https://github.com/conda-forge/rubinenv-feedstock

Table 1. Common dimensions present in the default dimension universe.

Name	Description
`instrument`	Instrument.
`band`	Waveband of interest.
`physical_filter`	Filter used for the exposure.
`day_obs`	The observing day.
`group`	Group identifier.
`exposure`	Individual exposure.
`visit`	Collection of 1 or 2 exposures.
`tract`	Tesselation of the sky.
`patch`	Patch within a tract.

that all the tests run in the same process and requiring global parameters to be well understood, tests can be run in parallel in multiple processes, plugins can be enabled to extend testing and record test coverage, and a test report can be created giving details of run times and test failures. Coding standards compliance with PEP 8 (G. van Rossum 2013) is enforced using GitHub Actions, the `ruff` package, and `pre-commit` checks. A Jenkins system provides the team with continuous integration facilities that includes running longer tests with pre-cursor datasets.

3. DATA ACCESS ABSTRACTION

3.1. *Butler*

Early in the development of the LSST Science Pipelines software it was decided that the algorithmic code should be written without knowing where files came from, what format they were written in, where the outputs are going to be written or how they are going to be stored. All that the algorithmic code needs to know is the relevant data model and the Python type. To meet these requirements we developed a library called the Data Butler (see e.g., T. Jenness et al. 2022; N. B. Lust et al. 2023).

The Butler internally is implemented as a registry, a database keeping track of datasets, and a datastore, a storage system that can map a Butler dataset to a specific collection of bytes. A datastore is usually a file store (including POSIX file system, S3 object stores, or WebDAV) but it is also possible to store metrics directly into the Sasquatch metrics service (A. Fausti 2023; A. Fausti Neto et al. 2024).

A core concept of the Butler is that every dataset must be given what we call a "data coordinate." The data coordinate locates the dataset in the dimensional space where dimensions are defined in terms that scientists understand. Some commonly used dimensions are listed in Table 1. Each dataset is uniquely located by specifying its dataset type, its run collection, and its coordinates, with Butler refusing to accept another dataset that matches all three of those values. The dataset type defines the relevant dimensions (such as whether this is referring to observations or a sky map) and the associated Python type representing the dataset. The run collection can be thought of as a folder grouping datasets created by the same batch operation, but does not have to be a folder within a file system.

As a concrete example, the file from one detector of an LSSTCam observation taken sometime in 2025 could have a data coordinate of `instrument="LSSTCam", detector=42, exposure=2025080300100` and be associated with a `raw` dataset type. The `exposure` record itself implies other information such as the physical filter and the time of observation. A deep coadd on a patch of sky would not have `exposure` dimensions at all and would instead be something like `instrument="LSSTCam", tract=105, patch=2, band="r", skymap="something"`, which would tell you exactly where it is located in the sky and in what waveband since you can calculate it from the tract, patch, band and skymap.

3.2. *Instrument Abstractions: Obs Packages*

The Butler and pipeline construction code know nothing about the specifics of a particular instrument. In the default dimension universe there is an `instrument` dimension that includes a field containing the full name of a Python `Instrument` class. This class, which uses a standard interface, is used by the system to isolate the instrument-specific from the pipeline-generic. Some of the responsibilities are:

- Register instrument-specific dimensions such as `detector`, `physical_filter` and the default `visit_system`.
- Define the default `raw` dataset type and the associated dimensions.
- Provide configuration defaults for pipeline task code that is processing data from this instrument.
- Provide a "formatter" class that knows how to read raw data.
- Define the default curated calibrations known to this instrument.

By convention we define the instrument class and associated configuration in `obs` packages. As an extension to the base definition of an "instrument", the LSST Science Pipelines define a modified `Instrument` class that includes focal plane distortions using the `afw` package (see §4.3). There are currently project-supported `obs` packages for:

- LSSTCam (A. Roodman et al. 2024; T. Lange et al. 2024; Y. Utsumi et al. 2024; S. M. Kahn et al. 2010), LATISS (P. Ingraham et al. 2020), and associated Rubin Observatory test stands and simulators.
- Hyper-SuprimeCam (S. Miyazaki et al. 2018).
- The Dark Energy Camera (B. Flaugher et al. 2015; D. L. DePoy et al. 2008).
- CFHT's MegaPrime (O. Boulade et al. 2003).

Additionally, teams outside the project have developed `obs` packages to support Subaru's Prime Focus Spectrograph (S.-Y. Wang et al. 2020), VISTA's VIRCAM (W. Sutherland et al. 2015), the Wide Field Survey Telescope (WFST; M. Cai et al. 2025), and the Gravitational-wave Optical Transient Observer (GOTO; J. R. Mullaney et al. 2021).

3.3. *Metadata Translation*

Every instrument uses different metadata standards but the Butler data model and pipelines require some form of standardization to determine values such as the coordinates of an observation, the observation type, or the time of observation. To perform that standard extraction of metadata each supported instrument must provide a metadata translator class using the `astro_metadata_translator` infrastructure.[10] The translator classes can understand evolving data models and allow the standardized metadata to be extracted for the lifetime of an instrument even if headers changed. Furthermore, in addition to providing standardized metadata the package can also provide programmatic or per-exposure corrections to data headers prior to calculating the translated metadata. This allows files that were written with incorrect headers to be recovered during file ingestion.

4. CORE INFRASTRUCTURE LIBRARIES

4.1. *Region Handling*

The `sphgeom` package is used for spherical geometry calculations, sky-based region defintions, and sky pixelization schemes. The `geom` package is used to locations and extents within a Cartesian coordinate space.

(Aside: I only just realized that lsst.geom has SpherePoint that is effectively LonLat from sphgeom and we have both lsst.geom.Angle and lsst.sphgeom.Angle... I feel like if I document this, that questions will be asked...)

For coordinates, we use ICRS everywhere and leave any required coordinate transformations to the Astropy infrastructure.

4.2. *Time and Hierarchical Data Structures*

The `daf_base` package provides core data structures for handling time and hierarchical data structures. The `DateTime` package is used in our C++ data models mostly to represent TAI times. For general manipulations of times in Python we now use `astropy.time`, following the recommendations from T. Jenness et al. (2016).

The `PropertySet` and `PropertyList` classes tallow `dict`-like data structures to be passed from Python to C++ and back again. The `PropertySet` represents a hierarchical key/value data structure whereas `PropertyList` is a flat data structure that is used to represent a FITS header and supports multi-valued keys and key comments.

4.3. *Application Framework*

`afw` – this is called the "Application Framework" in T. Axelrod et al. (2010)[11]

- Image/MaskedImage/Exposure
- Table and Catalogs.
- Detection
- Math
- Camera geometry
- FITS I/O
- WCS: AST library (D. S. Berry et al. 2016) backs the world coordinate system handling.

4.4. *Co-add Utilities*

`coadd_utils` ?

[10] https://astro-metadata-translator.lsst.io

[11] This document can be downloaded from https://ls.st/Document-9349

5. INSTRUMENT SIGNATURE REMOVAL

Raw images from charge-coupled devices (CCDs) contain instrumental effects, such as dark currents, clocking artifacts, or crosstalk between neighboring amplifiers, that can be removed in the data processing. In the Rubin pipeline, this step is called Instrument Signature Removal (ISR) and is the first processing applied to a raw CCD exposure. The package performing the ISR on an exposure, called `ip_isr`, is detailed below in Sec. 5.1: it is a critical package for Data Release Pipeline (DRP) used to process LSST images and requires calibration products produced and verified by `cp_pipe` and `cp_verify` respectively as described in Sec. 11.5.1. For further information about the life cycle of a calibration product and the procedures it entails, see C. Waters (2025). In Sec. 5.2, we specifically describe the correction of amplifier offset in more detail. A general overview of the ISR steps (based on the model in Fig. 2) and calibration products production (including generation, verification, certification, approval, and distribution) is given in A. A. Plazas Malagón et al. (2024).

We note that we focus here on our approach to performing ISR on data from LSST cameras only (LSSTCam, ComCam, and LATISS), although we also provide calibration pipelines for other cameras such as DECam and HSC (using a different ISR approach).

5.1. *ISR package*

Exposures from LSST cameras are affected by instrumental effects, ranging from well-known CCD effects like dark currents or bias levels to effects more recently characterized like tree-rings (see H. Y. Park et al. (2017); H. Park et al. (2020); J. H. Esteves et al. (2023); Y. Okura et al. (2015, 2016) for more details on tree rings in LSSTCam and their impact on science) or the Brighter-Fatter effect as discussed in A. Broughton et al. (2024). Correcting for these effects requires specific calibrations, which we refer to as calibration products. In LSST cameras, calibration products typically are a combined bias, a combined dark, a Photon Transfer Curve (PTC), a crosstalk matrix, a list of defects, and a look-up table of non-linearity parameters. The meaning of these calibration products and the details on the Rubin Observatory's ISR and calibration approach can be found in A. A. Plazas Malagón et al. (2024) and (P. Fagrelius & E. Rykoff 2025).

The `ip_isr` package[12] contains the codes needed to remove instrument signatures in exposures from LSST cameras and to produce calibration products. To in-

[12] https://github.com/lsst/ip_isr

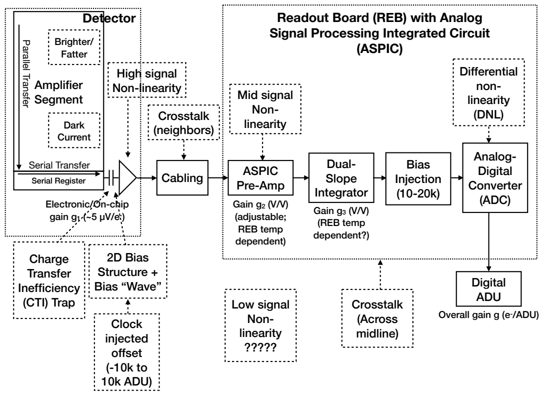
Detector
Parallel Transfer
Brighter/ Fatter
Amplifier Segment
Dark Current
Serial Transfer
Serial Register
Electronic/On-chip gain g1 (~5 μV/e-)
High signal Non-linearity
Crosstalk (neighbors)
Cabling
Charge Transfer Inefficiency (CTI) Trap
2D Bias Structure + Bias "Wave"
Clock injected offset (-10k to 10k ADU)
Readout Board (REB) with Analog Signal Processing Integrated Circuit (ASPIC)
Mid signal Non-linearity
ASPIC Pre-Amp
Gain g2 (V/V) (adjustable; REB temp dependent)
Dual-Slope Integrator
Gain g3 (V/V) (REB temp dependent?)
Bias Injection (10-20k)
Differential non-linearity (DNL)
Analog-Digital Converter (ADC)
Digital ADU
Overall gain g (e-/ADU)
Low signal Non-linearity ?????
Crosstalk (Across midline)

Figure 2. Schematic of the instrument model for detector effects in LSST cameras which `isrTaskLSST` is based on at the time of publication. More details about the model can be found in P. Fagrelius & E. Rykoff (2025) and A. A. Plazas Malagón et al. (2024).

form our ISR approach, we first designed a model of the instrument, displayed in Fig. 2, based on our knowledge of the hardware and electronics. This model states the order in which the different known instrumental effects happen, from a photon hitting the CCD to the output ADC unit (ADU) signal. In turn, `isrTaskLSST` in `ip_isr` sequentially applies corrections of these effects in the opposite order as their effects occur in the model, as we are attempting to remove the impact of those effects on the image. Such corrections are typically done by calling other `Tasks` (*e.g.* overscan, crosstalk, etc.) also implemented in `ip_isr`.

Overall, `isrTaskLSST` takes a `raw` CCD exposure, and calibration products if available, and outputs a `Struct` containing the output exposure, the `postISRCCD` output exposure as well as its binned version for easier display, the exposure without interpolation and statistics on the output exposure. `IsrTaskLSSTConfig` defines the configurations used in this `Task`, they are set by default to their expected value to perform ISR on a typical LSSTCam exposure. Configuration parameters starting with `do` will typically correspond to an ISR step, they are turned on or off in the pipelines when producing the different calibration products. We have also developed `isrMockLSST` which simulates a raw exposure and corresponding calibration products and is used to test `isrTaskLSST`.

