VERA C. RUBIN OBSERVATORY

Source Injection with the Science Pipelines: Updates and Plans

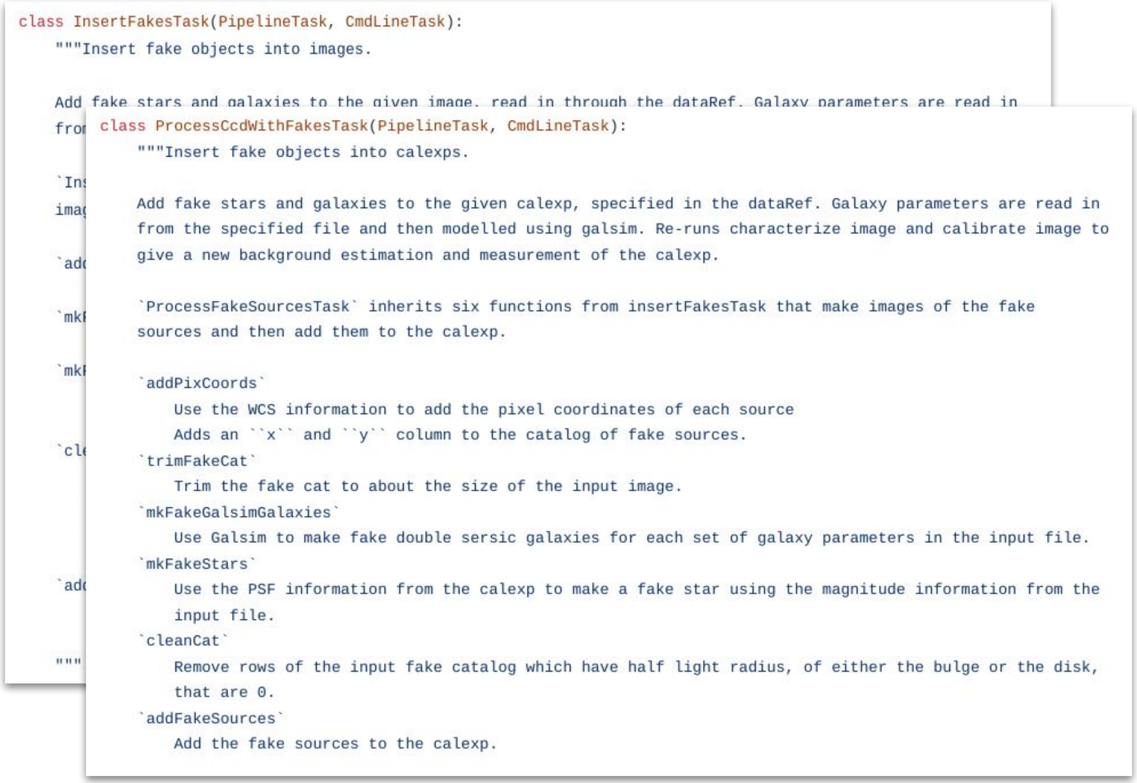
Lee Kelvin

Project and Community Workshop

August 8 2023

NSF AURA U.S. DEPARTMENT OF ENERGY SLAC CHARLES AND LISA SIMONYI FUND FOR ARTS AND SCIENCES LSST CORPORATION

The road so far…



```
class InsertFakesTask(PipelineTask, CmdLineTask):
    """Insert fake objects into images.

    Add fake stars and galaxies to the given image, read in through the dataRef. Galaxy parameters are read in
    fro

    `Ins
    imag

    `add

    `mkF

    `mkF


    `cl



    `add


    """
```

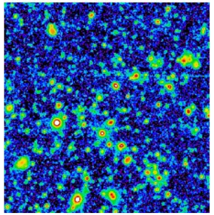
```
class ProcessCcdWithFakesTask(PipelineTask, CmdLineTask):
    """Insert fake objects into calexps.

    Add fake stars and galaxies to the given calexp, specified in the dataRef. Galaxy parameters are read in
    from the specified file and then modelled using galsim. Re-runs characterize image and calibrate image to
    give a new background estimation and measurement of the calexp.

    `ProcessFakeSourcesTask` inherits six functions from insertFakesTask that make images of the fake
    sources and then add them to the calexp.

    `addPixCoords`
        Use the WCS information to add the pixel coordinates of each source
        Adds an ``x`` and ``y`` column to the catalog of fake sources.
    `trimFakeCat`
        Trim the fake cat to about the size of the input image.
    `mkFakeGalsimGalaxies`
        Use Galsim to make fake double sersic galaxies for each set of galaxy parameters in the input file.
    `mkFakeStars`
        Use the PSF information from the calexp to make a fake star using the magnitude information from the
        input file.
    `cleanCat`
        Remove rows of the input fake catalog which have half light radius, of either the bulge or the disk,
        that are 0.
    `addFakeSources`
        Add the fake sources to the calexp.
```

InsertFakesTask injects into coadd-level imaging.

