Draft version September 29, 2025
Typeset using LaTeX **twocolumn** style in AASTeX7

https://doi.org/10.71929/rubin/2570545

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

Rubin Observatory Science Pipelines Developers[1]
The Rubin Observatory Science Pipelines Team

[1] *Vera C. Rubin Observatory Project Office, 950 N. Cherry Ave., Tucson, AZ 85719, 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 (SLAC National Accelerator Laboratory & NSF-DOE Vera C. Rubin Observatory 2025; A. Roodman et al. 2024) performing the main survey and the Rubin Auxiliary Telescope (P. Ingraham et al. 2020) providing supplementary atmospheric calibration data using the LATISS instrument (NSF-DOE Vera C. Rubin Observatory 2020). 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 the 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 the processing of data releases from Hyper Suprime-Cam on the Subaru Telescope in Hawaii (H. Aihara et al. 2022), along with additional instruments used for internal testing and from plugins developed by the community. It was used to make the first public data release from Rubin Observatory, the LSST Data Preview 1 (Vera C. Rubin Observatory Team 2025; NSF-DOE Vera C. Rubin Observatory 2025), which consisted of data from the LSST Commissioning Camera (LSSTComCam; SLAC National Accelerator Laboratory & NSF-DOE Vera C. Rubin Observatory 2024; B. Stalder et al. 2022).

In this paper we provide an overview of the software system, dividing it into four rough tiers:

- low-level utility code and our data access, configuration, and execution frameworks, and our core algorithmic primitives are described in sections 2, 3, and 4, respectively;
- reusable mid-level algorithmic components are described in 5;
- high-level tasks and pipelines are described in §6, along with some of the details of the algorithms specific to them;

- analysis and validation tooling is described in §7.

There are no sharp boundaries between these tiers; they are better considered to be a loose organizational aid than any kind of formal classification. We do not include details of the science validation of the individual algorithms, and do not attempt to cover components of the system in uniform detail; instead we focus attention on those that are unique or novel or not well described elsewhere. 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++20 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[2] 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. The naming convention is that if a package is named `some_name` then in Python it will be imported as `lsst.some.name`.

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.

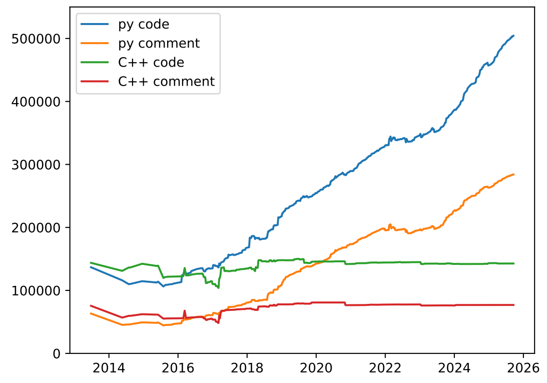

**Figure 1.** The number of lines of code comprising the LSST Science Pipelines software as a function of year. Line counts include comments but not blank lines. Python interfaces are implemented using `pybind11` and that is counted as C++ code. For the purposes of this count Science pipelines software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

As of September 2025, the Science Pipelines software is nearly 750,000 lines of Python and 220,000 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`[3]. 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] All repositories can be found at https://github.com/lsst and we do not give individual URLs to each package in this paper.

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

### 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 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 across multiple packages. This includes longer tests (up to a few hours) in which we run complete pipelines on small precursor datasets (typically a few GB) fetched via `git-lfs`.

#### 2.2.1. *Installation*

Where possible, individual packages can be installed from PyPI but that is mostly limited to packages that do not have any C++ code due to limitations in Python packaging allowing packages to link directly against shared libraries from other packages. To install the full set of pipelines software we provide an installation script that installs the appropriate Conda environment and then downloads the binaries (Linux x86_64, Linux aarch64, or macOS Apple Silicon) from our artifact server. As mentioned above it is also possible to install a Docker container with everything pre-installed. For developers we also provide a script to build and install the software directly from the Git repositories.

## 3. DATA ACCESS AND EXECUTION ABSTRACTIONS

The algorithmic components of the LSST Science Pipelines are built on a suite of packages that together form a powerful data access and execution framework (`pex_config`, `resources`, `daf_butler`, `pipe_base`, `ctrl_mpexec`, and `ctrl_bps`). Unlike most of the rest of the codebase, these packages can be individually installed with `pip` as well as EUPS and can be used on their own.

### 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).

**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. |

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.

Projects can use different sets of dimension definitions, called "universes", and the butler's database schema is derived from those definitions (so changing the dimension universe used by a butler repository is possible, but not undertaken lightly, as it constitutes a schema migration). Downstream science code generally needs to make assumptions about what is in the dimension universe (e.g. whether a `detector` concept exists, and if so, whether *implies* a `band` or can be associated with any `band`), which can limit its reusability.

## 3.2. *Pipelines and Tasks*

The data dimensions system also plays a fundamental role in how the LSST processing pipelines are assembled and run; high-level pieces of algorithmic code called `PipelineTasks` declare the dimensions of their units of work ("quanta"), their inputs, and their outputs, allowing a directed acyclic graph (a "quantum graph") describing the processing to be assembled from a YAML declaration of the tasks to be run, their configuration, and a Butler database query. Quantum graphs can range in size from a few tens of quanta (e.g., for the nightly processing performed on a single detector image) to millions (for a piece of the yearly data release pipelines), and serve as the common interface for multiple execution systems, including the low-latency nightly Prompt Processing framework (K.-T. Lim 2023) and the Batch Processing System (BPS; M. Gower et al. 2022), which adapts quantum graphs for execution at scale by third-party workflow management systems like HTCondor (HTCondor Team 2024), Parsl (Y. Babuji et al. 2019), and PanDA (E. Karavakis et al. 2024).

Algorithmic code below the `PipelineTask` level is often subdivided into multiple "subtasks" that (like `PipelineTask` itself) inherit from the base `Task` class, which provides easy access to hierarchical logging, metadata, and configuration.

## 3.3. *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, we provide several options for visualizing the pipeline graph (a simplified directed acyclic graph that shows how tasks relate to dataset types but without including data from a Butler query).

Diagrams can be generated as ASCII, useful for a quick inspection from a terminal session, or in graphical formats such as Graphviz DOT[4] or Mermaid[5]. The Mermaid format is particularly well-suited for sharing in accessible, web-based contexts. Figure 2 shows a visualization of a subset of two tasks from the `LSSTComCam/DRP-v2.yaml` pipeline. The diagram shows the relationships between tasks and their input and output dataset types 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 quantum graphs or failed pipeline execution. It is also possible to visualize quantum graphs, which include the actual data from a Butler query, but these can be very large and are only useful for small Butler queries to inspect the actual data flow when debugging.

## 3.4. *Configuration*

The `pex_config` package provides the foundational configuration system for running large scale processing campaigns with the science pipelines. `pex_config` is built resembling a Domain Specific Language embedded within Python, in which configuration options are defined via Python classes and values are set via snippets of Python code. This leverages the full power of a programming language for parsing and setting configuration values, while also supporting interoperability with standard text formats (e.g. JSON, YAML), import-time plugin extensibility, provenance through parameter change history, and validation of both types and values. The system can be installed and used on its own, and been adopted by the DRAGONS software (K. Labrie et al. 2023). A more detailed description of how the configuration system works is provided in Appendix A.

## 3.5. *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:

[4] https://www.graphviz.org
[5] https://mermaid.js.org

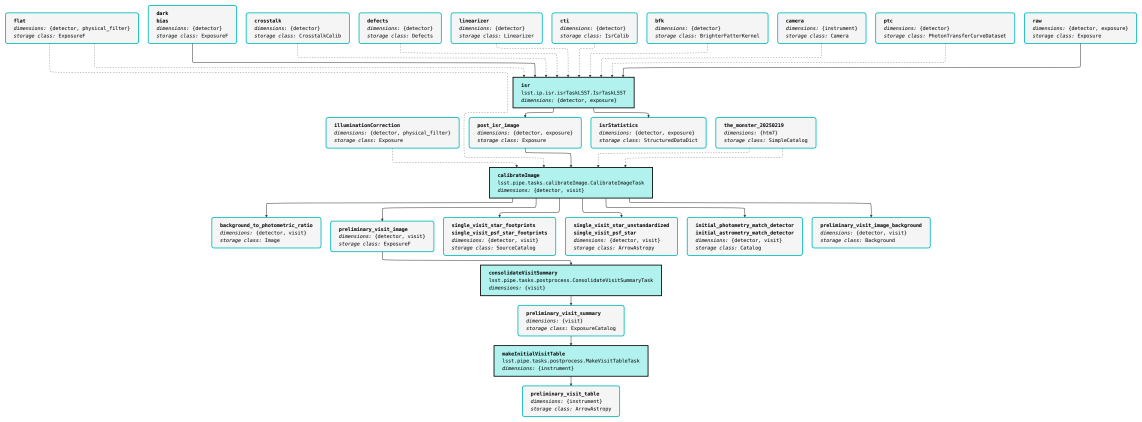
flat
dimensions: {detector, physical_filter}
storage class: ExposureF
dark
bias
dimensions: {detector}
storage class: ExposureF
crosstalk
dimensions: {detector}
storage class: CrosstalkCalib
defects
dimensions: {detector}
storage class: Defects
linearizer
dimensions: {detector}
storage class: Linearizer
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 2.** Example pipeline visualization of four selected tasks from the `LSSTComCam/DRP-v2.yaml` pipeline generated using Mermaid. 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.

