

PyIMCOM Hack Session 2025

Speaker: Kaili Cao (OSU)

Date: August 4, 2025

Co-Authors: Christopher M. Hirata (OSU), Katherine Lalotis (OSU), Masaya Yamamoto (Princeton), Emily Macbeth (OSU),
M. A. Troxel (Duke)

Acknowledgements: Roman HLIS Cosmology PIT, NASA
Goddard Space Flight Center, Ohio Supercomputer Center

Simulated Image

Outline of this hack session

- What it is? – The Python implementation of Image Combination (PyIMCOM) package being developed for the Roman HLIS PIT.
 - High-level view: slides from my talk at Roman2024
 - Configuration entries: with focus on **inputs** and **outputs**
- How to use it? – Analyze its outputs, run it, and/or develop it.
 - Installation: [pyimcom](#), [furry-parakeet](#), dependencies
 - Demo for basic analyses of PyIMCOM outputs
 - Demo for running PyIMCOM on simulated images
- ~~What's on the inside?~~ – high-level structure, references

Installation of PyIMCOM

- `git clone https://github.com/kailicao/pyimcom.git`
- Dependencies: `fitsio`, `asdf`; `healpy`, `galsim`
 - Others are just standard packages for scientific computing.
 - If there are version conflicts, check `pyimcom_hack.yml`.
- Optional: install `furry-parakeet` and/or `mkl_fft`
 - Depending on the machine you use, these can be faster.
 - For running small tests, the standalone `pyimcom` is fine.
 - For analyzing PyIMCOM outputs, these are largely irrelevant.

pyimcom_hack_2025Aug4

- pyimcom/: a clone of the pyimcom repository
- S25-RUN/: for “demo for running PyIMCOM on simulated images”
 - L2/: input images and masks; psf6/: input PSFs
 - Roman_WAS_obseq_11_1_23.fits: input observation list
 - test25_F_00_00_demo.fits: output file of today’s test run
- sample/: for “demo for basic analyses of PyIMCOM outputs”
- config_test25-F.json: configuration file for today’s test run
- pyimcom_hack(_blank).ipynb: notebook for both demos
- pyimcom_hack.yml: my (KC’s) base environment

Why do we coadd images? – “Oversample, we must!”

- Roman point spread functions (PSFs)
 - = Airy disk ($\sim \lambda/D$) + linear obscuration + diffraction spikes + detector effects
 - An example in H158 band shown below

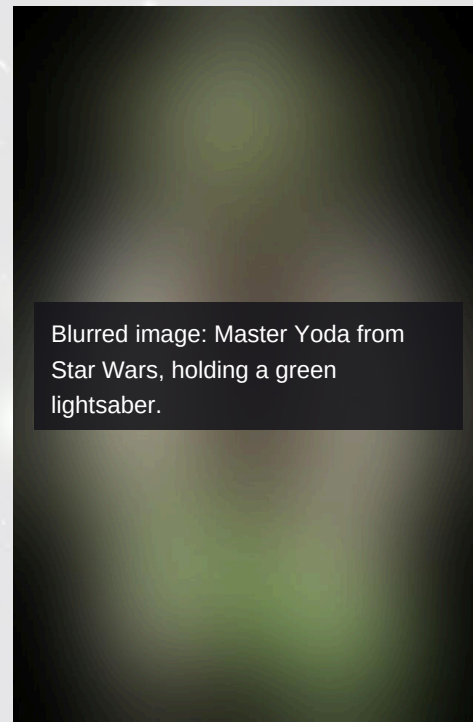
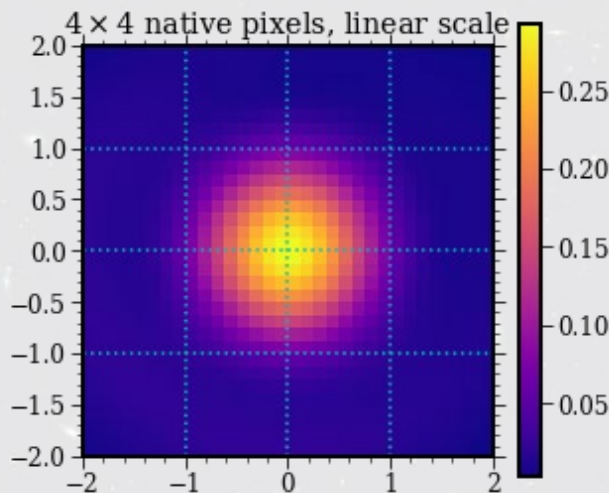
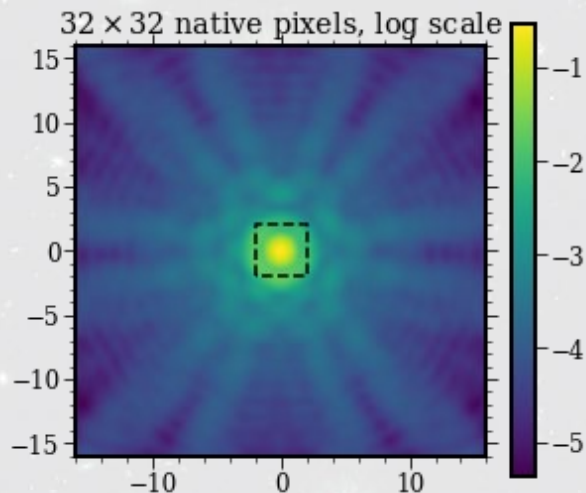


Image credits: Lucasfilm Ltd., Janson_G

How do we coadd images? – Through linear combinations.