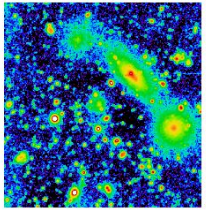
ProcessCcdWithFakesTask injects into single-frame-level imaging.

Allows for **bulge+disk**, PSF and FITS file injection.

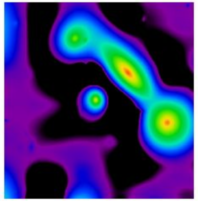
original



+fakes



difference



Limitations of InsertFakesTask

Monolithic framework

- difficult to modularize how and when sources are injected
- changes to single frame processing do not filter down to ProcessCcdWithFakesTask

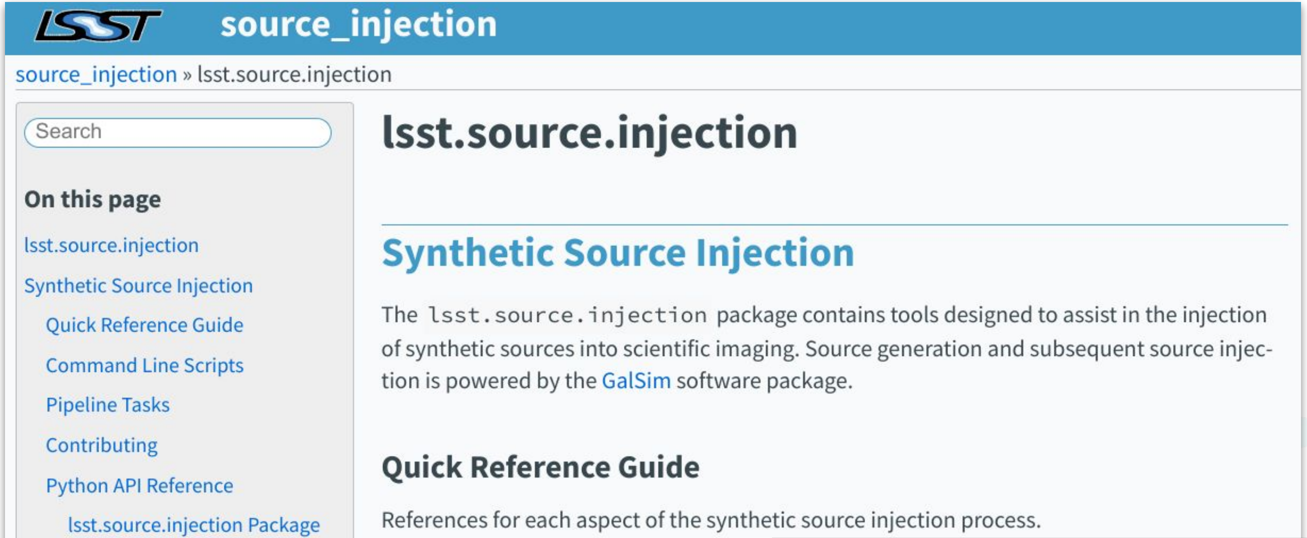
Non-trivial to ingest synthetic source catalogue into the data Butler

- no dedicated Rubin tool to facilitate simple ingestion
- raises the bar for new users who wish to inject synthetic sources

Strict formats for input catalogs: bulge+disk and/or PSF

- does not facilitate alternative models, e.g., simple Sérsic, bulge+disk+bar
- requires a translation layer to convert user supplied catalogues into GalSim format