5.2. *Amplifier Offset Correction*

The amplifier offset correction (commonly referred to as amp-offset correction, or pattern continuity correction) runs as part of the instrument signature removal (ISR) process. This correction is designed to address systematic discontinuities in background sky levels across amplifier boundaries. We believe that these discontinuities arise from electronic biases between adjacent amplifiers, persisting even after the application of dark and flat corrections.

Drawing on the `PANSTARRS`' Pattern Continuity algorithm (C. Z. Waters et al. 2020), our method aims to eliminate these offsets, thereby preventing problems such as background over-/under-subtraction at amplifier boundaries caused by discontinuities across the detector.

The amp-offset algorithm initially computes a robust flux difference measure between two narrow strips on opposite sides of each amplifier-amplifier interface. Regions containing detected sources, or pixel data which have been masked for other reasons, are not considered. These amp-interface differences are stored in an amp-offset matrix; diagonal entries represent the number of neighboring amplifiers, and off-diagonal entries encode information about the associations between amplifiers. A complementary interface matrix encodes directional information for these associations. Using this information, a least-squares minimization is performed to determine the optimal pedestal value to be added or subtracted to each amp which would reduce the amp-offset between that amplifier and all of its neighboring amplifiers. This method is generalized to support 2D amplifier geometries within a detector, as with LSSTCam, incorporating length-based weighting into the matrices to account for amplifiers that are not square.

6. DETECTION AND MEASUREMENT

We perform detection and measurement on images with the `meas` framework. We distinguish between *detection* and *measurement*:

- *detection*: identifying *Footprints* (TODO: add afw link!) of sources as being above a given flux or signal-to-noise level (see 6.2.1).
- *measurement*: running `plugins` on each source in the image to compute properties of that source (e.g. a centroid or aperture flux) (see below).

We also distinguish between measurement on the original detection image (*single-frame measurement*) vs. measurement on a different image from the original detection (*forced measurement*). Measurement could be performed on a single raw or calibrated image, a coadd of multiple images, or a difference of images: from the perspective of a measurement plugin, there is no difference between these cases. *forced measurement* is performed on one image, using a ”reference” catalog of sources that were detected on another image.

6.1. *meas_base*

The `meas` framework interface is defined in the `meas_base` package. Measurement plugins have the `Plugin` suffix if they are defined in python, and the `Algorithm` suffix if they are defined in C++. This package defines base classes for plugins (`SingleFramePlugin`, `ForcedPlugin` in python; `SingleFrameAlgorithm`, `ForcedAlgorithm` in C++) and the measurement tasks that can be configured to run them (`SingleFrameMeasurementTask`, `ForcedMeasurementTask`, `CatalogCalculationTask`), as well as some concrete implementations of plugins (`ApertureFluxAlgorithm`, `BlendednessAlgorithm`, `CircularApertureFluxAlgorithm`, `GaussianFluxAlgorithm`, `LocalBackgroundAlgorithm`, `PeakLikelihoodFluxAlgorithm`, `PixelFlagsAlgorithm`, `PsfFluxAlgorithm`, `ScaledApertureFluxAlgorithm`, `SdssCentroidAlgorithm`, `SdssShapeAlgorithm`). Each plugin has an associated config class, suffixed with `Config` in python or `Control` in C++ (e.g. `SdssCentroidAlgorithm` has `SdssCentroidControl`), used to configure parameters of that specific algorithm.

6.1.1. *Measurement plugins*

Plugins are added to a *registry*, so that they and their outputs can be referred to by a shorter common name that identifies the package it was defined in, for example `lsst.meas.base.SdssCentroidAlgorithm` is registered as `base_SdssCentroid`. This way, measurements produced by each plugin will have consistent, distinct names in the output schema, e.g. `base_SdssCentroid_x`, `base_SdssCentroid_y`, `base_SdssCentroid_flag`.

Measurement plugins often depend on each other, and must be run in a particular order. Rather than creating a directed acyclic graph to denote the dependencies, the plugins are batched and and are run in any order within a batch. The batch order is defined by the `getExecutionOrder` method, with smaller execution numbers being run first. `BasePlugin` defines a list of named constants for particular cases:

1. `CENTROID_ORDER` for plugins that require only footprints and peaks
2. `SHAPE_ORDER` for plugins that require a centroid to have been measured
3. `FLUX_ORDER` for plugins that require both a shape and centroid to have been measured.

Measurement plugins output their results to a `SourceCatalog` (TODO: crosslink to afw section!), which has a *slot* system for predefined aliases to allow a plugin to get a value without knowing exactly what plugin originally computed that value, e.g. `slot_Centroid` could point to `base_SdssCentroid`, or some other plugin that measures centroids.

6.1.2. *SingleFrameMeasurementTask*

Single frame measurement requires a catalog of detected source `Footprints`, which could still be blended, or could have been deblended (TODO: crosslink?). When initialized, the task creates a schema from the configured plugins, which defines the contents of the output catalog and cannot be modified after initialization.

Before performing any measurement, this task replaces all sources with noise (via the `NoiseReplacer`) in the regions defined by their detected `Footprints`. The task then loops over all ”parent” sources (those that were not deblended and those that represent the un-deblended state of blends), and then loops over all ”children” of parents (if any). For each such source, the source footprint is re-inserted into the image, all measurement plugins are run, and the footprint is then replaced with noise again. Then, for blended sources, the parent is inserted, measured (running plugins on both the parent and jointly on all the children via `measureN`), and again removed.

6.1.3. *ForcedMeasurementTask*

Forced measurement uses the known pixel position of objects from a reference catalog to constrain measurements on another image. Typically only photometric measurements are scientifically useful, as the centroid and shape are defined by the reference catalog, and transformed to the coordinate system of the image being measured on (e.g. shifting to the appropriate x/y origin, or transforming through the respective WCSs). Other than this coordinate transformation, forced measurement proceeds much like single frame measurement above. Two concrete implementations of the task include `ForcedPhotCcdTask` for single-visit images and `ForcedPhotCoaddTask` for coadd patch images, both using the output of a previous single frame measurement run on coadds as the reference catalog.

6.2. *meas_algorithms*

The `meas_algorithms` package contains a wide variety of astronomical algorithms. We briefly describe some of them here; for the full list of `Tasks` defined in this module, see the full package documentation.

- `MeasureApCorrTask` measures aperture corrections on an image (TODO: how? Eli?).
- `NormalizedCalibrationFluxTask` measures SOMETHING TODO: Eli?
- `ObjectSizeStarSelectorTask` is used to find likely PSF-like sources to be used to fit a PSF model during initial calibration.
- `SkyObjectsTask` generates ‘sky object’ `Footprints` on regions of an image that do not have a `DETECTED` mask plane set (TODO: link to afw Mask!).
- `SubtractBackgroundTask` fits and subtracts the background of an image, potentially appending it to an earlier fitted background model.
- `ScienceSourceSelectorTask` and `ReferenceSourceSelectorTask` select sources from a catalog given a set of configurable criteria.

This package also contains tools for defining and converting existing third party catalogs to be used as reference catalogs by Science Pipelines code, via `ConvertReferenceCatalogTask` and its commandline interface `convertReferenceCatalog`. These tools are described in more detail in the documentation for creating an LSST reference catalog.

6.2.1. *SourceDetectionTask*

We detect positive and negative sources on an image with `SourceDetectionTask` to produce a `SourceCatalog` of `Footprints`. This task requires that the image be background subtracted to produce good results. `SourceDetectionTask` convolves the image with a Gaussian approximation to the exposure PSF and detects peaks and footprints above a configurable threshold in either signal-to-noise or absolute flux level. The detected footprints may be significantly blended, depending on the detection threshold and source density in the input image: in order to separate footprints that contain many peaks, some form of deblending (TODO: section link!) must be performed.

6.2.2. *DynamicDetectionTask*

The `DynamicDetectionTask` is a specialized version of `SourceDetectionTask` that adapts detection thresholds based on the local background and noise. This task was initially developed to address detection efficiency issues noted in HSC data. First, `DynamicDetectionTask` detects sources using a lower detection threshold than normal. In so doing, we identify regions of the sky which are unlikely to contain real source flux. Next, a configurable number of sky objects are placed in these sky regions (1000 by default), and the PSF flux and standard deviation for each of these measurements is calculated. Using this information, we set the detection threshold such that the standard deviation of the measurements matches the median estimated error.

6.2.3. *MaskStreaksTask*

TODO: for Meredith or Clare?

6.3. *Deblending* [13]

Deblending in the science pipelines is performed differently for single-band (visit) image processing vs. multi-band (coadd) image processing. This section gives a basic description of each algorithm.

6.3.1. *Single-band Deblending*

Deblending on single-band images (ie. visit) is performed using the `meas_deblender` package and is based on the deblender used in SDSS (citation needed), with a few differences that will be discussed shortly. Similar to the SDSS deblender, the LSST deblender creates a template for each source in a blend using a very simple (yet computationally efficient) model for each peak position in a parent Footprint. Once a template has been created for each peak in the blend, the deblender combines all of the source templates into a single blend model by summing their values in each pixel. For each pixel in a source template, the ratio of the source template value to the total blend model is calculated and used to weight the pixel value from the image to create a model for each source. The source models are thus flux conserving in that adding them together will yield the original image except for pixels that do not appear in any of the individual templates. A cleanup algorithm is then run to allocate the remaining pixels to one of the sources in the blend based on a set of criteria including distance to the center, brighness of the nearest sources, etc.

6.3.2. *Deblender Template Generation*

The main ansatz of the SDSS deblending algorithm is that the flux from stars and galaxies in a ground based telescope is nearly 180 degree symmetric. Figure 3 illustrates how a 1D slice through the center of two blended sources can exploit this symmetry by setting the pixels on opposite sides of the (integer) center pixel to the minimum value of both pixels. In other words, for simple blends of only two sources the deblender can use the flux on the non-blended side to constrain the value of the flux on the blended side. Despite the fact that stars (PSFs) and galaxies are not exactly symmetric, especially since their position is not exactly centered in the center of a single pixel, this algorithm works quite well for generating templates in simple blends that very nearly approximate each source when redistributing flux from the image.

For sources with low SNR the algorithm fails due to noise in the image, generating galaxy templates that are typically very jagged and unphysical. To combat this, for each `peak` in the parent `Footprint` the deblender first attempts to fit the flux from the image with a simple PSF model that allows its position, amplitude, and a linear background, to vary. If the fit has a reasonable χ^2 value then the deblender will use this scaled PSF model as a template for the source. Only for sources that cannot be adequately modeled with the PSF are the symmetric templates used.

The main failure point of this algorithm is when three (or more) sources lie along the same axis. For example, Figure 4 illustrates a 1D slice through the center of three aligned sources. In this case the minimum pixel on each side of the central source cannot constrain the flux at that radial location and results in a template that has extra bumps from its neighbors. This turns out to be more catastrophic than one might expect. Notice that even the neighboring sources, which have very good templates created by using symmetry on their unblended side, have their resulting models contaminated due to the central object that steals flux from both of them. In single visits the number of ”three in a row” blends is small enough that we sacrifice the quality of the models for efficiency and still use the single-band deblender. For LSST-depth coadds this becomes a significant problem, as deep coadds can have as much as 40% of blends having 3 or more sources and a more sophisticated algorithm is needed.

6.3.3. *Multi-band Deblending*

The multi-band deblender is an implementation of the scarlet deblending algorithm described in P. Melchior et al. (2018). In our implementation, `scarlet_lite`, we have made our own set of simplifying assumptions that are different from the original scarlet algorithm to make it more efficient when used in a large ground based survey like LSST. Similar to the original `scarlet` we make the assumption that astrophysical objects can be thought of as a collection of components, where each component has the properties

- Components have a single color (spectrum) that is the same in all pixels over its shape (morphology)

[13] This section assumes that a dection section has already been written. Some changes might be necessary if there are topics not covered in the detection section or topics that are duplicated in this section.”