- Linear combination of input **signals**:
$$H_{\alpha} = \sum_{i=0}^{n-1} T_{\alpha i} I_i$$
 - H : output signal, α : output pixel index, $\mathbf{R}_{\alpha}$: output pixel position
 - I : input signal, i : input pixel index, $\mathbf{r}_i$: input pixel position
 - T : coaddition weight, n : number of (selected) input pixels
- Linear combination of input **PSFs**:
$$\text{PSF}_{\alpha, \text{out}}(\mathbf{R}_{\alpha} - \mathbf{s}) = \sum_{i=0}^{n-1} T_{\alpha i} G_i(\mathbf{r}_i - \mathbf{s})$$
 - PSF_{out} : reconstructed output PSF, Γ : target output PSF
 - IMCOM minimizes discrepancy between PSF_{out} and Γ (“leakage”).

Backup: Major operations of the IMCOM software

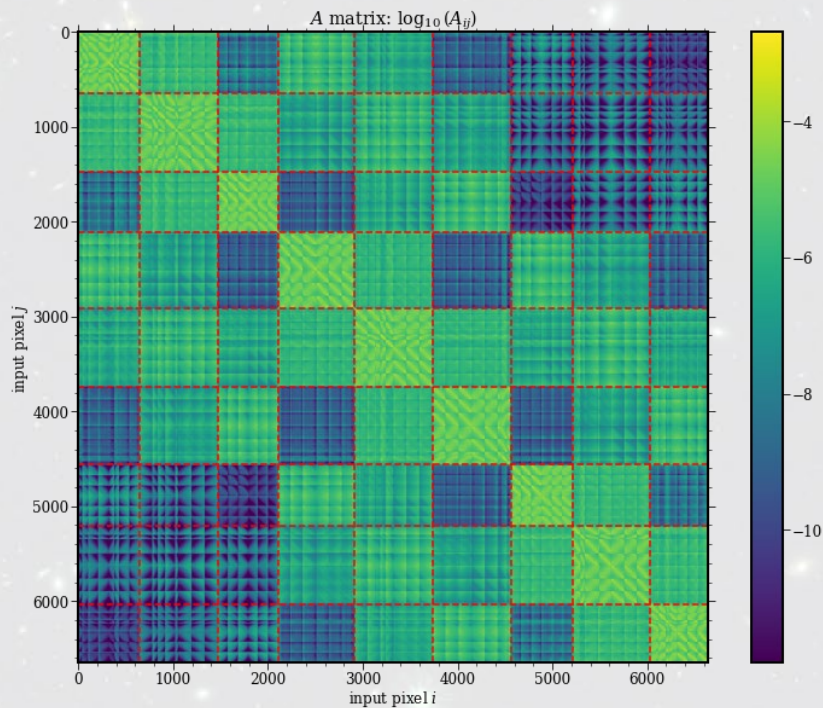
- Read input data, principally input signals (I_i), pixel positions ($\mathbf{r}_i$), and PSFs (G_i); parse configuration to get output pixel positions ($\mathbf{R}_\alpha$) and the target PSF (Γ).
- Perform fast Fourier transform (FFT) and inverse FFT to compute PSF overlaps ($G_j \otimes G_i$, $\Gamma \otimes G_i$, and $C = ||\Gamma||^2$).
- Perform interpolations (see Appendix A of Hirata *et al.* 2024 for details) using pixel positions to obtain system matrices ($\mathbf{A}$ and $\mathbf{B}$).
- Solve linear systems to get the optimal Lagrange multiplier κ_α and coaddition weights $T_{\alpha i}$ for each output pixel.
- Compute the output map (H_α), and report diagnostics for its quality (“PSF leakage” U_α/C and “noise amplification” Σ_α).