A fresh start: the new source_injection repo



source_injection

source_injection » lsst.source.injection

Search

On this page

- lsst.source.injection
- Synthetic Source Injection
  - Quick Reference Guide
  - Command Line Scripts
  - Pipeline Tasks
  - Contributing
  - Python API Reference
    - lsst.source.injection Package
      - Modules
      - Functions
      - Classes
      - Class Inheritance Diagram

lsst.source.injection

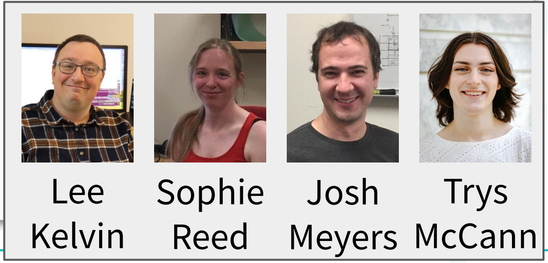
Synthetic Source Injection

The `lsst.source.injection` package contains tools designed to assist in the injection of synthetic sources into scientific imaging. Source generation and subsequent source injection is powered by the GalSim software package.

Quick Reference Guide

References for each aspect of the synthetic source injection process.

- Make an Injection Pipeline
- Generate an Injection Catalog
- Ingest an Injection Catalog
- Inject into Visit-Level Imaging
- Inject into Coadd-Level Imaging
- Explore Injected Outputs



Lee Kelvin | Sophie Reed | Josh Meyers | Trys McCann

DM-34170
Repository created, testing begins

↓

DM-34253
Majority of functionality in place; "soft launched"

↓

DM-39728
Documentation complete, added to `lsst_distrib`

The SI repo will need to be cloned and set up until DM-39728 is completed.

What does source_injection do?

The source_injection repo facilitates synthetic source injection into **single-frame-level** and **coadd-level** imaging.

Many features of GalSim are natively supported: injection of **simple** and **complex** synthetic sources or **user-supplied FITS files**.

A series of **utility functions** help users construct fully qualified injection pipelines, generate example injection catalogs and ingest injection catalogs into the Butler.

Primary expected use case: V&V stress tests → **truth is known**.

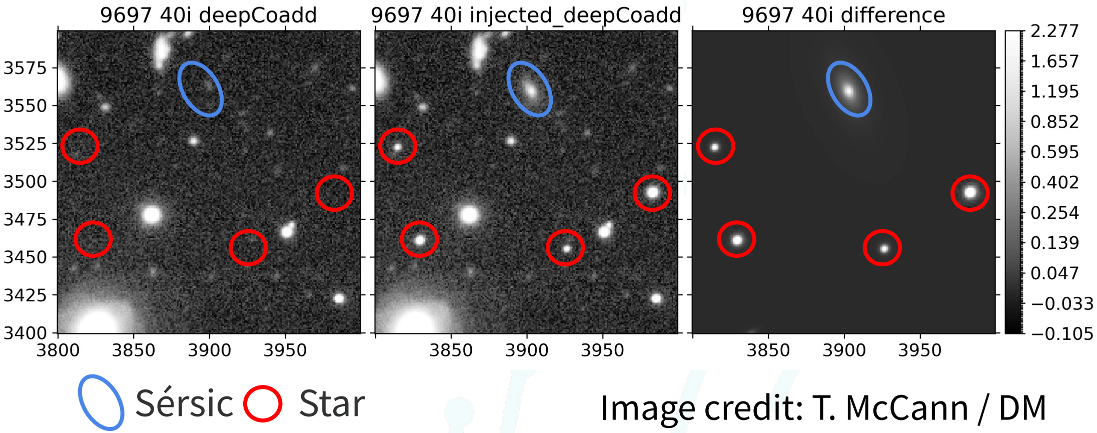
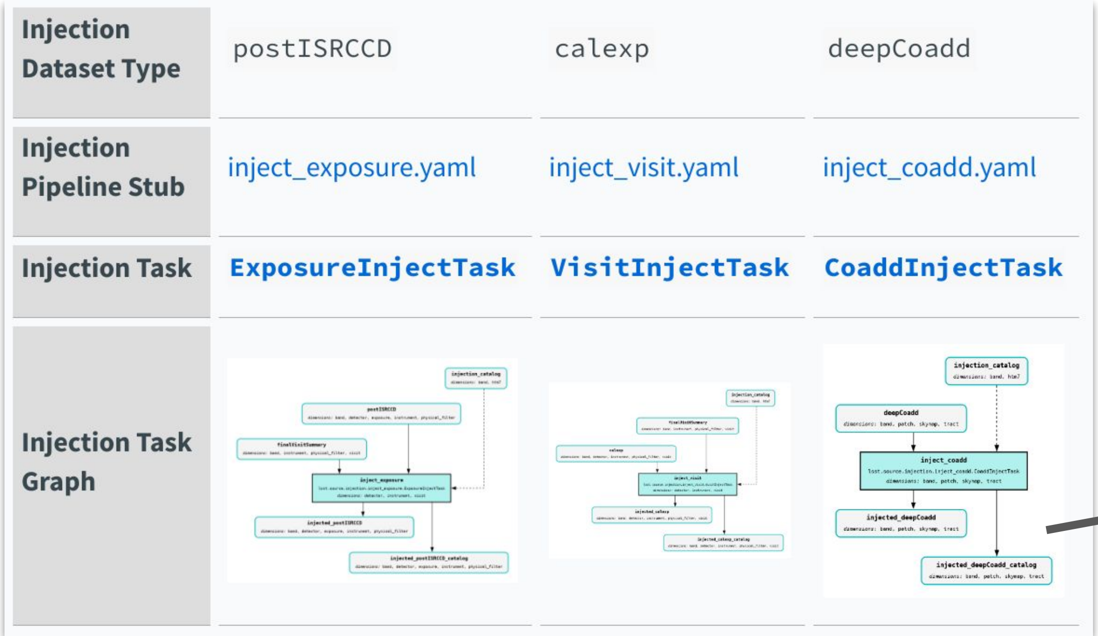