- 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.

The `Instrument` interface is defined in two levels: the minimal interface in the `pipe_base` package defines everything needed to use the Butler and execution system, while a more complete subclass in the `obs_base` package provides considerable additional functionality but is not in the minimal, `pip`-installable suite because of the additional dependencies on C++ code.

By convention we define the instrument class and associated configuration in `obs` packages. There are currently project-supported `obs` packages for:

- LSSTCam (SLAC National Accelerator Laboratory & NSF-DOE Vera C. Rubin Observatory 2025; A. Roodman et al. 2024; S. M. Kahn et al. 2010), LATISS (NSF-DOE Vera C. Rubin Observatory 2020; P. Ingraham et al. 2020), and associated Rubin Observatory test stands, calibration instruments, and simulators.
- Hyper-SuprimeCam on the Subaru telescope (S. Miyazaki et al. 2018).
- The Dark Energy Camera on the CTIO Blanco telescope (B. Flaugher et al. 2015; D. L. DePoy et al. 2008).
- CFHT's MegaPrime (O. Boulade et al. 2003).

All of these instruments use the default dimension universe.

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). SPHEREx (B. P. Crill et al. 2020) uses the pipeline middleware but has developed pipeline tasks without the Rubin C++ dependencies and so does not use an `obs` package. Most of these projects have made small changes to the default dimension universe to better represent their data model.

## 3.6. *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.[6] 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

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

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 ALGORITHMIC PRIMITIVES AND DATA STRUCTURES

The high-level algorithms in the LSST Science Pipelines are largely built from algorithmic primitives and data structures implemented in the `afw` package. `afw` is a large, complex, C++-heavy suite of multiple libraries that sometimes suffer from historical idiosyncrasies, but are nevertheless extremely powerful and well optimized. These include (but are not limited to):

- `afw.geom` holds our high-level geometry primitives. This includes composable coordinate transforms and world coordinate systems (WCS). It also includes `SpanSet`, a run-length encoding (RLE) description of a set of pixels in a 2-d image, a simple `Polygon` class, and routines for working with various ellipse parameterizations.

- `afw.math` includes convolution, resampling, general-purpose interpolation, statistics, and least-squares fitting algorithms.

- `afw.detection` contains threshold-based detection on images and point-spread function (PSF) model interfaces. This includes the `Footprint` class, which combines a `SpanSet` with a list of peaks to represent either a single source detection or a group of blended sources.

- `afw.image` centers around the `Exposure` class, which combines an image (`Image`), bitmask (`Mask`), and variance image with the many objects used to astrophysically characterize an observation or coadd (PSF, WCS, aperture corrections, etc).

- `afw.table` holds data structures for tabular data, with both row- and column-based views. The `afw.table.io` package defines a framework for persisting arbitrary objects to a series of FITS binary table HDUs, which is used in the on-disk form of the `Exposure` class.

- `afw.cameraGeom` provides a hierarchical description of large-format photometric cameras (like LSSTCam, HSC, or DECam), including optical distortions, focal plane layouts, and amplifier regions.

Most of the algorithms and data structures in `afw` are implemented there directly, and often represent evolutions of concepts first developed in the SDSS *Photo* pipeline or the Pan-STARRS Image Processing Pipelines. Others delegate to third-party libraries, particularly Eigen (linear algebra), the Gnu Scientific Library (interpolation, random numbers), Boost (image iterators and geometry), CFITSIO (image and table I/O) and Starlink AST (coordinate systems and transforms; D. S. Berry et al. 2016).

Even lower-level data structures are defined in a handful of packages just below `afw`. The `sphgeom` package is used for spherical geometry calculations, sky-based regions, and hierarchical sky pixelization schemes, while `geom` provides simple 2-d Euclidean `Point`, `Extent`, and `Box` types in both integer- and floating-point variants. `geom` also includes linear transforms and (for historical reasons) its own angle-manipulation and sky coordinate type.

C++ mapping types (with Python bindings) and date/time objects are defined in `daf_base`. The `DateTime` package is used in our C++ data structures mostly to represent TAI times. 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.

Another small set of core packages sits just above `afw`:

- `skymap` defines interfaces and a few implementations of our system for mapping the sky onto a set of slightly-overlapping image-friendly projections and tiles for coaddition. Each distinct projection in a skymap is called a `tract`, and each `tract` is further divided into multiple `patches`.

- `shapelet` includes optimizated evaluation of Gauss-Hermite and Gauss-Laguerre functions and their derivatives.

A substantial fraction of our core packages predate the now-ubiquitous Astropy package, and in some cases we now prefer to use Astropy types in *most* new code (date/time representations and tables in particular) and especially public interfaces, following the recommendations from T. Jenness et al. (2016). Fully retiring core libraries that have Astropy counterparts is at best a long-term project, however, due to our continued need for these objects in considerable amounts of C++ that has no equivalent in Astropy (or anywhere else).

## 5. KEY ALGORITHMIC COMPONENTS

Most of the key algorithms we have implemented for processing Rubin data are used in multiple pipelines, and in many cases an algorithmic component is used multiple times within a single pipeline. For the most part, these reusable algorithms are implemented as regular `Task` objects, and these are combined in higher-level `PipelineTasks` discussed later in §6. In other cases, the algorithm is inseparable from its I/O, and is implemented directly as a `PipelineTask`. This section includes examples of both.

### 5.1. *Instrument Signature Removal*

Raw images from charge-coupled devices (CCDs) contain instrumental effects, such as dark currents, tree-rings (H. Park et al. 2020), brighter-fatter (A. Broughton et al. 2024), amplifier offsets, 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 a critical package for pipelines used to process LSST images and requires calibration products produced and verified by pipelines from the `cp_pipe` and `cp_verify` packages as described in §6.5. For further information about the life cycle of a calibration product and the procedures it entails, see C. Z. Waters & E. S. Rykoff (2025) and P. Fagrelius & E. S. Rykoff (2025). 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. A general overview of the ISR steps and calibration products production (including generation, verification, certification, approval, and distribution) is given in A. A. Plazas Malagón et al. (2025).

### 5.2. *Background Subtraction*

Our low-level background subtraction algorithms (in the `meas_algorithms` package) operate by binning images, masking out detections and bad pixels, and computing robust statistics (clipped means, by default) on the remaining values in those bins. The bin averages can then be interpolated via Akima splines or approximated by Chebyshev polynomials.

This code is run and re-run in many different configurations in many different contexts. For example, it is the first step run after instrument signature removal (§5.1) on each detector in both the nightly and data release pipelines. In a full DRP it is also run across a full visit to remove large-scale background features (H. Aihara et al. 2019). We sometimes also run background subtraction on coadds before detecting objects on them, especially when only full-visit backgrounds were subtracted from the input images.

In addition, we routinely run background subtraction as a small tweak every time we run detection (§5.3). We *define* our background to be the sum of all light that we do not consider part of a detection, which means it needs to be adjusted when the detection threshold changes (e.g. when we detect fainter objects on coadds).

### 5.3. *Source Detection*

In the LSST Science Pipelines, *detection* refers specifically to the process of finding above-threshold regions in images and one or more peaks within them (i.e., `Footprints`, as described in §4). Before this thresholding, we convolve each image with an approximation of its PSF model, as this is (approximately) optimal for detecting isolated point sources (J. Bosch et al. 2018). After detection, `Footprints` with multiple peaks are generally deblended (§5.4) before being measured (§5.5). Our detection algorithms assume the image has already been background-subtracted (§5.2).

All of our detection tasks are implemented in the `meas_algorithms` package.

#### 5.3.1. *Source Detection*

The `SourceDetectionTask` convolves the image with a Gaussian approximation to the exposure PSF and detects `Footprints` above a configurable threshold in either signal-to-noise or absolute flux level. It can optionally detect negative deviations as well as positive (which is useful when operating on difference images).

A serious challenge for this algorithm in deep or otherwise crowded fields is that noise peaks or below-threshold sources can be pushed above the detection threshold when they land in the wings of brighter neighbors. To reduce the number of spurious peaks due to this effect, `SourceDetectionTask` can optionally perform a *temporary* small-scale background subtraction before looking for peaks (but after finding the above-threshold regions); while the spline and Chebyshev models are not good models for the wings of large objects, they are often better than nothing.

#### 5.3.2. *Dynamic Detection*

The `DynamicDetectionTask` is a specialized version of `SourceDetectionTask` that adjusts detection thresholds based on the local background and noise. This task was initially developed to address detection efficiency issues noted in HSC data, thought to be related to correlations in the noise due to warping. First, the task 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 (see §5.5.3) 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.

#### 5.3.3. *Streak Masking*

Instead of looking for astrophysical sources like our other detection tasks, `MaskStreaksTask` is responsible for identifying and masking pixels affected by streaks (primarily artificial satellites). It searches for linear features with a Canny filter and the Kernel-Based Hough Transform (L. A. F. Fernandes & M. M. Oliveira 2008), and operates most effectively on difference images. The algorithm is described in more detail in C. Saunders (2021), which focuses on its usage on differences between PSF-matched warps and a PSF-matched coadd, as described in §6.2. This task is also used in traditional difference images, those formed by subtracting a template coadd from a science images, as described in §6.3.