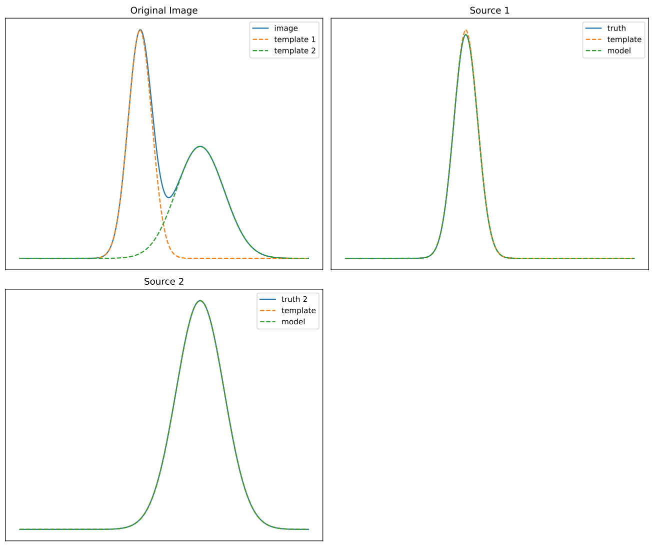

Figure 3. A 1D slice of two blended Gaussian sources illustrating how symmetry can be utilized to model blended sources.

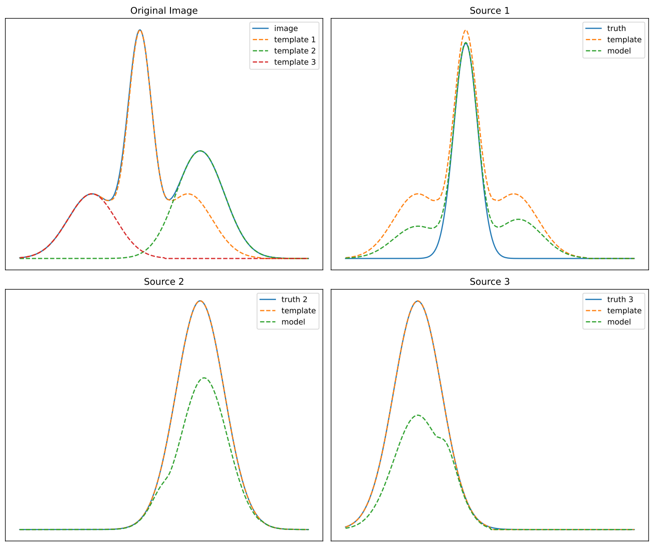

Figure 4. A 1D slice through three aligned Gaussian sources, demonstrating a failure case of relying on symmetry for generating deblender templates. Notice that for sources 2 and 3 the templates are reasonable but due to the inability of source 1 to use symmetry to constrain flux in the blended region, the resulting models for all three sources are poor. This catastrophic "three in a row" problem was part of the motivation for creating `scarlet` to incorporate spectral information and a more rigorous iterative deblending algorithm.

- Components have flux that monotonically decreases from the center
- Component flux is additive

The classic example is decomposing a single galaxy into bulge and disk components, where both the bulge and disk share a common center but have different spectra and morphologies. Something more complex, like a grand design spiral, could in theory be modeled as a source with multiple components, where spiral arms and star forming regions could still be thought of as separate monotonic components. For the science pipelines we ignore those more complicated structures, as detection typically already shreds large galaxies into multiple sources. Instead we use a signal to noise cut where low flux sources are modeled with a single component and higher flux sources are modeled with two components.

Scarlet lite initializes models with nearly the same templates as those generated by the single-band deblender. Using a χ^2-like monochromatic image created by weighting each band by its inverse variance, scarlet lite creates initial morphology models that are symmetric from the center in the monochromatic image, with the additional constraint that the flux is monotonically decreasing from the center. In order to satisfy the constraint that all pixels in the morphology have the same spectrum, scarlet models exist in a partially deconvolved frame with the seeing of a well sampled but narrow Gaussian. The initial spectrum of each source is determined using a least squares fit of each monochromatic morphology, convolved with the difference kernel in each band to match the image, for each component. It then uses proximal-ADAM (PADAM, P. Melchior et al. 2019) to iteratively update the spectrum and morphology with the given constraints until convergence or a maximum number of iterations is reached. It should be noted that although we do use symmetry to initialize the scarlet models, we do not implement a symmetry constraint and the final models are not guaranteed to be symmetric. The models are stored as the `deepCoadd_scarletModelData` data product, which contains all of the blends for a single patch. Like the single-band deblender, the `scarlet_lite` models are only used as templates to redistribute flux from the image and all measurements are performed on the flux redistributed models.

6.4. *meas_extensions_convolved*

6.5. *meas_extensions_gaap*

`meas_extensions_gaap` implements the Gaussian Aperture and PSF photometry (GAaP) algorithm (K. Kuijken 2008). It is an aperture photometry algorithm designed to obtain consistent colors of extended objects (i.e., galaxies). This is done by weighting each (pre-seeing) region of a galaxy by the same pre-defined 2D Gaussian function in all the bands and is thus largely insensitive to the seeing conditions in the different bands. In practice, this is done by first convolving each object by a kernel (using the same tools described in Sec. 7) so that the PSF is Gaussian and is larger by about 15% (this is configurable). As a second step, each Gaussianized object is then weighted with a Gaussian aperture so that the effective pre-seeing Gaussian aperture is the same for all objects in all the bands. The plugin is configured to use a series of circular Gaussian apertures, an elliptical Gaussian aperture (optionally) that matches the shape of the object in the reference band.

Although the two-step approach is motivated by the original implementation in K. Kuijken (2008), the implementation of this algorithm within the broader context of the measurement framework makes it different from the implementation used in the Kilo-Degree Survey (KiDS; A. H. Wright et al. 2025). In particular, because neighboring objects are replaced with noise before measurement, Gaussianization of the PSF does not result in increased blending as mentioned in Appendix A2 of K. Kuijken et al. (2015). Furthermore, the uncertainty handling is different. Correlations in noise introduced due to PSF-Gaussianization is included in the uncertainty estimates. However, because only per-pixel noise variance is tracked, the noise treatment is forced to assume that the noise is uncorrelated to begin with which is not true on the coadds. See A. Kannawadi (2022) for more details on the implementation details.

Note that this measurements from this plugin do not produce total fluxes, but should only be used to obtain colors. For total fluxes, measurements from `cModel` or `MultiProFit` (c.f. Sec. 6.12) are recommended.

6.6. *meas_extensions_photometryKron*

6.7. *PSF Modeling*

Within the pipeline, three distinct PSF models are defined: `pcaPsf`, `PSFex`, and `Piff`. Only `PSFex` and `Piff` are currently used. `PSFex` is a fast, and less accurate PSF estimation and is wrapped within `meas_extensions_psfex`. In contrast, `Piff` is a slightly slower, but more accurate PSF estimation that is incorporated in `meas_extensions_piff`. Both `meas_extensions_psfex` and `meas_extensions_piff` are described below.

6.7.1. *meas_extensions_psfex*

The `meas_extensions_psfex` package provides an interface to a patched version of the PSFEx tool (E. Bertin 2011) for modeling spatially varying point spread functions (PSFs). It uses PSF candidates typically selected from detected sources using a configurable star selector that selects clean, isolated stars. At its core is `PsfexPsfDeterminerTask`, which prepares these selected stars for input into PSFEx, runs the external binary, and converts the output into an LSST-specific PSF object (`PsfexPsf`). Key parameters such as spatial interpolation order and oversampling ratio are controlled via `PsfexPsfDeterminerConfig`.

For each CCD in the focal plane, PSFEx independently models the PSF as a linear combination of basis vectors and captures spatial variation using polynomial interpolation. `PsfexStarSelectorTask` offers a built-in mechanism for star selection using strict cuts on signal-to-noise ratio, FWHM range, ellipticity, and quality flags. The package raises specific exceptions for common failure modes: when no stars are available, when none pass quality cuts, or when too few good stars remain to support the required model complexity–as determined by the degrees of freedom needed for the fit. In the latter case, the insufficient sample size forces the PSFEx model to reduce its polynomial degree to an unsupported level. The code also includes hooks for visual debugging (via `afwDisplay` and optional `matplotlib` plots), enabling visual inspection of PSF candidates and rejection reasons during development or QA.

`meas_extensions_psfex` is used in contexts where PSFEx compatibility is required or when modeling speed is preferred over robustness. Another external PSF modeler (discussed in Section 6.7.2) provides greater robustness but is slower in comparison.

6.7.2. *meas_extensions_piff*

The `meas_extensions_piff` package is a wrapper around the PSF package `Piff` used to estimate and compute the PSF (M. Jarvis et al. 2021a,b). `Piff` is a modular package that supports various PSF models, interpolation schemes, coordinate systems, and can operate on a per-CCD basis or over the full field of view, as indicated by its name. The implementation within `meas_extensions_piff` does not exploit the full modularity of `Piff`; instead, it closely follows the method used for cosmic shear analysis like in DES (M. Jarvis et al. 2021b; T. Schutt et al. 2025).

The PSF model utilized is a `PixelGrid`, and the interpolation is performed using `BasisPolynomial` interpolation (M. Jarvis et al. 2021b). Modeling is executed per CCD and can employ either pixel or sky coordinates. A key difference from `PSFex` is that `Piff` implements outlier rejection based on chi-squared criteria (see M. Jarvis et al. 2021b, for more details).

Most of the configuration described here is adjustable through the `PiffPsfDeterminerConfig` that are exposing some of the configurable parameters of `Piff` and can be fine-tuned for a dedicated survey. However, some important features that were implemented by M. Jarvis et al. (2021b) and T. Schutt et al. (2025) have not yet been enabled but will be available in the near future. While M. Jarvis et al. (2021b) operates in sky coordinates with a WCS that includes CCD distortions such as treerings, `meas_extensions_piff` can work in sky coordinates and incorporate WCS; as written, it does not, however, account for CCD distortions like tree rings. Additionally, although T. Schutt et al. (2025) incorporated a color correction to account for chromatic effects on the PSF, this correction has not yet been implemented in `meas_extensions_piff`.

6.8. *meas_extensions_shapeHSM*

The `meas_extensions_shapeHSM` package contains the plugins to measure the shapes of objects. The plugins measure the moments of the sources and PSFs with adaptive Gaussian weights. The algorithm was initially described in C. Hirata & U. Seljak (2003) and was modified later in R. Mandelbaum et al. (2005). The implementation of these algorithms lives within the `hsm` module of the GalSim package (B. T. P. Rowe et al. 2015). `meas_extensions_shapeHSM` now interacts directly with the Python layer of GalSim to make the measurements.

The base plugin for measuring moments is the `HsmMomentsPlugin` and is the parent class of the `HsmSourceMomentsPlugin` and `HsmPsfMomentsPlugin` which are specialized to measure on the sources (and objects) and PSFs respectively. `HsmSourceMomentsRoundPlugin` is a further specialized plugin that measures the moments with circular Gaussian weights instead of the elliptical ones in `HsmSourceMomentsPlugin`. The `HsmPsfMomentsDebiasedPlugin` adds noise to the PSF image to degrade it to have the same signal-to-noise ratio (SNR) as the source image. This makes the ellipticity calculated from this plugin have the same bias as the source ellipticity The PSF moments from this plugin should be used when calculating ellipticity residuals so the bias is largely cancelled. Having the various specializations as distinct plugins allows an object to be measured under different configurations simultaneously and included in the output catalogs.

In addition to the plugins that measure (adaptive) weighted moments, there are also a series of `HsmShape` plugins to estimate the PSF-corrected ellipticities of objects. In particular, the outputs from `HsmShapeRegaussPlugin` have been used to measure weak gravitation lensing signals in the Hyper Suprime-Cam SSP data (R. Mandelbaum et al. 2018; X. Li et al. 2022).

6.9. *meas_extensions_simpleShape*

6.10. *meas_extensions_trailedSources*

6.11. *meas_modelfit*

6.12. *meas_extensions_multiprofit*

MultiProFit is a package for Gaussian mixture model fitting (D. S. Taranu 2025). MultiProFit is primarily used to provide multiband Sersic model fits to objects using all available coadds. The `multiprofit` package is a standalone Python-only package that provides the interfaces for astronomical object fitting. `multiprofit` depends primarily on `gauss2d_fit`, a standalone C++ package with Python bindings for fast evaluation of Gaussian mixture model likelihoods and gradients thereof. `gauss2d_fit` in turn is an extension of `gauss2d`, providing additional classes for parameters with arbitrary limits and transformations from the `modelfit_parameters` header-only C++ library. All of these packages are included in the science pipelines but can also be installed independently, as `multiprofit` only depends on other standalone packages like `pex_config`.