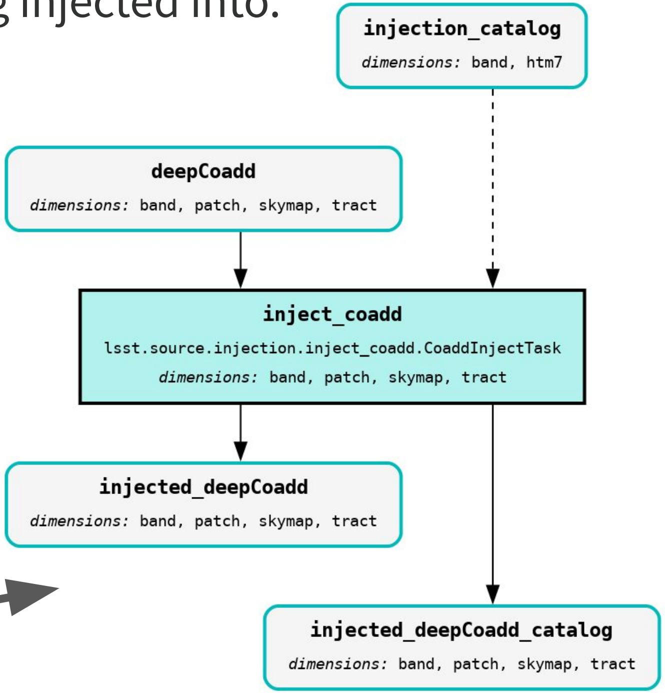
Image credit: T. McCann / DM

Source injection pipeline tasks

Three pipelines, corresponding to the dataset type being injected into.



Injection Dataset Type	postISRCCD	calexp	deepCoadd
Injection Pipeline Stub	inject_exposure.yaml	inject_visit.yaml	inject_coadd.yaml
Injection Task	**ExposureInjectTask**	**VisitInjectTask**	**CoaddInjectTask**
Injection Task Graph			



injection_catalog
dimensions: band, htm7

deepCoadd
dimensions: band, patch, skymap, tract

inject_coadd
lsst.source.injection.inject_coadd.CoaddInjectTask
dimensions: band, patch, skymap, tract

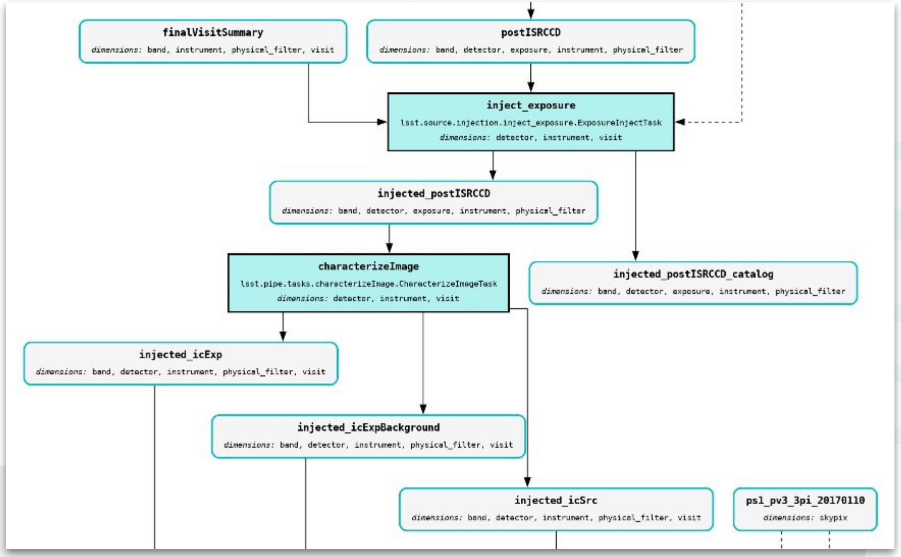
injected_deepCoadd
dimensions: band, patch, skymap, tract