### 5.4. *Deblending*

*Deblending* has become the standard astronomical term for dealing with images where multiple distinct astrophysical sources overlap. In the LSST Science Pipelines, it specifically means assigning different fractions of the flux of all of the pixels in a `Footprint` to each of the peaks in that `Footprint`, which are then each considered a "child" source. This fractional flux assignment is very different from creating a "segmentation map" that fully assigns each pixel to each child source, as done by, for example, Source Extractor (E. Bertin & S. Arnouts 1996). `HeavyFootprint` (an extension of `Footprint` that adds a flattened array of pixel values) is used to pass the per-pixel fluxes to downstream algorithms.

Deblending in the science pipelines is performed differently for single-band (visit) image processing vs. multi-band (coadd) image processing. For single-band images we use a modified version of the SDSS deblender (R. H. Lupton 2005) from the `meas_deblender` package. For multi-band images, we use a simplified version of the SCARLET deblending algorithm (P. Melchior et al. 2018) from the `scarlet_lite` package. Our motivation for simplifying the SCARLET algorithm can be found in F. Moolekamp (2023). More details on the algorithmic implementations of the deblending algorithms are given in §B.

### 5.5. *Source Measurement*

After sources are *detected* (§5.3) and optionally *deblended* (§5.4), the source measurement tasks are responsible for applying a suite of measurement *plugins* on the deblended pixels for each source. Centroiders, shape measurements, and photometry algorithms are all implemented as measurement plugins.

We also distinguish between measurement on the original detection image vs. measurement on a different image from the original detection. Measurement could be performed on a single-visit 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.

The measurement tasks, plugin base classes, and a suite of standard common plugins are defined in the `meas_base` package, as described in §5.5.4.

#### 5.5.1. *Framework Mechanics*

Plugins are enabled or disabled in a measurement task via the task's configuration (see §A for an example), and each plugin has its own configuration nested within the task configuration. When a measurement task is constructed, it constructs instances of its enabled plugins, providing them a schema object that they can use to declare and document their output columns. Each plugin is responsible for defining and filling in columns in the output source catalog, and almost all plugins include columns for uncertainties and at least one flag column to report failures.

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 are run in any order within a batch. The batch order is defined by the `getExecutionOrder` method, with smaller execution numbers being run first. The base class 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.

The measurement system also provides 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.

While the measurement tasks and plugin interfaces are pure Python, most concrete measurement plugins are implemented in C++, since they need to loop over pixels.

When a measurement task is run, it starts by making an empty `SourceCatalog` (from the "afw.table" library, see §4) with the plugin-defined schema and one row for each of the `Footprint` objects returned by previous detection and deblending tasks. It then temporarily replaces all pixels within `Footprints` by random noise. As the task loops over each row in the output catalog, that source's pixels are restored – either to the original `Exposure` pixels for isolated or otherwise un-deblended sources, or to the deblender's `HeavyFootprint` values for deblended children – and the plugins are called in execution order. Each plugin is given the full modified `Exposure` and a row of the output catalog to fill in. Note that plugins are *not* limited to using only the pixels within a `Footprint`; they get to decide themselves which pixels to use. After each source is measured, the task replaces its pixels with noise again, allowing the next source to be measured independently.

### 5.5.2. *Aperture Corrections*

With many different measures of photometry available for both stars and galaxies, establishing a consistent internal photometric system is a challenge, even before we consider the problem (covered in §5.7) of mapping that system to absolute photometry via an external reference catalog.

In addition, standard PSF modeling (§5.6) generally focuses on the core of the true PSF, which is only adequate for photometry for galaxies and fainter stars (while the wings of the PSF matter for bright galaxies as well, the semi-arbitrary definition of the boundary of such galaxies is typically a bigger source of photometry uncertainty).

Our solution to both of these problems is *aperture corrections*, in which we apply multiple photometry algorithms to a suite of isolated bright stars on each detector (by default, the same ones used to build the PSF model), compute the ratio of each algorithm to a standard one (a background-compensated top-hat aperture flux), and then interpolate that ratio using Chebyshev polynomials to other positions on the detector. The measured fluxes of other objects are then scaled by that interpolated ratio. This essentially forces all algorithms to produce the same results (on average) on stars.

It is clear that this approach is not exactly correct for galaxies or other extended sources in general, but we believe it is usually better than not correcting the galaxy photometry at all (it is, after all, the right thing to do for barely-resolved galaxies).

This scheme has two other serious limitations. The first is that many of the measurements we need to correct are noisy, even on bright stars, and the interpolated ratios can add significant uncertainty into the final fluxes. The second is that aperture corrections cannot in general be coadded along with the images (unlike PSF models); they are not a linear function of the data. Coadding the ratios *is* a valid first-order approximation in the limit where the photometry does not depend on the PSF and the PSFs of the contributing images are similar, and this is what we do at present, but we cannot currently quantify the error this approximation introduces. Our understanding is that using much larger PSF models is the cleanest solution to these problems, but the additional degrees of freedom that would entail may be hard to constrain with the information available.

### 5.5.3. *Sky Objects*

Sky objects are a special class of object that are not detected in the image being measured, but rather placed quasi-randomly in regions that are likely to be free of real sources. They're used to measure the sky background and noise properties of the image, and to assist in setting dynamic detection thresholds (see §5.3.2). After these pseudo-objects are placed, they're measured and processed in the same way as a true object. Their measurements are included in output catalogs, with an accompanying flag set to indicate that these are sky objects.

By default, sky objects are placed in regions of the sky whereby no detection footprints or bad pixels are located within an 8-pixel radius. To generate initial candidate positions, we use the quasi-random Halton sequence number generator with a seed based on the image ID. This is a deterministic sequence that mimics random sampling, but avoids clumping and gaps that can occur with true random sampling. Doing so allows us to generate a more unbiased sampling of the entire sky background, often with fewer required sky objects. The initial list of candidate positions is normally larger than the requested number of sky objects, by default of order 5 times. Attempts are made at placing sky objects iteratively until the requested number of sky objects is reached or the candidate list is exhausted. Typically, for visit-level detectors and coadd-level images, we place 100 sky objects.[7] Within the `DynamicDetectionTask` a larger number of temporary sky objects (default 1000) are placed in order to generate an updated detection threshold.

[7] When these pseudo-sources are placed into visit-level data, we instead refer to them as sky sources.

### 5.5.4. *Standard Measurement Plugins*

The `meas_base` package includes a suite of standard measurement plugins. These include second moment shapes (`base_SdssShape`) and adaptive centroids (J. R. Pier et al. 2003, `base_SdssCentroid`), circular aperture photometry (`base_CircularApertureFlux`), PSF-weighted photometry (base_PsfFlux), and many others. It also includes utility plugins that transform pixel bitmask values into catalog flags (`base_PixelFlags`) and apply coordinate transforms to centroids for use in forced photometry. The algorithms are described in detail in J. Bosch et al. (2018). There are also numerous extension plugins available in other pipeline packages described in the following sections.

### 5.5.5. *Gaussian Aperture and PSF Photometry*

`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 §6.3) 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 and/or an elliptical Gaussian aperture 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. 2024). 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 (2025) 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` (§5.5.9) or `MultiProFit` (§5.5.10) are recommended.

### 5.5.6. *Kron Photometry*

The `meas_extensions_photometryKron` implements Kron photometry (R. G. Kron 1980). Our Kron implementation uses a scaled version of the HSM shapes (See §5.5.7) to form an elliptical aperture, and then scales to 2.5 times the first radial moment.

Our implementation does not correct for the PSF in any way; this means its outputs should only be used for very well-resolved galaxies. We do not expect our Kron photometry to be competitive with most of our other galaxy photometry algorithms in robustness or precision, but it may be useful for comparison with external measurements.

### 5.5.7. *HSM Shapes*

The `meas_extensions_shapeHSM` package contains plugins that measure the shapes of objects.

This includes a suite of PSF-uncorrected shapes (for both objects and the PSF models) and a suite of PSF-corrected shapes intended for use in weak gravitational lensing. These are all based on the adaptive Gaussian-weighted moments algorithm described by G. M. Bernstein & M. Jarvis (2002) (the `base_SdssShape` algorithm described in §5.5.4 is another implementation of this algorithm). The implementation of these algorithms lives within the `hsm` module of the GalSim package (B. T. P. Rowe et al. 2015).

The PSF-uncorrected shapes include a variant that uses circular (rather than elliptical) Gaussian weights and another that adds noise to the PSF model image to match the noise in the data. Both of these play important roles in measuring PSF model quality by comparing the shapes of stars to the shapes of the PSF model at the positions of those stars.

In addition to the plugins that measure (adaptive) weighted moments, there are also a series of plugins to estimate the PSF-corrected ellipticities of objects. These are described more fully C. Hirata & U. Seljak (2003) in R. Mandelbaum et al. (2005). While no longer state-of-the-art in weak lensing, these algorithms are, fast, robust, and easy to use, as long as their biases are characterized via simulations (R. Mandelbaum et al. 2018; X. Li et al. 2022).