The `meas_extensions_multiprofit` package contains pipeline tasks (with interfaces defined in `pipe_tasks`) necessary to run `multiprofit` on coadded and deblended images. The first of these tasks fits a Gaussian mixture model to the PSF model image at the location of each object in a patch. This procedure is similar to the shapelet PSF fitting functionality in `meas_modelfit`. The main differences are that the components are pure Gaussians (shapelet parameters are not supported), can have independent shapes, and are constrained to have integrals summing to unity (i.e. they are normalized). Currently, only a maximum of two components are supported; this limitation may be removed in the future.

The remainder of the tasks in `meas_extensions_multiprofit` use the Gaussian mixture PSF model to fit a PSF-convolved model to all objects in a given patch, for all available bands. Convenient tasks are available for a variety of models, including a single Sersic, as well as multi-component bulge-disk models with an optional central point source component. In all cases, the structural parameters for each component are band-independent, with a separate total flux parameter for each band. That is, individual components do not have intrinsic color gradients (although the convolved models might, if the PSF parameters vary by band).

6.13. *Reliability Scoring*

The `meas_transiNet` package determines a numerical score for input cutout images using pre-trained machine-learning models. Image differencing may produce false detections, so time-domain surveys chacteristically use machine learning classifiers to distinguish astrophysical sources from artifacts ("Real/Bogus;" e.g., J. S. Bloom et al. 2012; D. A. Goldstein et al. 2015; D. A. Duev et al. 2019).

The `meas_transiNet` defines "model packages" that consist of a python architecture class, a PyTorch (A. Paszke et al. 2019) weights file, and associated metadata. The inference task may be configured to load a model package from disk or from the Butler.

The `RBTransiNetTask` PipelineTask takes as input three square cutouts of configurable size from the science, template, and difference images centered on the location of a source. These images are concatenated, batched into Torch blobs, and passed to the model for inference. Either CPU or GPU backends may be used for inference. The output of the task is a single float ranging from 0–1 for each cutout triplet, with higher values indicating that the DIASource is more likely to be astrophysical. These reliability scores are then joined with the DIASource catalogs by a later transformation task. Detailed discussion of the model architecture, training, and performance will be presented in T. Acero Cuellar et. al (in prep.).

7. DIFFERENCE IMAGING

Difference imaging is implemented in `ip_diffim`, and is divided into three steps. First, a base template image is constructed with `getTemplate` by warping previously-generated coadded images to the WCS and bounding box of the science image. Then the warped template is subtracted from the science image using one of several available algorithms in `subtractImages`, which produces a temporary difference image. Finally, peaks are detected on the difference image and DiaSources are measured in `detectAndMeasure`. The final difference image with updated mask planes is written along with the DiaSource catalog.

7.1. *subtractImages*

The primary implementation of image subtraction used by `subtractImages` is based on C. Alard & R. H. Lupton (1998), and uses spatially-varying Gaussian basis functions for the fit. The PSF-matching kernel can be constructed for either the science or the template image, and the resulting difference image is decorrelated D. J. Reiss & R. H. Lupton (2016). Optionally, the science image can be preconvolved with its own PSF before PSF-matching, producing a Score image analogous to B. Zackay et al. (2016).

7.2. *detectAndMeasure*

Positive and negative peaks are detected by thresholding the Score image if it is available. Otherwise, the difference image is smoothed with a Gaussian of the same width as the PSF of the science image, and thresholds are taken on the smoothed image. Contiguous pixels around each peak that are statistically brighter than the background are grouped into source footprints, and any overlapping footprints are merged. Footprints that contain both a positive and a negative peak are fit as dipoles. The dipole fit simultaneously solves for the negative and positive lobe centroids and fluxes using non-linear least squares minimization. DiaSources that are not classified as dipoles instead fall back on an SDSS-style centroid (J. R. Pier et al. 2003). Finally, all configured measurement plugins are run, including HSM shape measurements (C. Hirata & U. Seljak 2003; R. Mandelbaum et al. 2005) and trailed source measurements.

8. ASTROMETRIC AND PHOTOMETRIC CALIBRATION

8.1. *Astrometric Calibration*

Astrometric calibration is performed in two steps. First, an astrometric fit is done for single frames, which produces an astrometric solution sufficient for Alert Production and difference imaging. A final astrometric solution is then fit using the ensemble of images from a given band overlapping with a given tract. This final astrometric solution is then used for all downstream tasks. A full description of astrometric calibration in the pipeline is given in C. Saunders (2024).

8.2. *meas_astrom*

Single frame astrometric fits are performed by `AstrometryTask` in `meas_astrom` and run in `CalibrateImage` (TODO: crosslink?). This task matches a catalog of sources detected and measured on an image to a reference catalog and solves for the *World Coordinate System* (WCS) of the image. Matching and WCS fitting are performed iteratively, to reject astrometric outliers. The matcher is either the optimistic (`MatchOptimisticBTask`) or pessimistic (`MatchPessimisticBTask`) matcher from V. Tabur (2007), with the pessimistic matcher used by default due to better performance on dense fields; see (C. B. Morrison 2018) for details. The WCS fitter can be a simple affine model on top of the known camera geometry (TODO: link to afw cameraGeom section!), as in `FitAffineWcsTask`, or a FITS TAN-SIP WCS

(D. L. Shupe et al. 2005), as in `FitTanSipWCSTask` or `FitSipDistortionTask`. We default to fitting the simple affine model because we have a well fit distortion model from running `gbdes` in DRP (§8.3), thus we do not need the extra degrees of freedom provided by a TAN-SIP model.

Because `AstrometryTask` only requires a single image, it is suitable for use during Alert Production, which does not have access to the entire focal plane. This single frame astrometric fit is sufficient for initial calibration and difference imaging (assuming the modeled camera geometry is a close match to the true camera distortions), but during DRP we perform a full focal plane fit with `gbdes`.

8.3. *gbdes*

The final astrometric solution is fit by `GbdesAstrometricFitTask` in `drp_tasks`, which runs the `wcsfit` fitter from the `gbdes` package (G. M. Bernstein 2022; G. M. Bernstein et al. 2017) on the ensemble of images in a given band overlapping with a given tract. This task fits a per-detector polynomial distortion model, a per-exposure polynomial distortion model, and position for all the isolated star sources in the component images. This is done by first associating all isolated point sources in the input images and matching them with an external reference catalog. The model is then fit by iterating between fitting the per-detector and per-exposure polynomial models, and recalculating the best-fit solution for the object positions.

The task can be configured to fit either a two-parameter (position on the sky) or five-parameter (position, proper motion, and parallax) solution for the input objects. Correcting for differential chromatic refraction is another configurable option.

There are also options to run variants of the main `GbdesAstrometricFitTask`: `GbdesAstrometricMultibandFitTask` fits images from multiple bands at once, in which case the per-detector distortion model is also per-band; `GbdesGlobalAstrometricFitTask` removes the restriction to a single tract and fits images regardless of their location on the sky by splitting the images into contiguous groups; `GbdesGlobalAstrometricMultibandFitTask` combines these two options.

Lastly, the per-detector polynomial model fit by the task is also used to build a camera distortion model, which can be fed back into single-frame modeling or into the `gbdes` fit for other data. For use in single-frame modeling, the `BuildCameraFromAstrometryTask` subtask is used to build an `afw` Camera object out of the native polynomial model.

8.4. *Photometric Calibration*

8.4.1. *PhotoCal*

TODO: Eli should look at this? Single frame photometric calibration is performed by `PhotoCalTask`, in the `pipe_tasks` package. This task requires that the input catalog come from an image with a good astrometric solution. The catalog to be calibrated is down-selected to be bright ($S/N > 10$), well measured, PSF-like sources which are then matched to a reference catalog. The matched sources have their instrumental fluxes converted into rough magnitudes, which are are iteratively compared with the reference catalog magnitudes using a sigma-clipping algorithm, to fit a single magnitude zero point to the whole image.

Because `PhotoCalTask` only requires a single image, it is suitable for use during Alert Production, which does not have access to the entire focal plane. This single frame photometric fit is sufficient for initial calibration and difference imaging (assuming the flat field calibrations applied during ISR are a close match to the true instrument response), but during DRP we perform a full focal plane fit with `gbdes`.

8.5. *fgcmcal*

Global photometric calibration is computed by use of the Forward Global Calibration Method (FGCM D. L. Burke et al. 2018), as adopted for LSST Science Pipelines (P. Fagrelius & E. Rykoff 2025). This global calibration algorithm makes use of repeated observations of stars in all *ugrizy* bands, combining a forward model of the atmospheric parameters with instrumental throughputs measured with auxiliary information. In this way we simulateously constrain the atmospheric model as well as standardized top-of-atmosphere (TOA) star fluxes over a wide range of star colors, including full chromatic corrections from the instrument and atmosphere.

Running `fgcmcal` first requires generating a look-up table. The input to the look-up table includes the effect of a MODTRAN (A. Berk et al. 1999) atmospheric model at the elevation of the observatory, as well as the throughput as a function of wavelength and position from the optics, filters, and detector quantum efficiency. The quality of the output (in terms of repeatability of bright isolated stars across a wide range of colors) depends on the knowledge of the instrumental throughput.

The primary goal of fgcmcal is to provide a uniform relative photometric calibration of the survey. For "absolute" (relative) calibration, a reference catalog can be used as an additional constraint on the fit. Thus, the overall throughput output by `fgcmcal` depends on the reference catalog. This can be checked with (e.g.) specific white dwarfs or CALSPEC (R. C. Bohlin 2007) stars in the survey. However, the relative spatial and chromatic calibration of the `fgcmcal` calibration means that the absolute calibration reduces to a set of 6 numbers (one for each band, or one overall throughput and 5 absolute colors).

8.6. *jointcal*

`jointcal` fits both astrometry and photometry across multiple exposures of large mosaic cameras, fitting for both the true star positions/fluxes, and the distortions caused by the telescope and instrument. `jointcal` is no longer used used by the LSST camera pipeline, but is available for use by cameras that are not supported by `gbdes` and/or `fgcmcal` (for example, DECam). More details on the `jointcal` algorithm are available in (J. P. U. of Washington) & P. A. L. Paris) 2018).

9. SOURCE ASSOCIATION

The `ap_association` package contains multiple tasks for standardizing newly detected DiaSources and associating them with existing or new DiaObjects. Standardization converts the output catalogs from 7 to the format specified in `sdm_schemas` (Sec. 12), and applies filtering consistent with (W. O'Mullane et al. 2024). Once DiaSource catalogs are standardized, they are associated to DiaObjects in either of two modes: Data Release Production (DRP) or Alert Production (AP). Both implementations use the Pessimistic Pattern Matcher B (C. B. Morrison 2018) to score and match DiaSources, but differ in how DiaObjects are stored and how visits are ordered.

- DRP association loads all DiaSource catalogs from a set time period overlapping a single patch at once, and creates new DiaObjects for matched DiaSources from all visits simultaneously.
- AP association processes a single visit at a time, and creates new DiaObjects incrementally from unassociated DiaSources. DiaObjects and their associated DiaSources are stored in the Alert Production Database (APDB) (Sec. 9.2).

After association, an additional filtering step may be applied to DiaSources with no matched DiaObject of Solar System object (Sec 9.1). Properties of the source such as its reliability score (Sec. 6.13, source flags, or signal-to-noise cuts may be used to drop detections that are likely to be false detections and avoid creating erroneous new DiaObjects.

9.1. *Solar System object association*

Ephemerides from known Solar System objects are preloaded with approximate locations for the expected time of observation in `mpSkyEphemerisQuery`. Since these are loaded in Prompt Processing before the science image arrives and is calibrated, the orbital fit parameters are used in association to correct the position to the midpoint of the observation, including the shutter motion profile since the shutter takes a second to cross the focal plane. Solar System objects are associated to DiaSources using the closest match within a configurable radius.