injected_deepCoadd_catalog
dimensions: band, patch, skymap, tract

Making an injection pipeline

Static complete source injection pipeline definition YAML files do not exist.

Instead, source injection pipeline YAMLs can be generated (Python/command line) for a given dataset type using a reference pipeline and a given injection task YAML, optionally saving to a file:



postISRCCD
dimensions: band, detector, exposure, instrument, physical_filter

finalVisitSummary
dimensions: band, instrument, physical_filter, visit

inject_exposure
lsst.source.injection.inject_exposure.ExposureInjectTask
dimensions: detector, instrument, visit

injected_postISRCCD
dimensions: band, detector, exposure, instrument, physical_filter

characterizeImage
lsst.pipe.tasks.characterizeImage.CharacterizeImageTask
dimensions: detector, instrument, visit

injected_postISRCCD_catalog
dimensions: band, detector, exposure, instrument, physical_filter

injected_icExp
dimensions: band, detector, instrument, physical_filter, visit

injected_icExpBackground
dimensions: band, detector, instrument, physical_filter, visit

injected_icSrc
dimensions: band, detector, instrument, physical_filter, visit

psl_pv3_3pi_20170110
dimensions: skypix

```
make_injection_pipeline \
-t postISRCCD \
-r $DRP_PIPE_DIR/pipelines/HSC/DRP-RC2.yaml \
-i $SOURCE_INJECTION_DIR/pipelines/inject_exposure.yaml \
-f DRP-RC2+injection.yaml

Modified pipeline definition YAML file saved at /path/to/DRP-RC2+injection.yaml
```

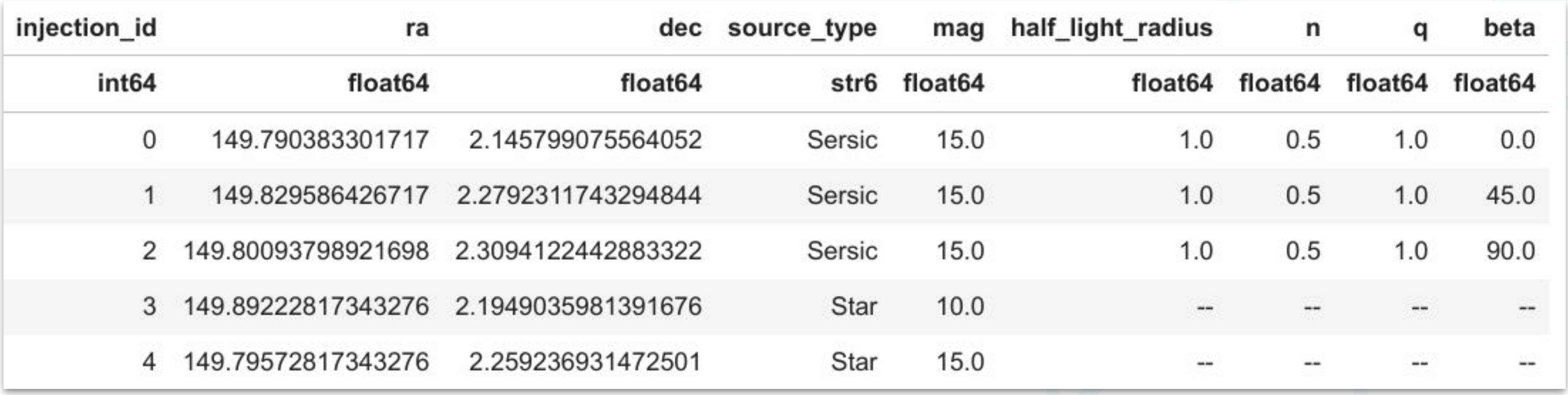
Swap the dataset type and injection YAML for visit or coadd injections as appropriate.