In addition to the second moments that characterize the size and ellipticity of the PSF, higher-order moments — those beyond second order — capture more subtle aspects of the PSF shape, such as skewness, kurtosis, and other asymmetric or non-Gaussian features. The definitions of the higher moments are given in T. Zhang et al. (2023).

#### 5.5.8. *Trailed Sources*

The `meas_extensions_trailedSources` package provides a plugin that measures the properties of trailed sources, such as those caused by moving objects like asteroids or satellites. It measures the length, angle, flux, centroid, and end points of a trailed source using the P. Vereš et al. (2012) model. This plugin is designed to refine the measurements of trail length, angle, and end points and of flux and centroid from previous measurement algorithms.

#### 5.5.9. *CModel Galaxy Fitting*

The `meas_modelfit` package's CModel algorithm is a reimplemention of the SDSS galaxy-model fitting approach, in which we fit a PSF-convolved elliptical exponential profile and de Vaucouleurs profile to each object separately, and then fit a linear combination of the two with the ellipse parameters held fixed. This is not as principled as a true bulge-disk decomposition, in which both models are fit simultaneously, but since each fit has fewer degrees of freedom, it can nevertheless work better on low signal-to-noise or poorly-resolved galaxies, where the degeneracies in a bulge-disk decomposition might otherwise lead to completely unphysical parameters. In practice, CModel is better thought of as a crude substitute for a single-Sersic fit than a kind of decomposition.

Because these are PSF-convolved models, we expect CModel to provide decent photometry (including for colors) on both small and well-resolved objects. At present, CModel flux uncertainties are known to be severely underestimated; at least some of this is due to the fact that the ellipse parameter uncertainties cannot easily be propagated into the flux uncertainties in the final fit.

The CModel code is essentially unchanged from the version used in the HSC pipelines, and additional details can be found in J. Bosch et al. (2018).

#### 5.5.10. *MultiProfit Galaxy Fitting*

MultiProFit is a package for Gaussian mixture model fitting (D. S. Taranu 2025). It is primarily used to provide multiband Sersic model fits to objects using all available coadds. The `multiprofit` package and its dependencies are included in the science pipelines but can also be installed independently, as they only depend on packages that are also available elsewhere.

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` (§5.5.9). 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. 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).

### 5.6. *PSF Modeling*

Point-spread function (PSF) modeling in the LSST Science Pipelines is largely delegated to external libraries, albeit with considerable wrapper code to adapt them to a common interface. We use both a heavily modified version of `PSFEx` (E. Bertin 2011, 2013) and `Piff` (M. Jarvis et al. 2021a,b) in our production pipelines. `PSFEx` is faster, and is used in nightly alert processing and to obtain a preliminary model in data release processing, while `Piff` is used in a second, more careful round of PSF estimation in data release processing for better accuracy (especially for weak gravitational lensing).

#### 5.6.1. *PSFEx*

The `meas_extensions_psfex` package provides an interface to our own library version of the `PSFEx` command-line tool. At its core is a task which prepares these selected stars for input into `PSFEx`, calls into the library, and converts the output into an LSST-specific PSF object. Key parameters such as spatial interpolation order and oversampling ratio are controlled via configuration.

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. A special task offers a built-in mechanism for star selection using strict cuts on signal-to-noise ratio, FWHM range, ellipticity, and quality flags.

#### 5.6.2. *Piff*

The `meas_extensions_piff` package is a wrapper around the `Piff` package. `Piff` is a modular package that supports various PSF models, interpolation schemes, and coordinate systems. It can operate on a per-CCD basis or over the full field of view. 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), but it has been designed to make it easy to add support for more options as needed.

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 configuration. 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. M. Jarvis et al. (2021b) configure `Piff` to fit in sky coordinates with a world-coordinate system transform (WCS) that includes CCD distortions such as tree rings. `meas_extensions_piff` is capable of doing the same, but our astrometric models do not yet include CCD distortions, and hence thus far we have not used this approach in our production configuration, as it doesn't significantly improve the quality of the PSF models. 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`.

## 5.7. *Astrometric and Photometric Calibration*

Astrometric and photometric calibration in data release processing are both performed in each of two very different steps. First, astrometric and photometric solutions for each detector are fit independently to a reference catalog, which is sufficient to enable matching and filtering for more sophisticated later algorithms and validation. In the nightly alert processing, we expect to be able to use a high-density reference catalog produced by the most recent Rubin data release, and these single-detector fits represent the final calibrations. In data release processing, much more sophisticated final astrometric and photometric transforms are then fit to catalogs from multiple epochs at once.

The Rubin pipeline uses the Starlink AST library (D. S. Berry et al. 2016) for persisting, composing, and evaluating coordinate transformations. We have our own class in the `afw` package for representing (among other things) photometric calibrations, usually with Chebyshev polynomials. These can also be multiplied and even mixed with AST-backed transform objects to represent pixel-area corrections. Our coordinate transforms are not exactly representable via the FITS WCS standard (E. W. Greisen & M. R. Calabretta 2002), which has no way to describe a chain of composed transforms. For public data releases, we approximate these transforms via a FITS TAN-SIP WCS (D. L. Shupe et al. 2005) for convenience, but precision astrometry should always use the exact composed transformation.

### 5.7.1. *Single-Frame Astrometric Calibration*

Single-frame astrometric fits are performed by a task in the `meas_astrom` package. 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 or pessimistic 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 fixed camera geometry or a FITS TAN-SIP WCS. We default to fitting the simple affine model because we have good distortion models for most of the instruments we support, often from previous fitting with `gbdes` (§5.7.2), and hence we do not need the extra degrees of freedom provided by a TAN-SIP model.

### 5.7.2. *GBDES*

The final astrometric solution is fit by a task 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 task: one that fits images from multiple bands at once, in which case the per-detector distortion model is also per-band; one that removes the restriction to a single tract and fits images regardless of their location on the sky by splitting the images into contiguous groups; and one that 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, a subtask is used to build an `afw Camera` object out of the native polynomial model.

Once the astrometric model has been calculated it is exported into a form understood by the AST library. A full description of astrometric calibration in the pipeline is given in C. Saunders (2024).

### 5.7.3. *Single-Frame Photometric Calibration*

Single frame photometric calibration is performed by a task in the `pipe_tasks` package. The catalog to be calibrated is down-selected to contain only 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 iteratively compared with the reference catalog magnitudes using a sigma-clipping algorithm, to fit a single magnitude zero point to the whole image. Precomputed color terms can also be applied to the reference catalog fluxes when needed.

### 5.7.4. *FGCM*

Global photometric calibration is computed via the Forward Global Calibration Method (FGCM D. L. Burke et al. 2018), as adapted for LSST (P. Fagrelius & E. S. 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 simultaneously 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, for example, 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).

### 5.7.5. *jointcal*

The `jointcal` packages provides an algorithm that 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. K. Parejko et al. (2025).

## 5.8. *Catalog Schemas*

Pipeline products must be transformed from the internal data model to the public data model defined in M. Jurić et al. (2023). A set of YAML files in the `pipe_tasks` repository are used for transforming the internal pipelines representation of the data to a standardized parquet output format. These parquet files are continually validated against an appropriate schema to ensure that the column names and types are correct as the pipelines codebase evolves. For data previews and releases, the parquet files are then ingested into the Qserv database, where the data catalogs are stored.

The public data model is defined by a set of files in the Felis (J. McCormick et al. 2024) YAML format which describe the schema of a data catalog, including its tables, columns, constraints and metadata. These are collectively referred to as the Science Data Model (SDM) schemas. The YAML files are managed via the `sdm_schemas` package with all changes validated by GitHub workflows. The schemas corresponding to Science Pipelines output are continually evolving with the pipelines codebase, so, for instance, column names and types may be updated to reflect changes to the internal data model. Schemas for data previews and releases represent a snapshot of the public data model at the time of the release and would typically only be updated with bug fixes, minor changes, or updates and additions to the metadata. For public-facing data catalogs, the Felis representation is used to generate a `TAP_SCHEMA` database describing the tables and columns available in the TAP service (P. Dowler et al. 2019) and serves as a source of documentation.

## 6. HIGH-LEVEL TASKS AND PIPELINES

In this section, we describe how the reusable components of §5 are assembled into pipelines for generating nightly alerts immediately from just-observed data, producing cumulative data releases roughly every year, and creating calibrations that are input to both on many cadences in between. This also includes some description of algorithmic details that are specific to certain pipelines.

### 6.1. *Single-Frame Processing and Calibration*

`CalibrateImageTask`, from the `pipe_tasks` package, performs "single frame processing" on a post-ISR (§5.1) single detector exposure. We repair and mask cosmic rays and defects, perform an initial set of detection (§5.3) and measurement (§5.5) passes to estimate the image PSF, compute an astrometric (§5.7.1) and photometric (§5.7.3) calibration, and compute summary statistics on the resulting exposure and catalog. The primary user-facing outputs of this task are the photometrically calibrated, background-subtracted `preliminary_visit_image`, the calibrated `preliminary_visit_image_background` that was subtracted from it, and `single_visit_star_unstandardized`, a catalog of bright point-like sources that were used as inputs to calibration, with only a small number of measurements performed on them. As this task only processes a single detector, it is used by both DRP (§6.7) and AP (§6.6), though with different configurations (AP is focused on latency, while DRP performs more measurements).