9.2. *Alert Production Database (APDB)*

The Alert Production Database (APDB; A. Salnikov & J. McCormick 2024)) supports SQL, Postgres, and Cassandra database formats. The previous history of DiaObjects, DiaSources, and DiaForcedSources for the region containing the science image is loaded with `loadDiaCatalogs`, which are passed to `diaPipe` for association. Loading is split from the association step to enable preloading of catalogs from the database in Prompt Processing during the interval when the next visit has been scheduled but the images have not yet been taken. When AP-style association is run outside of Prompt Processing, it is therefore essential to process all association tasks in strict visit order to prevent loading catalogs from the APDB prematurely and losing DiaObject history in association.

10. ALERT GENERATION

In order to to enable real-time science, the AP pipelines generate alert packets for each detected DIASource. These packets are serialized in Apache Avro[14] format and then transmitted to community alert brokers via Kafka for further processing. M. Patterson et al. (2020) provides a high-level overview of the alert system.

Within the pipelines, alert packets are constructed by `packageAlertsTask` within `ap_association`. Alert packets contain the triggering DIASource record; the associated DIAObject or SSObject record; up to twelve months of past history from DIASources, DIAForcedSources, and/or upper limits; and cutout images of the science, template, and difference images centered at the position of the cutout. Cutouts are provided as FITS images serialized by the astropy `CCDData` class, and in-

[14] https://avro.apache.org/

clude image, variance, and mask planes along with WCS information and an image of the approximate PSF.

Avro schemas are stored in the `alert_packet` package. They are derived from the corresponding AP schemas in `sdm_schemas` used to instantiate the AP databases.

11. PIPELINES

11.1. *Pex Config*

Pex Config is the foundational configuration system for the LSST Rubin Observatory's ambitious science pipelines. It's far more than a simple parameter parser; it's a framework that mediates between diverse configuration sources and the complex software that processes astronomical data. At its core, Pex Config functions as an intermediate representation, decoupling the pipelines from the specifics of configuration file formats (like YAML, JSON) and providing a unified, Python-native interface to all configurable parameters. This intermediate representation, resembling a Domain Specific Language embedded within Python, also allows leveraging the full power of a programming language for parsing or setting configuration values. An example of this can be seen in the following code block which shows a fragment used to configure one of the shape measurement routines. This abstraction is critical for maintainability, allowing the underlying file formats and or execution systems to evolve without impacting the pipeline code. It also provides a mechanism to deprecate configurables which will change in future versions of the software stack, allowing users an easy migration path.

```python
import os.path
from lsst.utils import getPackageDir

try:
    location =
        getPackageDir("meas_extensions_shapeHSM")
    path = os.path.join(, "config", "enable.py")
    config.load(path)
    plugins = config.plugins
    plugin =
        plugins["ext_shapeHSM_HsmShapeRegauss"]
    plugin.deblendNChild = "deblend_nChild"
    # Enable debiased moments
    config.plugins.names |=
        ["ext_shapeHSM_HsmPsfMomentsDebiased"]
except LookupError as e:
    print("Cannot enable shapeHSM (%s): disabling
        HSM shape measurements" % (e,))
```

Listing 1. Code configuration in python

The design of Pex Config centers around the concepts of "Fields" and "Config" objects. Fields represent individual configurable values – things like exposure times, image quality thresholds, or database connection strings. Each Field is strongly typed, supporting a variety of data types (such as integers, floats, strings, booleans, and lists). Config objects, on the other hand, are containers that group related Fields together, creating logical units of configuration. One of the highlights of Pex Config is its composability. Config objects can be nested within other Config objects using a special "ConfigField," allowing for the creation of complex, hierarchical configuration trees that mirror the structure of the pipelines themselves. This allows for modularity and reuse of configuration components across different parts of the system.

A strength of Pex Config is its flexible application of configuration values. Values can be set at multiple stages: via command-line arguments, loaded from configuration files, or defined directly within the pipeline code. Importantly, these stages are applied progressively, with later stages overriding earlier ones. This allows for a powerful combination of default settings, user-defined customizations, and dynamic adjustments. Mechanisms also exist to apply values to all instances of a particular Config object within a tree, simplifying the management of shared parameters and ensuring consistency.

Beyond runtime configuration, Pex Config is deeply concerned with data provenance and reproducibility. It provides mechanisms for persisting and restoring configuration values, allowing for complete tracking of pipeline parameters used in a particular data processing run. Crucially, it also maintains a history of each Field's value, recording when and where it was set – whether via the command line, a configuration file, or programmatically. This detailed history is invaluable for debugging, auditing, and ensuring the reproducibility of scientific results. The system also incorporates robust validation mechanisms, enabling checks on individual Fields and groups of values before they are used by the pipelines, preventing errors and ensuring data quality. Validation can range from simple type checking, ensuring values fall within acceptable ranges or specific patters, to complex custom functions that enforce specific constraints.

Finally, Pex Config is designed with documentation in mind. All Fields and Config objects can be richly documented using documentation strings and attributes. This documentation structure is not only readable by humans but can also be parsed by automated tools to generate comprehensive documentation pages, eliminating the need for manual documentation creation. This ensures that the configuration system is well-documented and easy to understand, even for new developers. The system is flexible enough that it has been adopted by the DRAGONS software (K. Labrie et al. 2023).

11.2. *Pipeline Support*

The `Task` Python class provides a standard interface for how to execute an algorithm and has an associated `Config` class which contains its configurable parameters. The `PipelineTask` variant provides stronger guarantees on configuration and provides a means by which the pipeline execution framework can determine how to link a task into a pipeline and how to determine what type of data should be read from a Butler and what should be written out to a Butler.

Pipeline in YAML.

Show plot of a simple pipeline visualization.

Graph building.

Show plot of a graph where a pipeline now includes specific datasets as inputs.

Describe that provenance is stored in the output files and in the graph itself.

Execution system and how BPS provides the interface between a quantum graph and a workflow system.

11.3. *Task library*

11.4. *`pipe_tasks`*

Many subsections!

11.4.1. *`drp_tasks`*

Coaddition Tasks

The LSST Science Pipelines provide a modular task framework for constructing coadds from multiple single-epoch images. Coadds are used as static-sky templates for image subtraction and detecting and measuring faint sources. The coaddition process is divided into two main stages: resampling the input images onto a common projection and stacking those resampled images into a single coadd. Each stage is implemented via configurable tasks that allow the pipelines to be adapted for different instruments and observing strategies. The first step in coaddition is to resample each single-epoch exposure onto a common projection and pixel grid called a *skyMap*. This step is performed by the following Tasks:

- `MakeDirectWarpTask` performs a straightforward resampling of calibrated exposures.
- `MakePsfMatchedWarpTask` also convolves them to a configurable, common model-PSF. This PSF-homogenized variant is useful when the scientific goals require uniform PSF properties across the coadd and is used for artifact rejection during coaddition

All warping tasks use a configurable interpolation kernel. A 5th-order Lanczos kernel is used by default, balancing fidelity and computational efficiency. Input images are geometrically transformed using the World Coordinate System (WCS) and interpolated onto the target projection defined by a tract and patch geometry. Each resulting resampled image is called a `warp`.

Once warps are generated, they are stacked into a final coadd by `AssembleCoaddTask` or one of its subclasses. The default implementation, `CompareWarpAssembleCoaddTask`, performs outlier rejection to remove transient artifacts such as cosmic rays, satellite trails, and moving objects. The algorithm compares pixel values across epochs and masks those that significantly deviate from the expected distribution. The artifact rejection algorithm is detailed in Y. AlSayyad (2019). By default, weights for stacking are derived from the inverse of the average variance of each warp, with optional filters on PSF quality and seeing. The stacked image is accompanied by a mask plane and variance map, and the set of input PSF models is combined into a spatially-varying coadd PSF model (`CoaddPsf`) to serve as the PSF model for the coadd.

Users can adapt the pipeline to their data by modifying warp selection criteria, choosing a projection as defined in a `SkyMap` object, and adjusting artifact rejection parameters. While the default LSST pipelines build direct (non-PSF-matched) coadds for source detection and measurement, the modularity of the task framework makes it easy to substitute PSF-homogenized coadds or experiment with alternative stacking strategies (J. Bosch 2016).

11.4.2. *Pipeline Visualization*

Visualizing pipeline execution is crucial for understanding task dependencies, debugging, optimizing workflows, and ensuring correct data flow within the LSST Science Pipelines. To support this, `pipetask build` provides several options for visualizing the pipeline graph–a simplified directed acyclic graph that shows how tasks relate to dataset types, without including data IDs.

A text-based view can be generated using `pipetask build --show pipeline-graph`, which outputs an ASCII-style diagram. This format is especially useful for quick inspection or when working in a terminal-only environment. For graphical visualization, the `--pipeline-dot` and `--pipeline-mermaid` options export the pipeline graph in Graphviz DOT[15] and Mermaid[16] formats, respectively. The Mermaid format is particularly well-suited for sharing in accessible, web-based contexts.

[15] https://www.graphviz.org

[16] https://mermaid.js.org

Unlike DOT files, which typically require rendering with external tools like Graphviz's `dot`, Mermaid definitions can be directly rendered in Markdown-based platforms such as GitHub, GitLab, some Jupyter environments, and even Slack with the appropriate plugin. This makes Mermaid an effective format for generating interactive, easily shareable pipeline graphs that can be directly embedded in documentation, notebooks, or code review tools.

Figure 5 shows a visualization of a subset of two tasks from the `LSSTComCam/DRP-v2.yaml` pipeline using the Mermaid format. The diagram shows the relationships between tasks and their input and output datasets as well as the sequence in which the tasks are expected to run. Such visualizations can help uncover misconfigurations, missing inputs, or unexpected data dependencies that might otherwise result in issues such as empty QuantumGraphs or failed pipeline execution.

11.5. *Pipeline Collections*

11.5.1. *Calibration pipelines*

The pipelines to build calibration products (`cp`) for the LSST cameras are defined in `cp_pipe`[17]. They set `isrTaskLSST` configuration parameters needed for each calibration product, by enabling all the sequential steps of the ISR task up to the step before the correction being generated. In some cases, configurations also specify whether to combine exposures (for bias or dark exposures for instance) and to bin exposures to support display.

Once calibration products are produced, they are "verified" (see C. Waters (2025) for more details) using `cp_verify`[18] pipelines by checking they pass metrics defined in R. Lupton et al. (2025). In this case, verify configuration parameters enable all corrections in the ISR task up to and including the application of the correction being verified. As a result, the calibration products can then be certified to be available in the `butler` and used to ISR an exposure.

11.5.2. *`drp_pipe`*

11.5.3. *`ap_pipe`*

The `ap_pipe` package defines the pipeline(s) to be used for real-time Alert Production processing (K.-T. Lim 2022). These pipelines include instrument signature removal (§5), calibration (§**??**), measurement plugins (§6), image differencing (§7), source association (§9), and alert generation (§10). Some of these tasks are shared with the pipelines in `drp_pipe`, but configured to prioritize speed over strict quality; for example, they use a minimal set of measurement plugins.

`ap_pipe` currently has pipeline variants for LSSTCam, LSSTComCam, LATISS, the Rubin Observatory simulators, Hyper-SuprimeCam, and the Dark Energy Camera. Because these variants serve as testbeds for AP-specific algorithms and configuration settings, they are, as much as possible, the "same" pipeline, differing almost entirely in loading instrument defaults from `obs` packages (§3.2). The only other customization is an extra task for handling DECam's inter-chip crosstalk, which does not have an equivalent for Rubin instruments.

[17] https://github.com/lsst/cp_pipe and see documentation at https://pipelines.lsst.io/modules/lsst.cp.pipe/constructing-calibrations.html

[18] https://github.com/lsst/cp_verify

12. CATALOG SCHEMAS

Must transform pipeline products from the internal data model to the public data model defined in M. Jurić et al. (2023).

`sdm_schemas`

`felis` (J. McCormick et al. 2024)

13. DISPLAY ABSTRACTIONS

The Python object representing an image with metadata is a bespoke object not understand by generic tooling. To display an image we provide a display abstraction layer that allows the image to be displayed and graphics overlaid by using a plugin mechanism.

In some plugins the pixel data can be extracted from the exposure object and sent directly to display, in other plugins we form a simple single HDU FITS image (possibly with simplified world coordinates) and pass the temporary FITS file to the display system.

There a currently plugins for matplotlib (J. D. Hunter 2007), Firefly (W. Roby et al. 2020), SAOImage DS9 (W. A. Joye & E. Mandel 2003), and Ginga (E. Jeschke et al. 2013, via Astrowidgets).