Injection catalogs

Injection catalogs now **natively support** all GalSim surface brightness profile classes: `Sersic, InclinedSersic, DeltaFunction, RandomKnots, …`

The `Star` alias for `DeltaFunction` has been retained.

We recommend stacking together multiple input catalogs using astropy vstack.



injection_id	ra	dec	source_type	mag	half_light_radius	n	q	beta
int64	float64	float64	str6	float64	float64	float64	float64	float64
0	149.790383301717	2.145799075564052	Sersic	15.0	1.0	0.5	1.0	0.0
1	149.829586426717	2.2792311743294844	Sersic	15.0	1.0	0.5	1.0	45.0
2	149.80093798921698	2.3094122442883322	Sersic	15.0	1.0	0.5	1.0	90.0
3	149.89222817343276	2.1949035981391676	Star	10.0	--	--	--	--
4	149.79572817343276	2.259236931472501	Star	15.0	--	--	--	--

Generating an injection catalog

You're free to make your own input catalog, or use the auto generation script to help get started.

Sources will be quasi-randomly scattered between RA and dec ranges.

GalSim model parameters must be given exactly as expected. The Sérsic source type expects magnitude, half light radius, Sérsic index, and optionally an axis ratio and position angle.

A number of combination repeats and output file can be optionally specified.

```
generate_injection_catalog \
-a 149.778 149.971 \
-d 2.134 2.327 \
-p source_type Sersic \
-p mag 15 17 19 \
-p half_light_radius 1 5 \
-p n 0.5 1 4 \
-p q 1 0.5 \
-p beta 0 45 90 \
-n 10 \
-f sersic_injection_catalog.fits
```

```
Generated an injection catalog containing
1080 sources: 108 combinations repeated 10
times.

Written injection catalog to
'sersic_injection_catalog.fits'.
```

Ingesting an injection catalog

The catalog ingestion utility can be used to ingest your input injection catalog into the data Butler for later use.

Provide the path to your input catalog, and the space-separated list of bands to be associated with it.

Finally, give the RUN collection where these data will be ingested.

```
ingest_injection_catalog \
-b $REPO \
-i sersic_injection_catalog.fits g r i \
-o u/lskelvin/pcw2023/inject_input

Ingested 2 g band injection_catalog
DatasetRefs into the butler.

Ingested 2 r band injection_catalog
DatasetRefs into the butler.

Ingested 2 i band injection_catalog
DatasetRefs into the butler.
```

Input catalogs will be sharded in the data Butler by band (provided) and HTM7 trixel ID (calculated based on the RA/Dec coordinates in the catalog).