In DRP, `ReprocessVisitImageTask`, from the `drp_tasks` package, takes the outputs of the global astrometric and photometric models, a visit-level background, and PSF model and re-runs detection and measurement on the post-ISR single-detector exposure. The primary user-facing outputs of this task are the photometrically calibrated, background-subtracted `visit_image`, the calibrated `visit_image_background` that was subtracted from it, and `source_unstandardized`, a catalog of all sources detected to 5-sigma, with all relevant measurements performed on them (this is later schema-standardized and consolidated into the `source` per-visit catalog). Besides its primary purpose of performing detection and measurement using the "best" available inputs, this task also allows us to only have to save a small number of relatively small-sized intermediate products in order to re-generate the `visit_image` from a `raw` image, reducing long-term storage needs.

### 6.2. *Coaddition and Object Tables*

Coadded images are used both as static-sky templates for image subtraction and for detecting and measuring faint objects. The coaddition process is divided into two main stages: resampling the input images onto a common projection ("warping") defined by a `skymap` (§4) 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. An optional intermediate step can also be used to convolve the warped images 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, with a nearest-neighbor kernel for the integer bit mask. Input images are geometrically transformed using the 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. The default implementation performs outlier rejection to remove transient artifacts such as cosmic rays, ghosts, 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 (2018). 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.

Deep object catalogs are generated from coadds by running our detection code (see §5.3) on the coadd from each band independently, and then spatially matching the detected peaks across bands. The combined cross-band `Footprint` objects are formed from the union of the per-band ones. These are then passed to our multi-band deblender (see §B.3).

While a few of our measurement codes can then run natively on multiple bands together (e.g., the galaxy fitting codes described in §5.5.10), for most algorithms we obtain consistent cross-band measurements in two phases. First we measure on each band independently (albeit using the multi-band deblender outputs); then we measure again in each band in forced photometry mode, holding the position and (when relevant) shape fixed to the values measured in that object's reference band.

### 6.3. *Difference Image Analysis*

Image subtraction for transient/variable detection and analysis is implemented in the `ip_diffim` package, and is divided into three steps. While the image subtraction system could be used to work on other combinations, we primarily focus on subtracting a "template" coadd from a single-visit "science" image. First, the template coadd is warped 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, which produces a temporary difference image. Finally, peaks are detected on the difference image and DIASources ("difference image analysis sources") are measured. The final difference image with updated mask planes is written along with the DIASource catalog.

#### 6.3.1. *PSF Matching and Subtraction*

The primary implementation of image subtraction is based on C. Alard & R. H. Lupton (1998), and uses spatially-varying Gauss-Hermite 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).

#### 6.3.2. *DIA Detection and Measurement*

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 the `base_SdssCentroid` (§5.5.4). Finally, all configured measurement plugins are run, including HSM shape measurements (§5.5.7) and a trailed-source fit (§5.5.8).

#### 6.3.3. *Filtering Non-astrophysical DIASources*

A subset of the DIASources detected during `detectAndMeasureDiaSource` are expected to be non-astrophysical. These may be due to a variety of causes, including uncorrected instrument signatures, diffraction spikes and wings of bright stars, unmasked cosmic rays, algorithmic failures in background subtraction or image differencing, and satellites or space debris orbiting the Earth. The latter can leave long streaks that cross one or more LSSTCam detectors (J. A. Tyson et al. 2020; I. Hasan et al. 2022).

To reduce the number of DiaSources originating from non-astrophysical sources, we apply a multi-prong approach:

- During detection, we remove sources that have pixel flags characteristic of artifacts and unusable measurements, such as when all of the central pixels of a DIASource are saturated.
- In AP, we identify and filter out DIASources that are spatially and temporally coincident with the predicted positions of catalogued artificial satellites.
- DIASources with negative forced fluxes on the direct science image characteristic of bad background subtraction are filtered. This and subsequent catalog filtering is carried out by a task in the `ap_association` package.
- Trailed DIASources consistent with a rate of motion greater than 10 degrees per day are filtered.
- Blank sky sources used for noise estimation are removed from the output catalog.

No pixels are altered or redacted during the steps above. The machine-learned reliability score (§6.3.4) provides a further diagnostic that users can apply to remove remaining artifacts. These approaches are intended to enable science users to control the tradeoff between completeness and purity during their analysis while ensuring that the pipelines and databases can maintain their required performance.

#### 6.3.4. *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.).

#### 6.3.5. *Source Association*

The `ap_association` package contains multiple tasks for standardizing newly detected DIASources and associating them into "DIAObjects". Standardization converts the output catalogs from difference imaging to the format specified in `sdm_schemas` (§5.8), 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: DRP or 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; §6.3.6).

After association, an additional filtering step may be applied to DIASources with no matched DIAObject or Solar System object (§6.4). Properties of the source such as its reliability score (§6.3.4), 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.

#### 6.3.6. *Alert Production Database (APDB)*

The Alert Production Database (APDB; A. Salnikov & J. McCormick 2025)) supports SQL, Postgres, and Cassandra database formats. The previous history of DIAObjects, DIASources, and DiaForcedSources for the region containing the science image are loaded and passed to a task 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.

#### 6.3.7. *Alert Generation*

In order to enable real-time science, the AP pipelines generate alert packets for each detected DIASource. These packets are serialized in Apache Avro[8] format and then transmitted to community alert brokers via Kafka for further processing. M. T. Patterson et al. (2024) provides a high-level overview of the alert system.

Within the pipelines, alert packets are constructed 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 include 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.

### 6.4. *Solar System Pipelines*

The Solar System Pipeline (SSP; Figure 3) suite is responsible for (i) discovering previously unknown solar system objects by linking together observations (usually DIASources) unattributable to static (non-moving) sources, (ii) reporting these to the Minor Planet Center (MPC), (iii) computing basic physical characteristics such as absolute magnitudes and slope parameters for all asteroids where sufficient data is available, and (iv) using the orbits received from the MPC to associate their apparitions in the DIASource tables (both in real-time and as precovery). Unlike the other package described in this paper the Solar System Pipeline is not installed

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

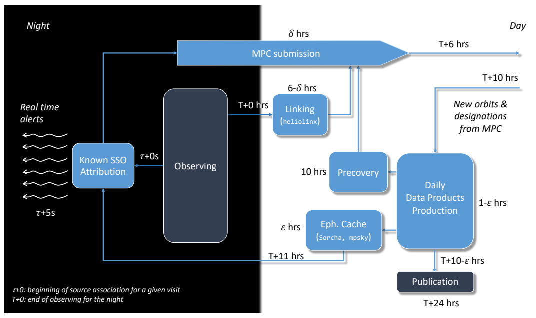

**Figure 3.** Detection, attribution, linking, submission and precovery of moving sources within the nightly data: The attribution is performed in real-time by the AP pipelines querying the `mpsky` service with resulting information attached to the alerts and queued for submission to the MPC. The linking is performed in daytime using `heliolinx`, with resulting links queued for submission to the MPC. Fetching of data from the MPC is performed automatically using PostgreSQL replication, with new data triggering recomputation of physical properties and precovery runs in the Daily Data Products Pipeline. Any observations discovered by the precovery procedure are queued for submission to the MPC, using the submission manager tool. The ephemeris cache is precomputed at dusk using `Sorcha` and `mpsky`, to enable fast attribution at nighttime. All timings denote design goals.

as part of the standard science pipelines software via the `lsst_distrib` metapackage.

The core element of the SSP is the linking pipeline, named `heliolinx` (A. Heinze et al. 2023). This code, run in daytime, clusters newly detected DIAObjects to search for candidate asteroids. The high-level procedure is to link DIASource detections within a night (when on-sky motion is approximately linear) into *tracklets*, to link these tracklets across multiple nights (into tracks) and to fit the tracks with an orbital model to identify those tracks that are consistent with an asteroid orbit. The Rubin implementation of this software (Heinze et al., in prep.) is based on the HelioLinC algorithm (M. J. Holman et al. 2018), with the key change being that the clustering is performed not on the sky, but in 3D space. It is designed to be capable of detecting 95% of all Solar System objects whose tracklets are observed over three nights within a 15-night window.[9]. `heliolinx` is written in C++, but provides a Python API including a standard `Task` API.

Candidate discoveries with high degree of certainty, as well as re-observations of already known objects, are reported to the Minor Planet Center (MPC) using the observation submission pipeline. The astrometric and photometric data are converted to the PSV variant of the Astrometric Data Exchange Standard (ADES; S. R. Chesley et al. 2017), and submitted via a HTTPS POST API provided by the MPC.

Following processing and validation of newly reported candidates, they're added to the MPC's central database. This database, including the table of orbits as well as observations, is replicated using PostgreSQL logical replication. Following the replication, the Daily Data Products Pipeline recomputes the absolute magnitudes of objects in the SSObject table, as well as some auxiliary per-observation information for individual observations (the SSSource table).

The replicated orbits and computed absolute magnitudes are utilized to predict positions (ephemerides) and magnitudes of solar system objects in subsequent night. To enable speedy retrieval (on order of 100 msec or less) of all objects in a visit, we precompute on-sky locations of all solar system objects, fit Chebyshev polynomials, and build an efficient HEALpix-based index allowing for fast lookup. These ephemerides are then served to association pipelines described in §??. This element of the pipeline is based on Sorcha (computation; S. R. Merritt et al. 2025) and `mpsky` (fast lookup and serving; M. Juric 2014). While still being constructed, a similar service is planned for "precovery" – the association of originally missed observations of solar system objects observed earlier in the survey.