14. DATA ANALYSIS

14.0.1. *Analysis Tools*

The `analysis_tools` package provides a framework to allow reproducible, automatic creation of plots and metrics through a set of configurable, reusable tools that can be used in pipeline execution and interactive analysis. The package allows metrics and plots to be consistently created at various points in the pipeline and ensures that the metrics dispatched to the monitoring dashboard (better word?) are generated in sync with the archived plots. The package was designed to handle the large data volumes and memory requirements that the survey will generate to ensure that the initial QA

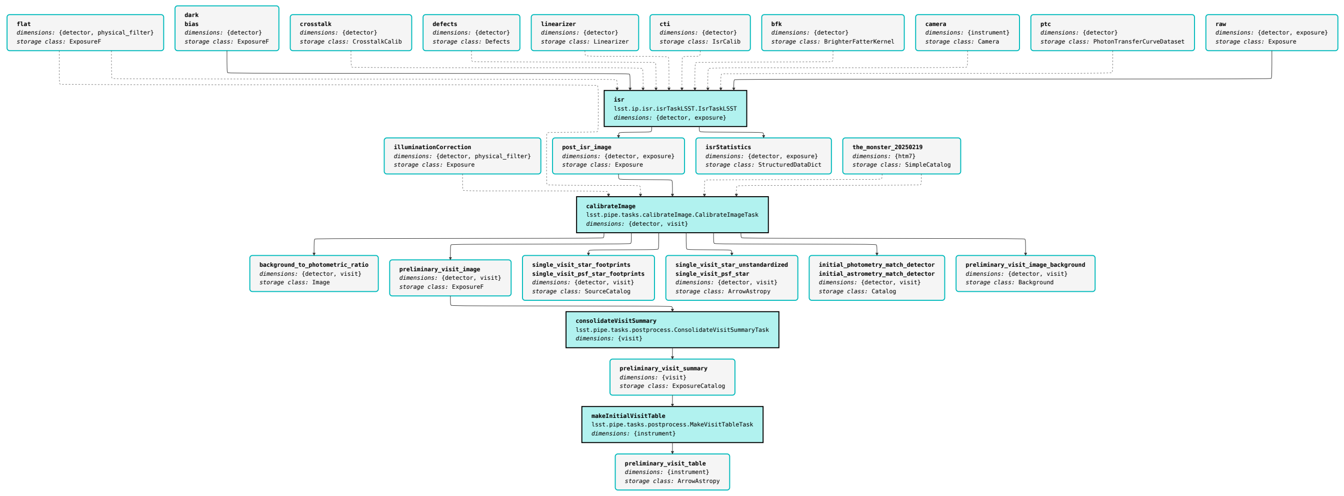
flat
dimensions: {detector, physical_filter}
storage class: ExposureF
dark
bias
dimensions: {detector}
storage class: ExposureF
crosstalk
dimensions: {detector}
storage class: CrosstalkCalib
defects
dimensions: {detector}
storage class: Defects
linearizer
dimensions: {detector}
storage class: Linearizer
cti
dimensions: {detector}
storage class: IsrCalib
bfk
dimensions: {detector}
storage class: BrighterFatterKernel
camera
dimensions: {instrument}
storage class: Camera
ptc
dimensions: {detector}
storage class: PhotonTransferCurveDataset
raw
dimensions: {detector, exposure}
storage class: Exposure
isr
lsst.ip.isr.isrTaskLSST.IsrTaskLSST
dimensions: {detector, exposure}
illuminationCorrection
dimensions: {detector, physical_filter}
storage class: Exposure
post_isr_image
dimensions: {detector, exposure}
storage class: Exposure
isrStatistics
dimensions: {detector, exposure}
storage class: StructuredDataDict
the_monster_20250219
dimensions: {htm7}
storage class: SimpleCatalog
calibrateImage
lsst.pipe.tasks.calibrateImage.CalibrateImageTask
dimensions: {detector, visit}
background_to_photometric_ratio
dimensions: {detector, visit}
storage class: Image
preliminary_visit_image
dimensions: {detector, visit}
storage class: ExposureF
single_visit_star_footprints
single_visit_psf_star_footprints
dimensions: {detector, visit}
storage class: SourceCatalog
single_visit_star_unstandardized
single_visit_psf_star
dimensions: {detector, visit}
storage class: ArrowAstropy
initial_photometry_match_detector
initial_astrometry_match_detector
dimensions: {detector, visit}
storage class: Catalog
preliminary_visit_image_background
dimensions: {detector, visit}
storage class: Background
consolidateVisitSummary
lsst.pipe.tasks.postprocess.ConsolidateVisitSummaryTask
dimensions: {visit}
preliminary_visit_summary
dimensions: {visit}
storage class: ExposureCatalog
makeInitialVisitTable
lsst.pipe.tasks.postprocess.MakeVisitTableTask
dimensions: {instrument}
preliminary_visit_table
dimensions: {instrument}
storage class: ArrowAstropy

Figure 5. Example pipeline visualization of four selected tasks from the `LSSTComCam/DRP-v2.yaml` pipeline in the Mermaid format. The diagram illustrates the flow of datasets between tasks, with dashed lines indicating prerequisite inputs. This visualization helps validate task dependencies and the expected sequence of execution.

products required are rapidly made and readily available for fast action on any emergent data quality issues. The individual tools run in the pipelines to calculate the metrics can then be reused in an interactive environment, such as a script or notebook, allowing further investigation into arising issues to reproduce exactly what was originally run.

14.0.2. *Verification*

`verify`

`faro` — **do not document this as we are no longer using it for primary metrics calculation**.

15. VALIDATING THE SCIENCE PIPELINES

We use small, of order of a few gigabyte, datasets that can be processed as part of continuous integration. These take of order an hour to process. There are regular re-processings of standard datasets that can take a few days to process. For formal data releases there are additional metrics calculated and a test report is issued, such as the one made available with release 28.0 (J. Carlin 2025).

15.1. *Source Injection*

The `source_injection` package contains tools designed to assist in the injection of synthetic sources into scientific imaging. Source injection is a powerful tool for testing the algorithmic performance of the LSST Science Pipelines, generating measurements on synthetic sources where the truth is known and facilitating subsequent quality assurance checks. Synthetic source generation and injection capability is provided by the GalSim software package (B. T. P. Rowe et al. 2015). An example showcasing the injection of a series of synthetic Sérsic sources into an HSC i-band image is shown in Figure 6.

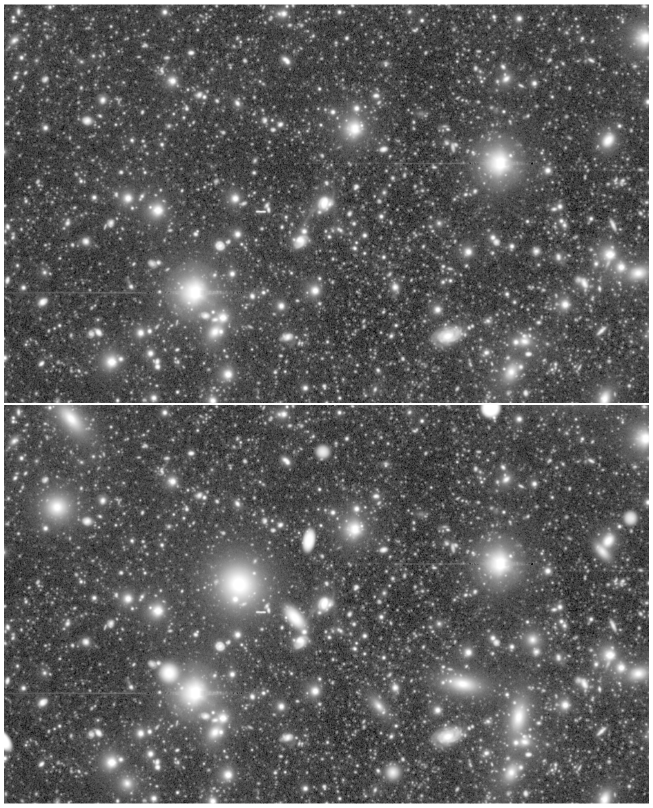

Figure 6. An HSC i-band cutout from tract 9813, patch 42, showing before (top) and after (bottom) the injection of a series of synthetic Sérsic sources. Images are 100 arcseconds on the short axis, log scaled across the central 99.5% flux range, and smoothed with a Gaussian kernel of FWHM 3 pixels.

Synthetic sources can be injected into any imaging data product output by the LSST Science Pipelines, including visit-level exposure-type or visit-type datasets (i.e., datasets with the dimension `exposure` or `visit`), or into a coadd-level coadded dataset. These injection tasks are defined in `ExposureInjectTask`, `VisitInjectTask` and `CoaddInjectTask`, respectively. Each task operates similarly: read in an injection catalog containing the parameters of the sources to be injected, generate sources using GalSim, and inject them into the input image. An additonal mask plane (`INJECTED` by default) is appended to the image mask to identify pixels which have been touched by injected sources. Optional modifications to the noise profiles of injected sources and the variance plane of the image can also be performed.

With GalSim we have the capacity to generate synthetic sources of varying profile types, including Gaussian, exponential and Sérsic profiles (J. L. Sérsic 1963, 1968), each convolved with the local PSF. We also have the option to inject scaled versions of the PSF model itself in order to simulate stars. If preferred, a pregenerated FITS image of a source can be injected instead of a model generated by GalSim, allowing for the injection of complex sources or postage stamp cutouts of real data.

Alongside the primary injection tasks, a suite of helper tools are also provided to optionally assist in the generation of synthetic source catalogs and injection pipelines. Fully qualified source injection pipeline definition YAML files are normally constructed using an existing pipeline as a baseline reference. A user specifies which dataset type they would like to inject synthetic sources into, and the `source_injection` package generates a new pipeline definition YAML file that includes the correctly configured source injection task. By default, all tasks in the pipeline downstream of the point at which source injection occurs are modified such that their connection names are prefixed with `injected_`. This ensures that an injected dataset is not confused with the original dataset when stored together in a common collection.

Once source injection has completed, the source injection task will output two dataset types: an injected image, and an associated injected catalog. The injected image is a copy of the original image with the injected sources added. The injected catalog is a catalog of the injected sources, with the same schema as the original catalog and additional columns describing per-source source injection success outcomes.

16. CONCLUSIONS

The LSST Science Pipelines Software has been developed over 20 years to support the processing of the Legacy Survey of Space and Time.

ACKNOWLEDGMENTS

This material is based upon work supported in part by the National Science Foundation through Cooperative Agreement AST-1258333 and Cooperative Support Agreement AST-1202910 managed by the Association of Universities for Research in Astronomy (AURA), and the Department of Energy under Contract No. DE-AC02-76SF00515 with the SLAC National Accelerator Laboratory managed by Stanford University. Additional Rubin Observatory funding comes from private donations, grants to universities, and in-kind support from LSSTC Institutional Members.