Source injection

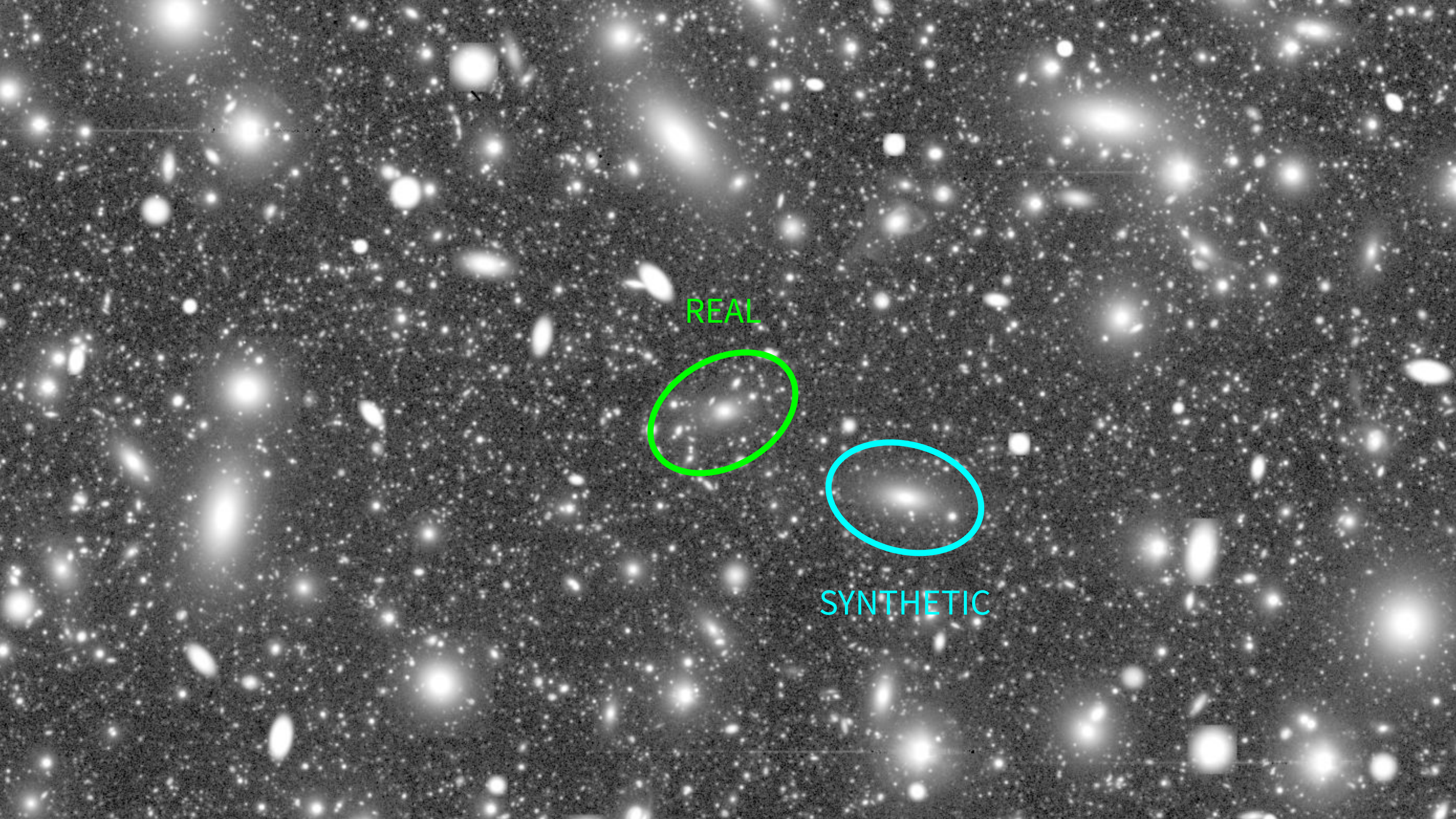
The pipetask run utility is used to inject your injection catalog into a dataset defined by a particular data query and store in a given output collection.

The process is the same for injection into exposure-types, visit-types or coadd-types.

So what does this look like?

```
DATA_COLL=HSC/runs/RC2/w_2023_23/DM-39610

INJECT_COLL=u/lskelvin/pcw2023/inject_input

pipetask --long-log --log-file $LOGFILE \
run --register-dataset-types \
--instrument lsst.obs.subaru.HyperSuprimeCam \
-b $REPO \
-i $DATA_COLL,$INJECT_COLL \
-o $OUTPUT \
-p DRP-RC2+injection.yaml#inject_coadd \
-d "instrument='HSC' AND skymap='hsc_rings_v1' AND
tract=9813 AND patch=42 AND band='i'"

Retrieved 1080 injection sources from 2 HTM trixels.

Generating 1080 injection sources consisting of 1
unique type: Sersic(1080).
```