Taken together, this suite of pipelines enables Rubin to identify sources consistent with being observations of objects in the solar system (both new and previously known), and makes these data public by reporting their discoveries to the Minor Planet Center and making them available to Rubin users within via the Prompt Products Database (PPDB; A. Salnikov & J. McCormick 2025).

## 6.5. *Calibration pipelines*

The high-level pipelines to build calibration products (`cp`) for the LSST cameras are defined in `cp_pipe`. They set ISR (see §5.1) 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 diagnostic displays.

Once calibration products are produced, they are "verified" (see C. Z. Waters & E. S. Rykoff (2025) for more details) using `cp_verify` pipelines by checking they pass metrics defined in R. H. Lupton et al. (2025). In this case, verify configuration parameters enable all correc-

[9] Detailed criteria are specified in the LSST Observatory System Specification (OSS) document `OSS-REQ-0159` (C. F. Claver & The LSST Systems Engineering Integrated Project Team 2018)

tions 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.

### 6.6. *ap_pipe*

The `ap_pipe` package defines the pipeline(s) to be used for real-time Alert Production processing (K.-T. Lim 2023). These pipelines include instrument signature removal (§5.1), calibration (§5.7), measurement plugins (§5.5), image differencing (§6.3), source association (§6.3.5), and alert generation (§6.3.7). Some of these tasks are shared with the pipelines in `drp_pipe` (§6.7), but are 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 (§3.5). The only other customization is an extra task for handling DECam's inter-chip crosstalk, which does not have an equivalent for Rubin instruments.

### 6.7. *drp_pipe*

The `drp_pipe` package defines the pipeline(s) to be used for the annual Data Release Production processing. These pipelines include instrument signature removal (§5.1), calibration (§5.7), measurement plugins (§5.5), coaddition and coadd-processing (§6.2), image differencing (§6.3), source association (§6.3.5), and global calibration (§5.7). Some of these tasks are shared with the pipelines in `ap_pipe`, but are configured to prioritize accuracy over speed.

`drp_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 DRP-specific algorithms and configuration settings, they are, as much as possible, the "same" pipeline, differing primarily entirely in loading instrument defaults from `obs` packages (§3.5). The pipelines for some instruments also disable certain multi-visit calibration routines like FGCM (§5.7.4) and GBDES (§5.7.2), as those can take significant effort to commission on a new instrument, and this has not been done in all cases. The only other customization is an extra task for handling DECam's inter-chip crosstalk, which does not have an equivalent for Rubin instruments.

## 7. ANALYSIS TOOLING

### 7.1. *Display Abstractions*

The `afw.display` subpackage defines a simple image display abstraction layer that be used to show our image objects via multiple applications and libraries. This includes programmatic control of stretch levels and colors, WCS mapping, semi-transparent bitmask overlays, and simple geometric region support.

There are currently implementations 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). While most of these tools have considerable functionality beyond what our abstraction layer provides, the ability to interact easily with them in a consistent, programmatic way in many different contexts (e.g., DS9 on personal machines, matplotlib and Ginga in Jupyter notebooks, and Firefly in the Rubin Science Platform) is invaluable, and usually the minimal `afw.display` interface provides all we need.

### 7.2. *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 are generated in sync with the archived plots.

`analysis_tools` is built on top of `pex_config` (see §3.4 for details); this allows it to integrate into the rest of the software pipelines and to be very flexible in its application. It also means that the configurations for all tools are saved every time, allowing them to be accurately recreated with information such as the signal to noise ratio used to filter the data.

An example plot, made with HSC data, is shown in Fig. 4. The package was designed to handle the large data volumes and memory requirements that the survey will generate to ensure that the initial QA 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.

`analysis_tools` is the successor to `faro` (L. P. Guy et al. 2022) and `analysis_drp`. Previously the QA plots and metrics were calculated by different packages and run in different tasks; this allowed the code and selection criteria used to get out of sync between the plots and metrics, and in some cases it led to extreme duplication of I/O. The more consolidated approach used by `analysis_tools` improves memory and speed performance as well as maintaining consistency.

Further information and examples can be found in S. L. Reed (2025).

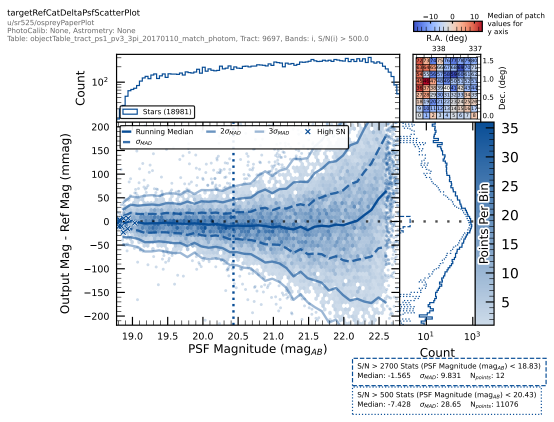

**Figure 4.** An example figure produced as part of the standard processing by analysis tools, the plot is information dense as it is designed for an audience familiar with the outputs but a simplified version can also be produced for talks and publications by setting a config option. The metrics shown in the bottom right of the plot are also saved separately to be displayed by various pieces of QA tooling.

## 7.3. *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 5.

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.

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

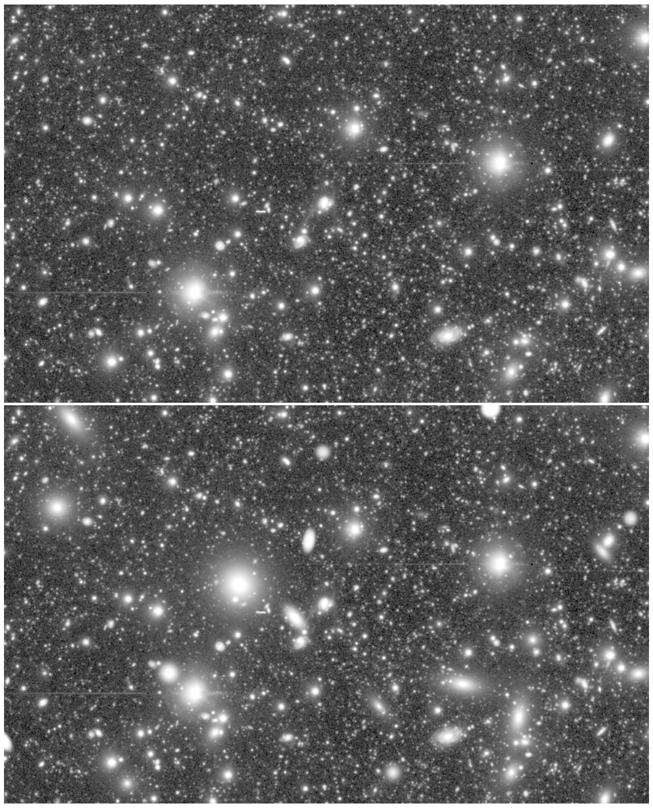

**Figure 5.** An HSC i-band cutout from tract 9813, patch 42, showing before (top) and after (bottom) the injection of a series of synthetic Sérsic sources. Images are 100 arcseconds on the short axis, log scaled across the central 99.5% flux range, and smoothed with a Gaussian kernel of FWHM 3 pixels.

sources. Optional modifications to the noise profiles of injected sources and the variance plane of the image can also be performed.

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.

## 8. CONCLUSIONS

The LSST Science Pipelines Software has been developed over 20 years to support the processing of the Legacy Survey of Space and Time. It has been used to process formal data releases from both Hyper Suprime-Cam and the Rubin Observatory's LSSTComCam and is now being used to process LSSTCam commissioning data. The software is designed to be extensible and reusable, supporting a plugin architecture that allows new algorithms to be added without modifying the core codebase and includes a dataset tracking system and graph builder that supports scaling of processing on large batch systems.

## ACKNOWLEDGMENTS

This material is based upon work supported in part by the National Science Foundation through Cooperative Agreements AST-1258333 and AST-2241526 and Cooperative Support Agreements AST-1202910 and AST-2211468 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 LSST-DA Institutional Members. This research has made use of NASA's Astrophysics Data System Bibliographic Services.