Facilities: Rubin:Simonyi (LSSTCam), Rubin:1.2m (LATISS)

Software: ndarray (https://github.com/ndarray/ndarray), astropy (Astropy Collaboration et al. 2022), pytest (H. Krekel 2017), matplotlib (J. D. Hunter 2007), galsim (B. T. P. Rowe et al. 2015), numpy (C. R. Harris et al. 2020), gbdes (G. M. Bernstein 2022), Starlink's (D. Berry et al. 2022) AST (D. S. Berry et al. 2016), fgcm (https://github.com/erykoff/fgcm),

REFERENCES

Alard, C., & Lupton, R. H. 1998, A Method for Optimal Image Subtraction, ApJ, 503, 325, doi: 10.1086/305984

AlSayyad, Y. 2019, Coaddition Artifact Rejection and CompareWarp, Data Management Technical Note DMTN-080, Vera C. Rubin Observatory. https://dmtn-080.lsst.io/

Astropy Collaboration, Price-Whelan, A. M., Sipőcz, B. M., et al. 2018, The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package, AJ, 156, 123, doi: 10.3847/1538-3881/aabc4f

Astropy Collaboration, Price-Whelan, A. M., Lim, P. L., et al. 2022, The Astropy Project: Sustaining and Growing a Community-oriented Open-source Project and the Latest Major Release (v5.0) of the Core Package, ApJ, 935, 167, doi: 10.3847/1538-4357/ac7c74

Axelrod, T., Connolly, A., Ivezic, Z., et al. 2004, The LSST Data Processing Pipeline, in American Astronomical Society Meeting Abstracts, Vol. 205, American Astronomical Society Meeting Abstracts, 108.11

Axelrod, T., Kantor, J., Lupton, R. H., & Pierfederici, F. 2010, An open source application framework for astronomical imaging pipelines, in Proc. SPIE, Vol. 7740, Software and Cyberinfrastructure for Astronomy, ed. N. M. Radziwill & A. Bridger, 15, doi: 10.1117/12.857297

Berk, A., Anderson, G. P., Bernstein, L. S., et al. 1999, MODTRAN4 radiative transfer modeling for atmospheric correction, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 3756, Optical Spectroscopic Techniques and Instrumentation for Atmospheric and Space Research III, ed. A. M. Larar, International Society for Optics and Photonics (SPIE), 348 – 353, doi: 10.1117/12.366388

Bernstein, G. M. 2022, gbdes: DECam instrumental signature fitting and processing programs„ Astrophysics Source Code Library, record ascl:2210.011

Bernstein, G. M., Armstrong, R., Plazas, A. A., et al. 2017, Astrometric Calibration and Performance of the Dark Energy Camera, PASP, 129, 074503, doi: 10.1088/1538-3873/aa6c55

Berry, D., Graves, S., Bell, G. S., et al. 2022, Starlink - The Original and Best, in Astronomical Society of the Pacific Conference Series, Vol. 532, Astronomical Data Analysis Software and Systems XXX, ed. J. E. Ruiz, F. Pierfedereci, & P. Teuben, 559

Berry, D. S., Warren-Smith, R. F., & Jenness, T. 2016, AST: A library for modelling and manipulating coordinate systems, Astronomy and Computing, 15, 33, doi: 10.1016/j.ascom.2016.02.003

Bertin, E. 2011, Automated Morphometry with SExtractor and PSFEx, in Astronomical Society of the Pacific Conference Series, Vol. 442, Astronomical Data Analysis Software and Systems XX, ed. I. N. Evans, A. Accomazzi, D. J. Mink, & A. H. Rots, 435

Bloom, J. S., Richards, J. W., Nugent, P. E., et al. 2012, Automating Discovery and Classification of Transients and Variable Stars in the Synoptic Survey Era, PASP, 124, 1175, doi: 10.1086/668468

Bohlin, R. C. 2007, HST Stellar Standards with 1% Accuracy in Absolute Flux, in Astronomical Society of the Pacific Conference Series, Vol. 364, The Future of Photometric, Spectrophotometric and Polarimetric Standardization, ed. C. Sterken, 315, doi: 10.48550/arXiv.astro-ph/0608715

Bosch, J. 2016, Flavors of Coadds, Data Management Technical Note DMTN-015, Vera C. Rubin Observatory. https://dmtn-015.lsst.io/

Bosch, J., Armstrong, R., Bickerton, S., et al. 2018, The Hyper Suprime-Cam software pipeline, PASJ, 70, S5, doi: 10.1093/pasj/psx080

Boulade, O., Charlot, X., Abbon, P., et al. 2003, MegaCam: the new Canada-France-Hawaii Telescope wide-field imaging camera, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 4841, Instrument Design and Performance for Optical/Infrared Ground-based Telescopes, ed. M. Iye & A. F. M. Moorwood, 72–81, doi: 10.1117/12.459890

Broughton, A., Utsumi, Y., Plazas Malagón, A. A., et al. 2024, Mitigation of the Brighter-fatter Effect in the LSST Camera, PASP, 136, 045003, doi: 10.1088/1538-3873/ad3aa2

Burke, D. L., Rykoff, E. S., Allam, S., et al. 2018, Forward Global Photometric Calibration of the Dark Energy Survey, AJ, 155, 41, doi: 10.3847/1538-3881/aa9f22

Cai, M., Xu, Z., Fan, L., et al. 2025, The 2.5-meter Wide Field Survey Telescope Real-time Data Processing Pipeline I: From raw data to alert distribution, arXiv e-prints, arXiv:2501.15018, doi: 10.48550/arXiv.2501.15018

Carlin, J. 2025, Characterization Metric Report: Science Pipelines Version 28.0.0, Data Management Test Report DMTR-451, Vera C. Rubin Observatory. https://dmtr-451.lsst.io/

DePoy, D. L., Abbott, T., Annis, J., et al. 2008, The Dark Energy Camera (DECam), in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 7014, Ground-based and Airborne Instrumentation for Astronomy II, ed. I. S. McLean & M. M. Casali, 70140E, doi: 10.1117/12.789466

Duev, D. A., Mahabal, A., Masci, F. J., et al. 2019, Real-bogus classification for the Zwicky Transient Facility using deep learning, MNRAS, 489, 3582, doi: 10.1093/mnras/stz2357

Esteves, J. H., Utsumi, Y., Snyder, A., et al. 2023, Photometry, Centroid and Point-spread Function Measurements in the LSST Camera Focal Plane Using Artificial Stars, PASP, 135, 115003, doi: 10.1088/1538-3873/ad0a73

Fagrelius, P., & Rykoff, E. 2025, Rubin Baseline Calibration Plan, Commissioning Technical Note SITCOMTN-086, Vera C. Rubin Observatory. https://sitcomtn-086.lsst.io/

Fausti, A. 2023, Sasquatch: beyond the EFD, SQuaRE Technical Note SQR-068, Vera C. Rubin Observatory. https://sqr-068.lsst.io/

Fausti Neto, A., Economou, F., Reuter, M. A., et al. 2024, Sasquatch: Rubin Observatory metrics and telemetry service, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 13101, Software and Cyberinfrastructure for Astronomy VIII, ed. J. Ibsen & G. Chiozzi, 131011M, doi: 10.1117/12.3019081

Flaugher, B., Diehl, H. T., Honscheid, K., et al. 2015, The Dark Energy Camera, The Astronomical Journal, 150, 150, doi: 10.1088/0004-6256/150/5/150

Goldstein, D. A., D'Andrea, C. B., Fischer, J. A., et al. 2015, Automated Transient Identification in the Dark Energy Survey, AJ, 150, 82, doi: 10.1088/0004-6256/150/3/82

Gower, M., Kowalik, M., Lust, N. B., Bosch, J. F., & Jenness, T. 2022, Adding Workflow Management Flexibility to LSST Pipelines Execution, arXiv e-prints, arXiv:2211.15795, doi: 10.48550/arXiv.2211.15795

Harris, C. R., Millman, K. J., van der Walt, S. J., et al. 2020, Array programming with NumPy, Nature, 585, 357, doi: 10.1038/s41586-020-2649-2

Hirata, C., & Seljak, U. 2003, Shear calibration biases in weak-lensing surveys, MNRAS, 343, 459, doi: 10.1046/j.1365-8711.2003.06683.x

Hunter, J. D. 2007, Matplotlib: A 2D Graphics Environment, Computing in Science and Engineering, 9, 90, doi: 10.1109/MCSE.2007.55

Ingraham, P., Clements, A. W., Ribeiro, T., et al. 2020, Vera C. Rubin Observatory auxiliary telescope commissioning as a control system pathfinder, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 11452, Software and Cyberinfrastructure for Astronomy VI, ed. J. C. Guzman & J. Ibsen, 114520U, doi: 10.1117/12.2561112

Ivezić, Ž., Kahn, S. M., Tyson, J. A., et al. 2019, LSST: From Science Drivers to Reference Design and Anticipated Data Products, ApJ, 873, 111, doi: 10.3847/1538-4357/ab042c

Jarvis, M., Meyers, J., Leget, P.-F., & Davis, C. 2021a, Piff: PSFs In the Full FOV" Astrophysics Source Code Library, record ascl:2102.024

Jarvis, M., Bernstein, G. M., Amon, A., et al. 2021b, Dark Energy Survey year 3 results: point spread function modelling, MNRAS, 501, 1282, doi: 10.1093/mnras/staa3679

Jenness, T. 2020, Modern Python at the Large Synoptic Survey Telescope, in Astronomical Society of the Pacific Conference Series, Vol. 522, Astronomical Data Analysis Software and Systems XXVII, ed. P. Ballester, J. Ibsen, M. Solar, & K. Shortridge, 541, doi: 10.48550/arXiv.1712.00461

Jenness, T., Economou, F., Findeisen, K., et al. 2018, LSST data management software development practices and tools, in Proc. SPIE, Vol. 10707, Software and Cyberinfrastructure for Astronomy V, 1070709, doi: 10.1117/12.2312157

Jenness, T., Bosch, J., Owen, R., et al. 2016, Investigating interoperability of the LSST data management software stack with Astropy, in Proc. SPIE, Vol. 9913, Software and Cyberinfrastructure for Astronomy IV, 99130G, doi: 10.1117/12.2231313

Jenness, T., Bosch, J. F., Salnikov, A., et al. 2022, The Vera C. Rubin Observatory Data Butler and pipeline execution system, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 12189, Software and Cyberinfrastructure for Astronomy VII, 1218911, doi: 10.1117/12.2629569

Jeschke, E., Inagaki, T., & Kackley, R. 2013, Introducing the Ginga FITS Viewer and Toolkit, in Astronomical Society of the Pacific Conference Series, Vol. 475, Astronomical Data Analysis Software and Systems XXII, ed. D. N. Friedel, 319

Joye, W. A., & Mandel, E. 2003, New Features of SAOImage DS9, in Astronomical Society of the Pacific Conference Series, Vol. 295, Astronomical Data Analysis Software and Systems XII, ed. H. E. Payne, R. I. Jedrzejewski, & R. N. Hook, 489

Jurić, M., Ciardi, D., Dubois-Felsmann, G., & Guy, L. 2019, LSST Science Platform Vision Document, Systems Engineering Controlled Document LSE-319, Vera C. Rubin Observatory. https://lse-319.lsst.io/

Jurić, M., Kantor, J., Lim, K. T., et al. 2017, The LSST Data Management System, in Astronomical Society of the Pacific Conference Series, Vol. 512, Astronomical Data Analysis Software and Systems XXV, ed. N. P. F. Lorente, K. Shortridge, & R. Wayth, 279, doi: 10.48550/arXiv.1512.07914

Jurić, M., Axelrod, T., Becker, A., et al. 2023, Data Products Definition Document, Systems Engineering Controlled Document LSE-163, Vera C. Rubin Observatory. https://lse-163.lsst.io/

Kahn, S. M., Kurita, N., Gilmore, K., et al. 2010, Design and development of the 3.2 gigapixel camera for the Large Synoptic Survey Telescope, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 7735, Ground-based and Airborne Instrumentation for Astronomy III, ed. I. S. McLean, S. K. Ramsay, & H. Takami, 0, doi: 10.1117/12.857920

Kannawadi, A. 2022, Consistent galaxy colors with Gaussian-Aperture and PSF photometry, Data Management Technical Note DMTN-190, Vera C. Rubin Observatory. https://dmtn-190.lsst.io/

Karavakis, E., Guan, W., Yang, Z., et al. 2024, Integrating the PanDA Workload Management System with the Vera C. Rubin Observatory, in European Physical Journal Web of Conferences, Vol. 295, European Physical Journal Web of Conferences (EDP), 04026, doi: 10.1051/epjconf/202429504026

Knight, S. 2005, Building software with SCons, Computing in Science Engineering, 7, 79, doi: 10.1109/MCSE.2005.11

Krekel, H. 2017, pytest: helps you write better programs, https://docs.pytest.org

Kuijken, K. 2008, GaaP: PSF- and aperture-matched photometry using shapelets, A&A, 482, 1053, doi: 10.1051/0004-6361:20066601

Kuijken, K., Heymans, C., Hildebrandt, H., et al. 2015, Gravitational lensing analysis of the Kilo-Degree Survey, MNRAS, 454, 3500, doi: 10.1093/mnras/stv2140

Labrie, K., Simpson, C., Cardenes, R., et al. 2023, DRAGONS-A Quick Overview, Research Notes of the American Astronomical Society, 7, 214, doi: 10.3847/2515-5172/ad0044