HSC deepCoadd, tract 9813, patch 42, i-band


REAL
SYNTHETIC

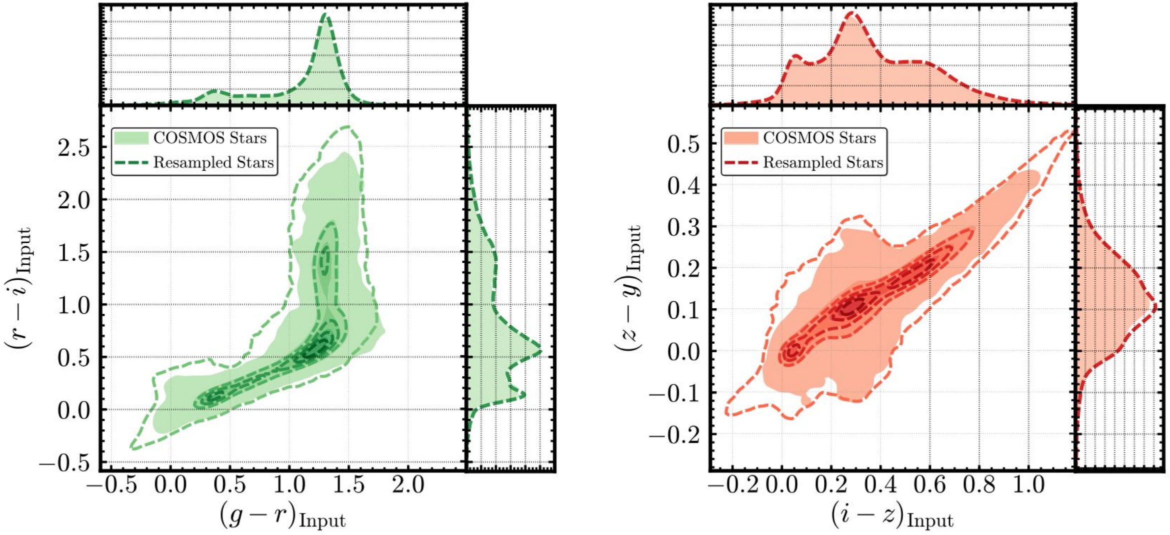
VERA C. RUBIN
OBSERVATORY

Regularly processed injection catalogs

We plan to replicate the synthetic samples described in Huang et al. 2018 and reduce these data regularly.

Using the SynPipe software, this study describes **~100,000 synthetic stars** (i ~ 19-26 mag) and **~58,000 synthetic Sérsic galaxies** (i ~ 20-25 mag).

This sample will be used to test algorithmic changes, astrometry, photometry, etc.

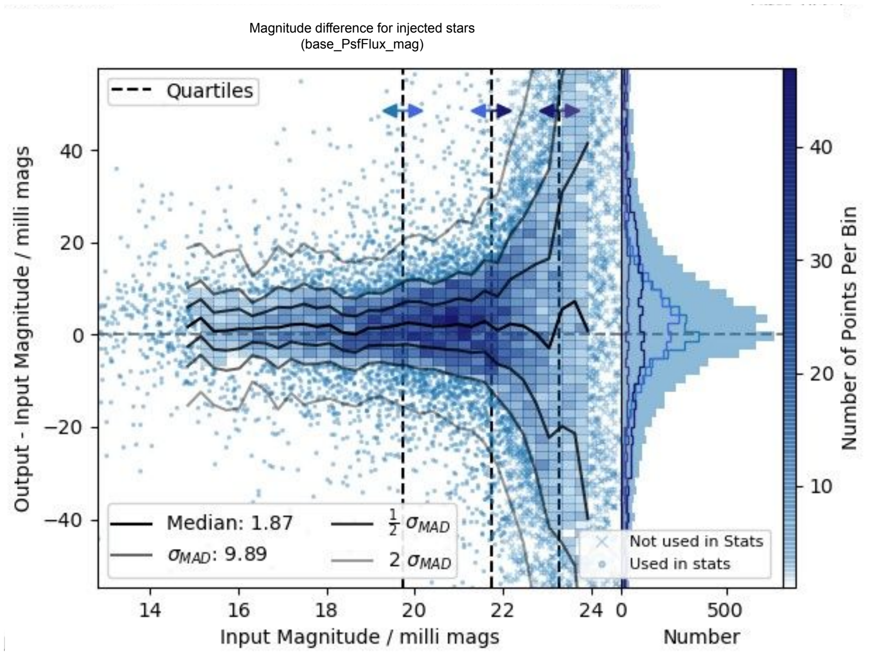


Color-color distribution of synthetic stars.
From Figure 2, Huang et al. 2018

Making source injection plots and metrics

We plan to use the analysis_tools package to **generate source injection plots/metrics** to help monitor our algorithmic and data reduction pipeline health.

Contributions from in-kind contributors and science collaboration members are welcome!



Legacy fakes mag. diff. measurements.
Image credit: Sophie Reed / DM

Future plans and Summary

The source_injection repo is now ***live***, enabling synthetic source injection into both **single-frame-level** and **coadd-level** imaging.

Many features of GalSim are now natively supported, allowing us to inject **simple** and **complex** synthetic sources or **user-supplied FITS imaging**, using utility functions for source/pipeline generation/ingestion.

Coming soon: the source_injection package will be **added into lsst_distrib**, with updated user-facing documentation.

We will **transition legacy fakes code** currently in use across the Science Pipelines the new package, deprecating old interfaces.

A series of **regularly reprocessed synthetic catalogs** based on Huang et al. 2018 will be ingested at the USDF as part of RC2 → analysis_tools will be used to **generate source injection plots/metrics** to help monitor the pipeline health.