*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

Aihara, H., AlSayyad, Y., Ando, M., et al. 2019, PASJ, 71, 114, doi: 10.1093/pasj/psz103

Aihara, H., AlSayyad, Y., Ando, M., et al. 2022, PASJ, 74, 247, doi: 10.1093/pasj/psab122

Alard, C., & Lupton, R. H. 1998, ApJ, 503, 325, doi: 10.1086/305984

AlSayyad, Y. 2018, Coaddition Artifact Rejection and CompareWarp, Data Management Technical Note DMTN-080, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2583441

Astropy Collaboration, Price-Whelan, A. M., Sipőcz, B. M., et al. 2018, AJ, 156, 123, doi: 10.3847/1538-3881/aabc4f

Astropy Collaboration, Price-Whelan, A. M., Lim, P. L., et al. 2022, ApJ, 935, 167, doi: 10.3847/1538-4357/ac7c74

Axelrod, T., Connolly, A., Ivezic, Z., et al. 2004, 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, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 7740, Software and Cyberinfrastructure for Astronomy, ed. N. M. Radziwill & A. Bridger, 774015, doi: 10.1117/12.857297

Babuji, Y., Woodard, A., Li, Z., et al. 2019, in Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing, HPDC '19 (New York, NY, USA: Association for Computing Machinery), 25–36, doi: 10.1145/3307681.3325400

Berk, A., Anderson, G. P., Bernstein, L. S., et al. 1999, 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, 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 http://ascl.net/2210.011

Bernstein, G. M., & Jarvis, M. 2002, AJ, 123, 583, doi: 10.1086/338085

Bernstein, G. M., Armstrong, R., Plazas, A. A., et al. 2017, PASP, 129, 074503, doi: 10.1088/1538-3873/aa6c55

Berry, D., Graves, S., Bell, G. S., et al. 2022, 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, Astronomy and Computing, 15, 33, doi: 10.1016/j.ascom.2016.02.003

Bertin, E. 2011, 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

Bertin, E. 2013, PSFEx: Point Spread Function Extractor,, Astrophysics Source Code Library, record ascl:1301.001 http://ascl.net/1301.001

Bertin, E., & Arnouts, S. 1996, A&AS, 117, 393, doi: 10.1051/aas:1996164

Bloom, J. S., Richards, J. W., Nugent, P. E., et al. 2012, PASP, 124, 1175, doi: 10.1086/668468

Bohlin, R. C. 2007, 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., Armstrong, R., Bickerton, S., et al. 2018, PASJ, 70, S5, doi: 10.1093/pasj/psx080

Boulade, O., Charlot, X., Abbon, P., et al. 2003, 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, PASP, 136, 045003, doi: 10.1088/1538-3873/ad3aa2

Burke, D. L., Rykoff, E. S., Allam, S., et al. 2018, AJ, 155, 41, doi: 10.3847/1538-3881/aa9f22

Cai, M., Xu, Z., Fan, L., et al. 2025, arXiv e-prints, arXiv:2501.15018, doi: 10.48550/arXiv.2501.15018

Chesley, S. R., Hockney, G. M., & Holman, M. J. 2017, in AAS/Division for Planetary Sciences Meeting Abstracts, Vol. 49, AAS/Division for Planetary Sciences Meeting Abstracts #49, 112.14

Claver, C. F., & The LSST Systems Engineering Integrated Project Team. 2018, Observatory System Specifications (OSS), Systems Engineering Controlled Document LSE-30, NSF-DOE Vera C. Rubin Observatory. https://ls.st/LSE-30

Crill, B. P., Werner, M., Akeson, R., et al. 2020, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 11443, Space Telescopes and Instrumentation 2020: Optical, Infrared, and Millimeter Wave, ed. M. Lystrup & M. D. Perrin, 114430I, doi: 10.1117/12.2567224

DePoy, D. L., Abbott, T., Annis, J., et al. 2008, 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

Dowler, P., Rixon, G., Tody, D., & Demleitner, M. 2019, Table Access Protocol Version 1.1,, IVOA Recommendation 27 September 2019 doi: 10.5479/ADS/bib/2019ivoa.spec.0927D

Duev, D. A., Mahabal, A., Masci, F. J., et al. 2019, MNRAS, 489, 3582, doi: 10.1093/mnras/stz2357

Fagrelius, P., & Rykoff, E. S. 2025, Rubin Observatory Baseline Calibration Plan, Commissioning Technical Note SITCOMTN-086, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2583850

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

Fausti Neto, A., Economou, F., Reuter, M. A., et al. 2024, 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

Fernandes, L. A. F., & Oliveira, M. M. 2008, Pattern Recognition, 41, 299, doi: 10.1016/j.patcog.2007.04.003

Flaugher, B., Diehl, H. T., Honscheid, K., et al. 2015, AJ, 150, 150, doi: 10.1088/0004-6256/150/5/150

Goldstein, D. A., D'Andrea, C. B., Fischer, J. A., et al. 2015, AJ, 150, 82, doi: 10.1088/0004-6256/150/3/82

Gower, M., Kowalik, M., Lust, N. B., Bosch, J. F., & Jenness, T. 2022, arXiv e-prints, arXiv:2211.15795, doi: 10.48550/arXiv.2211.15795

Greisen, E. W., & Calabretta, M. R. 2002, A&A, 395, 1061, doi: 10.1051/0004-6361:20021326

Guy, L. P., Bechtol, K., Carlin, J. L., et al. 2022, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 12189, Software and Cyberinfrastructure for Astronomy VII, 121890M, doi: 10.1117/12.2628887

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

Hasan, I., Tyson, J. A., Saunders, C., & Xin, B. 2022, Astronomy and Computing, 39, 100584, doi: 10.1016/j.ascom.2022.100584

Heinze, A., Juric, M., & Kurlander, J. 2023, heliolinx: Open Source Solar System Discovery Software, https://github.com/heliolinx/heliolinx

Hirata, C., & Seljak, U. 2003, MNRAS, 343, 459, doi: 10.1046/j.1365-8711.2003.06683.x

Holman, M. J., Payne, M. J., Blankley, P., Janssen, R., & Kuindersma, S. 2018, AJ, 156, 135, doi: 10.3847/1538-3881/aad69a

HTCondor Team. 2024, HTCondor, 24.2.1 Zenodo, doi: 10.5281/zenodo.2579447

Hunter, J. D. 2007, Computing in Science and Engineering, 9, 90, doi: 10.1109/MCSE.2007.55

Ingraham, P., Clements, A. W., Ribeiro, T., et al. 2020, 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, 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 http://ascl.net/2102.024

Jarvis, M., Bernstein, G. M., Amon, A., et al. 2021b, MNRAS, 501, 1282, doi: 10.1093/mnras/staa3679

Jenness, T. 2020, 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., Bosch, J., Owen, R., et al. 2016, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 9913, Software and Cyberinfrastructure for Astronomy IV, ed. G. Chiozzi & J. C. Guzman, 99130G, doi: 10.1117/12.2231313

Jenness, T., Economou, F., Findeisen, K., et al. 2018, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 10707, Software and Cyberinfrastructure for Astronomy V, ed. J. C. Guzman & J. Ibsen, 1070709, doi: 10.1117/12.2312157

Jenness, T., Bosch, J. F., Salnikov, A., et al. 2022, 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, 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, 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

Juric, M. 2014, mpsky: Multi-purpose sky catalog cross-matching, https://github.com/mjuric/mpsky

Jurić, M., Ciardi, D., Dubois-Felsmann, G., & Guy, L. 2019, LSST Science Platform Vision Document, Systems Engineering Controlled Document LSE-319, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2587242

Jurić, M., Kantor, J., Lim, K. T., et al. 2017, 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. S., Becker, A. C., et al. 2023, Data Products Definition Document, Systems Engineering Controlled Document LSE-163, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2587118

Kahn, S. M., Kurita, N., Gilmore, K., et al. 2010, 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, 77350J, doi: 10.1117/12.857920

Kannawadi, A. 2025, Consistent galaxy colors with Gaussian-Aperture and PSF photometry, Data Management Technical Note DMTN-190, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2583849

Karavakis, E., Guan, W., Yang, Z., et al. 2024, in European Physical Journal Web of Conferences, Vol. 295, European Physical Journal Web of Conferences, 04026, doi: 10.1051/epjconf/202429504026

Knight, S. 2005, 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

Kron, R. G. 1980, ApJS, 43, 305, doi: 10.1086/190669

Kuijken, K. 2008, A&A, 482, 1053, doi: 10.1051/0004-6361:20066601

Kuijken, K., Heymans, C., Hildebrandt, H., et al. 2015, MNRAS, 454, 3500, doi: 10.1093/mnras/stv2140

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

Li, X., Miyatake, H., Luo, W., et al. 2022, PASJ, 74, 421, doi: 10.1093/pasj/psac006

Lim, K.-T. 2023, Proposal and Prototype for Prompt Processing, Data Management Technical Note DMTN-219, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2585429

Lupton, R. H. 2005, SDSS Image Processing I: The Deblender, https://www.astro.princeton.edu/textasciitilderhl/photomisc/deblender.pdf

Lupton, R. H., Plazas Malagón, A. A., & Waters, C. Z. 2025, Verifying LSST Calibration Data Products, Data Management Technical Note DMTN-101, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2586569

Lust, N. B., Jenness, T., Bosch, J. F., et al. 2023, arXiv e-prints, arXiv:2303.03313, doi: 10.48550/arXiv.2303.03313

Mandelbaum, R., Hirata, C. M., Seljak, U., et al. 2005, MNRAS, 361, 1287, doi: 10.1111/j.1365-2966.2005.09282.x

Mandelbaum, R., Miyatake, H., Hamana, T., et al. 2018, PASJ, 70, S25, doi: 10.1093/pasj/psx130

McCormick, J., Dubois-Felsmann, G. P., Salnikov, A., Van Klaveren, B., & Jenness, T. 2024, arXiv e-prints, arXiv:2412.09721, doi: 10.48550/arXiv.2412.09721