Lange, T., Nordby, M., Pollek, H., et al. 2024, Integrating the LSST camera, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 13096, Ground-based and Airborne Instrumentation for Astronomy X, ed. J. J. Bryant, K. Motohara, & J. R. D. Vernet, 130961O, doi: 10.1117/12.3019302

Li, X., Miyatake, H., Luo, W., et al. 2022, The three-year shear catalog of the Subaru Hyper Suprime-Cam SSP Survey, PASJ, 74, 421, doi: 10.1093/pasj/psac006

Lim, K.-T. 2022, Proposal and Prototype for Prompt Processing, Data Management Technical Note DMTN-219, Vera C. Rubin Observatory. https://dmtn-219.lsst.io/

Lupton, R., Malagón, A. A. P., & Waters, C. 2025, Verifying LSST Calibration Data Products, Data Management Technical Note DMTN-101, Vera C. Rubin Observatory. https://dmtn-101.lsst.io/

Lust, N. B., Jenness, T., Bosch, J. F., et al. 2023, Data management and execution systems for the Rubin Observatory Science Pipelines, arXiv e-prints, arXiv:2303.03313, doi: 10.48550/arXiv.2303.03313

Mandelbaum, R., Hirata, C. M., Seljak, U., et al. 2005, Systematic errors in weak lensing: application to SDSS galaxy-galaxy weak lensing, MNRAS, 361, 1287, doi: 10.1111/j.1365-2966.2005.09282.x

Mandelbaum, R., Miyatake, H., Hamana, T., et al. 2018, The first-year shear catalog of the Subaru Hyper Suprime-Cam Subaru Strategic Program Survey, PASJ, 70, S25, doi: 10.1093/pasj/psx130

McCormick, J., Dubois-Felsmann, G. P., Salnikov, A., Van Klaveren, B., & Jenness, T. 2024, Using Felis to Represent the Semantics and Metadata of Astronomical Data Catalogs, arXiv e-prints, arXiv:2412.09721, doi: 10.48550/arXiv.2412.09721

Melchior, P., Joseph, R., & Moolekamp, F. 2019, Proximal Adam: Robust Adaptive Update Scheme for Constrained Optimization, arXiv e-prints, arXiv:1910.10094, doi: 10.48550/arXiv.1910.10094

Melchior, P., Moolekamp, F., Jerdee, M., et al. 2018, SCARLET: Source separation in multi-band images by Constrained Matrix Factorization, Astronomy and Computing, 24, 129, doi: 10.1016/j.ascom.2018.07.001

Miyazaki, S., Komiyama, Y., Kawanomoto, S., et al. 2018, Hyper Suprime-Cam: System design and verification of image quality, PASJ, 70, S1, doi: 10.1093/pasj/psx063

Morrison, C. B. 2018, Pessimistic Pattern Matching for LSST, Data Management Technical Note DMTN-031, Vera C. Rubin Observatory. https://dmtn-031.lsst.io/

Mueller, F., et al. 2023, Qserv: A Distributed Petascale Database for the LSST Catalogs, in ASP Conf. Ser., Vol. TBD, ADASS XXXII, ed. S. Gaudet, S. Gwyn, P. Dowler, D. Bohlender, & A. Hincks (San Francisco: ASP), in press. https://dmtn-243.lsst.io

Mullaney, J. R., Makrygianni, L., Dhillon, V., et al. 2021, Processing GOTO data with the Rubin Observatory LSST Science Pipelines I: Production of coadded frames, PASA, 38, e004, doi: 10.1017/pasa.2020.45

of Washington), J. P. U., & Paris), P. A. L. 2018, jointcal: Simultaneous Astrometry & Photometry for thousands of Exposures with Large CCD Mosaics, Data Management Technical Note DMTN-036, Vera C. Rubin Observatory. https://dmtn-036.lsst.io/

Okura, Y., Petri, A., May, M., Plazas, A. A., & Tamagawa, T. 2016, Consequences of CCD Imperfections for Cosmology Determined by Weak Lensing Surveys: From Laboratory Measurements to Cosmological Parameter Bias, The Astrophysical Journal, 825, 61, doi: 10.3847/0004-637X/825/1/61

Okura, Y., Plazas, A. A., May, M., & Tamagawa, T. 2015, Spurious shear induced by the tree rings of the LSST CCDs, Journal of Instrumentation, 10, C08010, doi: 10.1088/1748-0221/10/08/C08010

O'Mullane, W., Economou, F., Lim, K.-T., et al. 2022, Software Architecture and System Design of Rubin Observatory, arXiv e-prints, arXiv:2211.13611, doi: 10.48550/arXiv.2211.13611

O'Mullane, W., Economou, F., Huang, F., et al. 2024, Rubin Science Platform on Google: the story so far, in Astronomical Society of the Pacific Conference Series, Vol. 535, Astromical Data Analysis Software and Systems XXXI, ed. B. V. Hugo, R. Van Rooyen, & O. M. Smirnov, 227, doi: 10.48550/arXiv.2111.15030

O'Mullane, W., Allbery, R., AlSayyad, Y., et al. 2024, Rubin Observatory Data Security Standards Implementation, Data Management Technical Note DMTN-199, Vera C. Rubin Observatory. https://dmtn-199.lsst.io/

Padmanabhan, N., Lupton, R., & Loomis, C. 2015, EUPS — a Tool to Manage Software Dependencies, https://github.com/RobertLuptonTheGood/eups

Park, H., Karpov, S., Nomerotski, A., & Tsybychev, D. 2020, Tree rings in Large Synoptic Survey Telescope production sensors: its dependence on radius, wavelength, and back bias voltage, Journal of Astronomical Telescopes, Instruments, and Systems, 6, 011005, doi: 10.1117/1.JATIS.6.1.011005

Park, H. Y., Nomerotski, A., & Tsybychev, D. 2017, Properties of tree rings in LSST sensors, Journal of Instrumentation, 12, C05015, doi: 10.1088/1748-0221/12/05/C05015

Paszke, A., Gross, S., Massa, F., et al. 2019, PyTorch: An Imperative Style, High-Performance Deep Learning Library, in Advances in Neural Information Processing Systems, ed. H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett, Vol. 32 (Curran Associates, Inc.), doi: 10.48550/arXiv.1912.01703

Patterson, M., Bellm, E., Swinbank, J., & Nelson, S. 2020, Design of the LSST Alert Distribution System, Data Management Technical Note DMTN-093, Vera C. Rubin Observatory. https://dmtn-093.lsst.io/

Pier, J. R., Munn, J. A., Hindsley, R. B., et al. 2003, Astrometric Calibration of the Sloan Digital Sky Survey, AJ, 125, 1559, doi: 10.1086/346138

Plazas Malagón, A. A., Waters, C., Broughton, A., et al. 2024, Instrument Signature Removal and Calibration Products for the Rubin Legacy Survey of Space and Time, arXiv e-prints, arXiv:2404.14516, doi: 10.48550/arXiv.2404.14516

Reiss, D. J., & Lupton, R. H. 2016, Implementation of Image Difference Decorrelation, Data Management Technical Note DMTN-021, Vera C. Rubin Observatory. https://dmtn-021.lsst.io/

Roby, W., Wu, X., Dubois–Felmann, G., et al. 2020, Firefly and Python — New Ways to Visualize Data on the Web, in Astronomical Society of the Pacific Conference Series, Vol. 527, Astronomical Data Analysis Software and Systems XXIX, ed. R. Pizzo, E. R. Deul, J. D. Mol, J. de Plaa, & H. Verkouter, 243

Roodman, A., Rasmussen, A., Bradshaw, A., et al. 2024, LSST camera verification testing and characterization, in Ground-based and Airborne Instrumentation for Astronomy X, ed. J. J. Bryant, K. Motohara, & J. R. D. Vernet, Vol. 13096, International Society for Optics and Photonics (SPIE), 130961S, doi: 10.1117/12.3019698

Rowe, B. T. P., Jarvis, M., Mandelbaum, R., et al. 2015, GALSIM: The modular galaxy image simulation toolkit, Astronomy and Computing, 10, 121, doi: 10.1016/j.ascom.2015.02.002

Salnikov, A., & McCormick, J. 2024, Current status of APDB and PPDB implementation, Data Management Technical Note DMTN-293, Vera C. Rubin Observatory. https://dmtn-293.lsst.io/

Saunders, C. 2024, Astrometric Calibration in the LSST Pipeline, Data Management Technical Note DMTN-266, Vera C. Rubin Observatory. https://dmtn-266.lsst.io/

Schutt, T., Jarvis, M., Roodman, A., et al. 2025, Dark Energy Survey Year 6 Results: Point-Spread Function Modeling, The Open Journal of Astrophysics, 8, 26, doi: 10.33232/001c.132299

Sérsic, J. L. 1963, Influence of the atmospheric and instrumental dispersion on the brightness distribution in a galaxy, Boletin de la Asociacion Argentina de Astronomia La Plata Argentina, 6, 41

Sérsic, J. L. 1968, Atlas de Galaxias Australes (Observatorio Astronomico, Universidad Nacional de Cordoba)

Shupe, D. L., Moshir, M., Li, J., et al. 2005, The SIP Convention for Representing Distortion in FITS Image Headers, in Astronomical Society of the Pacific Conference Series, Vol. 347, Astronomical Data Analysis Software and Systems XIV, ed. P. Shopbell, M. Britton, & R. Ebert, 491

Sutherland, W., Emerson, J., Dalton, G., et al. 2015, The Visible and Infrared Survey Telescope for Astronomy (VISTA): Design, technical overview, and performance, A&A, 575, A25, doi: 10.1051/0004-6361/201424973

Tabur, V. 2007, Fast Algorithms for Matching CCD Images to a Stellar Catalogue, PASA, 24, 189, doi: 10.1071/AS07028

Taranu, D. S. 2025, The MultiProFit astronomical source modelling code, Data Management Technical Note DMTN-312, Vera C. Rubin Observatory. https://dmtn-312.lsst.io/

Thomas, S. J., Barr, J., Callahan, S., et al. 2022, Rubin Observatory Simonyi Survey Telescope status overview, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 12182, Ground-based and Airborne Telescopes IX, ed. H. K. Marshall, J. Spyromilio, & T. Usuda, 121820W, doi: 10.1117/12.2630226

Utsumi, Y., Antilogus, P., Astier, P., et al. 2024, LSST Camera focal plane optimization, in X-Ray, Optical, and Infrared Detectors for Astronomy XI, ed. A. D. Holland & K. Minoglou, Vol. 13103, International Society for Optics and Photonics (SPIE), 131030W, doi: 10.1117/12.3019117

van Rossum, G. 2013, PEP 8 – Style Guide for Python Code, Python Software Foundation. https://www.python.org/dev/peps/pep-0008/

Wang, D. L., Monkewitz, S. M., Lim, K.-T., & Becla, J. 2011, Qserv: A Distributed Shared-nothing Database for the LSST Catalog, in State of the Practice Reports, SC '11 (New York, NY, USA: ACM), 12:1–12:11, doi: 10.1145/2063348.2063364

Wang, S.-Y., Huang, P.-J., Chen, H.-Y., et al. 2020, Prime Focus Spectrograph (PFS): the prime focus instrument, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 11447, Ground-based and Airborne Instrumentation for Astronomy VIII, ed. C. J. Evans, J. J. Bryant, & K. Motohara, 114477V, doi: 10.1117/12.2561194

Waters, C. 2025, Calibration Generation, Verification, Acceptance, and Certification., Data Management Technical Note DMTN-222, Vera C. Rubin Observatory. https://dmtn-222.lsst.io/

Waters, C. Z., Magnier, E. A., Price, P. A., et al. 2020, Pan-STARRS Pixel Processing: Detrending, Warping, Stacking, ApJS, 251, 4, doi: 10.3847/1538-4365/abb82b

Wright, A. H., Kuijken, K., Hildebrandt, H., et al. 2025, The fifth data release of the Kilo Degree Survey: Multi-epoch optical/NIR imaging covering wide and legacy-calibration fields, arXiv e-prints, arXiv:2503.19439. https://arxiv.org/abs/2503.19439

Zackay, B., Ofek, E. O., & Gal-Yam, A. 2016, Proper Image Subtraction—Optimal Transient Detection, Photometry, and Hypothesis Testing, ApJ, 830, 27, doi: 10.3847/0004-637X/830/1/27