Melchior, P., Joseph, R., & Moolekamp, F. 2019, arXiv e-prints, arXiv:1910.10094, doi: 10.48550/arXiv.1910.10094

Melchior, P., Moolekamp, F., Jerdee, M., et al. 2018, Astronomy and Computing, 24, 129, doi: 10.1016/j.ascom.2018.07.001

Merritt, S. R., Fedorets, G., Schwamb, M. E., et al. 2025, AJ, 170, 100, doi: 10.3847/1538-3881/add3ec

Miyazaki, S., Komiyama, Y., Kawanomoto, S., et al. 2018, PASJ, 70, S1, doi: 10.1093/pasj/psx063

Moolekamp, F. 2023, The current state of scarlet and looking toward the future, Data Management Technical Note DMTN-194, NSF-DOE Vera C. Rubin Observatory. https://dmtn-194.lsst.io/

Morrison, C. B. 2018, Pessimistic Pattern Matching for LSST, Data Management Technical Note DMTN-031, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2586578

Mueller, F., et al. 2023, 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, PASA, 38, e004, doi: 10.1017/pasa.2020.45

NSF-DOE Vera C. Rubin Observatory. 2020, LSST Atmospheric Transmission Imager and Slitless Spectrograph (LATISS), NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2571930

NSF-DOE Vera C. Rubin Observatory. 2025, Legacy Survey of Space and Time Data Preview 1 [Data set], NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/RUBIN/2570308

O'Mullane, W., Economou, F., Lim, K.-T., et al. 2022, arXiv e-prints, arXiv:2211.13611, doi: 10.48550/arXiv.2211.13611

O'Mullane, W., Economou, F., Huang, F., et al. 2024, 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, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2586668

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

Parejko, J. K., Astier, P., & Bosch, J. F. 2025, jointcal: Simultaneous Astrometry & Photometry for thousands of Exposures with Large CCD Mosaics, Data Management Technical Note DMTN-036, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2586685

Park, H., Karpov, S., Nomerotski, A., & Tsybychev, D. 2020, Journal of Astronomical Telescopes, Instruments, and Systems, 6, 011005, doi: 10.1117/1.JATIS.6.1.011005

Paszke, A., Gross, S., Massa, F., et al. 2019, 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. T., Bellm, E. C., Swinbank, J. D., Nelson, S., & Smart, B. M. 2024, Design of the LSST Alert Distribution System, Data Management Technical Note DMTN-093, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2586493

Pier, J. R., Munn, J. A., Hindsley, R. B., et al. 2003, AJ, 125, 1559, doi: 10.1086/346138

Plazas Malagón, A. A., Waters, C., Broughton, A., et al. 2025, Journal of Astronomical Telescopes, Instruments, and Systems, 11, 011209, doi: 10.1117/1.JATIS.11.1.011209

Reed, S. L. 2025, An introduction to Analysis Tools, Data Management Technical Note DMTN-314, NSF-DOE Vera C. Rubin Observatory. https://dmtn-314.lsst.io/

Reiss, D. J., & Lupton, R. H. 2016, Implementation of Image Difference Decorrelation, Data Management Technical Note DMTN-021, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2586490

Roby, W., Wu, X., Dubois–Felmann, G., et al. 2020, 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, 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, 130961S, doi: 10.1117/12.3019698

Rowe, B. T. P., Jarvis, M., Mandelbaum, R., et al. 2015, Astronomy and Computing, 10, 121, doi: 10.1016/j.ascom.2015.02.002

Salnikov, A., & McCormick, J. 2025, Current status of APDB and PPDB implementation, Data Management Technical Note DMTN-293, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2586676

Saunders, C. 2021, Streak Masking in DM Image Processing, Data Management Technical Note DMTN-197, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2586496

Saunders, C. 2024, Astrometric Calibration in the LSST Pipeline, Data Management Technical Note DMTN-266, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2583846

Schutt, T., Jarvis, M., Roodman, A., et al. 2025, The Open Journal of Astrophysics, 8, 26, doi: 10.33232/001c.132299

Shupe, D. L., Moshir, M., Li, J., et al. 2005, 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

SLAC National Accelerator Laboratory, & NSF-DOE Vera C. Rubin Observatory. 2024, LSST Commissioning Camera, SLAC National Accelerator Laboratory (SLAC), Menlo Park, CA (United States), doi: 10.71929/RUBIN/2561361

SLAC National Accelerator Laboratory, & NSF-DOE Vera C. Rubin Observatory. 2025, The LSST Camera (LSSTCam), SLAC National Accelerator Laboratory (SLAC), Menlo Park, CA (United States), doi: 10.71929/rubin/2571927

Stalder, B., Reil, K., Aguilar, C., et al. 2022, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 12184, Ground-based and Airborne Instrumentation for Astronomy IX, ed. C. J. Evans, J. J. Bryant, & K. Motohara, 121840J, doi: 10.1117/12.2630184

Sutherland, W., Emerson, J., Dalton, G., et al. 2015, A&A, 575, A25, doi: 10.1051/0004-6361/201424973

Tabur, V. 2007, PASA, 24, 189, doi: 10.1071/AS07028

Taranu, D. S. 2025, The MultiProFit astronomical source modelling code, Data Management Technical Note DMTN-312, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2584108

Thomas, S. J., Barr, J., Callahan, S., et al. 2022, 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

Tyson, J. A., Ivezić, Ž., Bradshaw, A., et al. 2020, AJ, 160, 226, doi: 10.3847/1538-3881/abba3e

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

Vera C. Rubin Observatory Team. 2025, The Vera C. Rubin Observatory Data Preview 1, Technical Note RTN-095, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2570536

Vereš, P., Jedicke, R., Denneau, L., et al. 2012, PASP, 124, 1197, doi: 10.1086/668616

Wang, D. L., Monkewitz, S. M., Lim, K.-T., & Becla, J. 2011, 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, 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. Z., & Rykoff, E. S. 2025, Calibration Generation, Verification, Acceptance, and Certification, Data Management Technical Note DMTN-222, NSF-DOE Vera C. Rubin Observatory, doi: 10.71929/rubin/2586498

Wright, A. H., Kuijken, K., Hildebrandt, H., et al. 2024, A&A, 686, A170, doi: 10.1051/0004-6361/202346730

Zackay, B., Ofek, E. O., & Gal-Yam, A. 2016, ApJ, 830, 27, doi: 10.3847/0004-637X/830/1/27

Zhang, T., Almoubayyed, H., Mandelbaum, R., et al. 2023, MNRAS, 520, 2328, doi: 10.1093/mnras/stac3350

## APPENDIX

## A. PIPELINE CONFIGURATION

An example of a pipeline configuration 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 configuration settings 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")
except LookupError as e:
    print(f"Cannot enable shapeHSM ({e})")
else:
    path = os.path.join(location, "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"]
```

**Listing 1.** Code configuration in python

The design of pex_config centers around the concepts of Field 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.

## B. DEBLENDING

### B.1. *Single-band Deblending*

Deblending on single-band images (i.e., visit) is performed using the meas_deblender package and is based on the deblender used in SDSS (R. H. Lupton 2005), 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, brightness of the nearest sources, etc.

### B.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 6 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 $\chi^2$ 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 7 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.

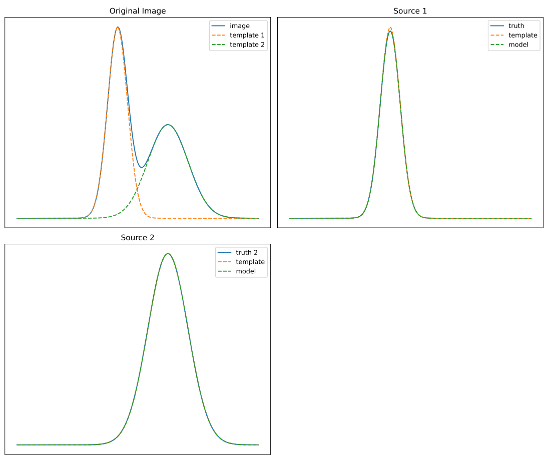

**Figure 6.** A 1D slice of two blended Gaussian sources illustrating how symmetry can be utilized to model blended sources.

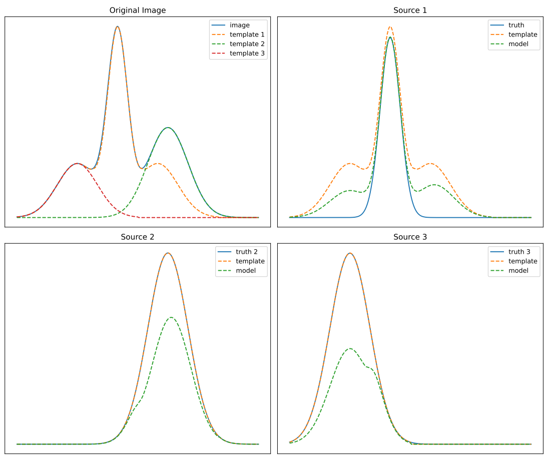

**Figure 7.** A 1D slice through three aligned Gaussian sources, demonstrating a failure case of relying on symmetry for generating deblender templates. Notice that for sources 2 and 3 the templates are reasonable but due to the inability of source 1 to use symmetry to constrain flux in the blended region, 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.

### B.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)
- 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 $\chi^2$-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 `object_model_data` 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.