High-level structure of pyimcom core modules

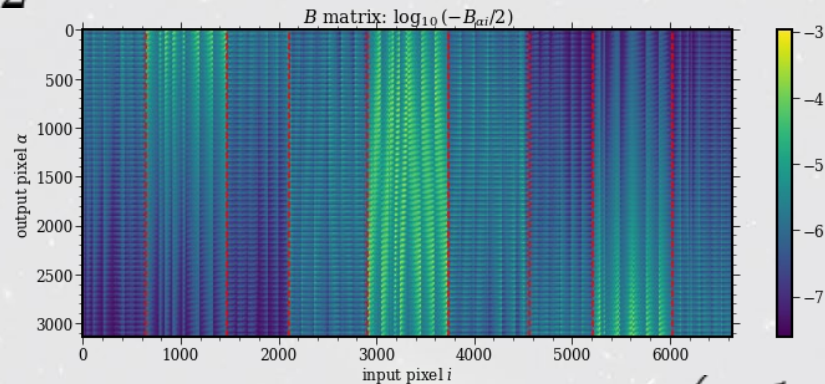
- Copied from “[Roman IMCOM Notes](#)” Slack Canvas
- config.py: Encapsulation of pyimcom background settings and configuration.
- coadd.py: Driver to coadd a block (2D array of postage stamps).
 - layer.py: Utilities to generate additional layers and handle masks.
 - psfutil.py: Utilities for PSFs and system matrices.
 - lakernel.py: Linear algebra kernels to solve linear systems.
- analysis.py: Tools to analyze coadded images.

Backup: IMCOM formalism and example matrices

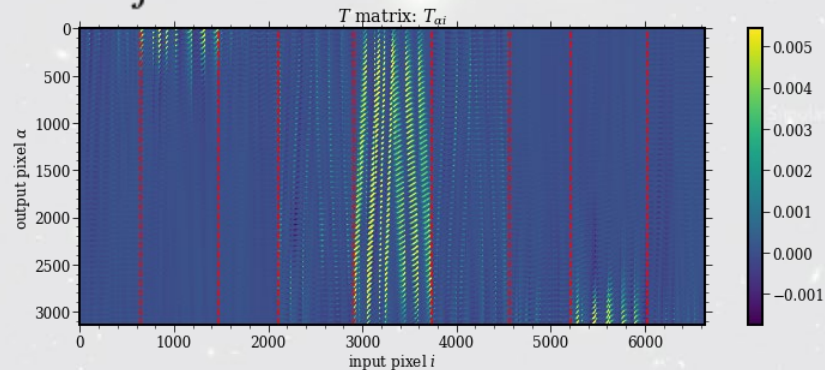
$$A_{ij} = [G_j \otimes G_i](\mathbf{r}_i - \mathbf{r}_j)$$



$$-\frac{1}{2}B_{\alpha i} = [\Gamma \otimes G_i](\mathbf{r}_i - \mathbf{R}_\alpha)$$

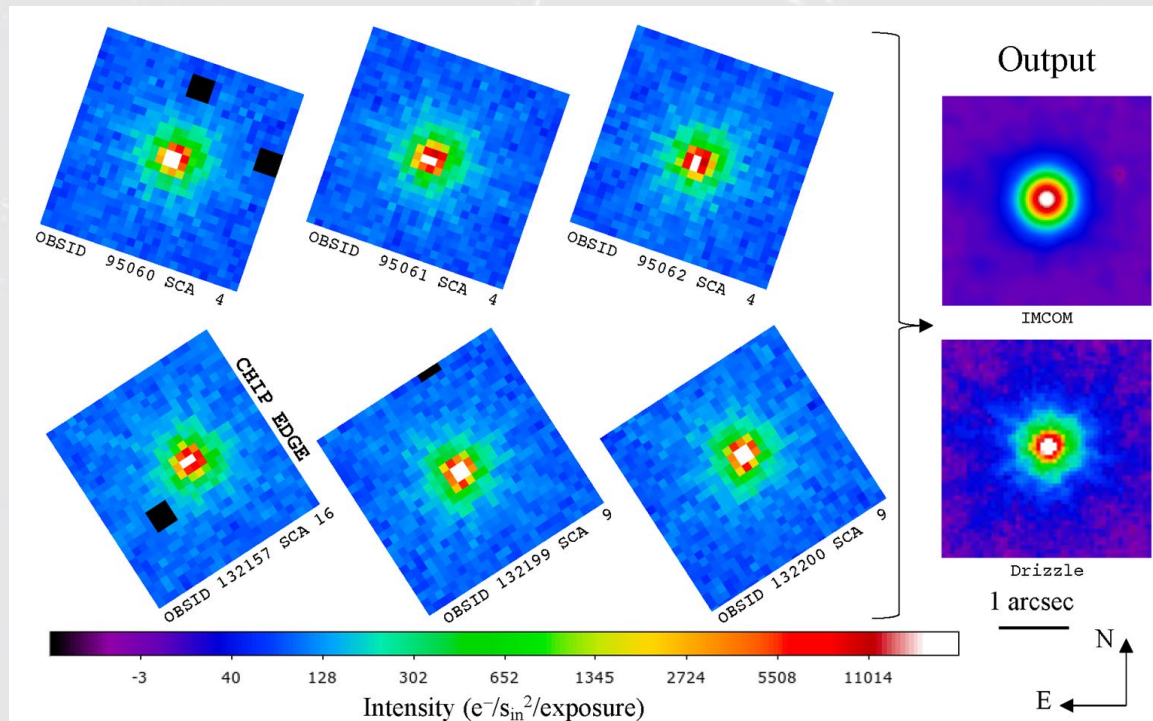


$$T_{\alpha i} = \sum_j [(\mathbf{A} + \kappa_\alpha \mathbf{I}_{n \times n})^{-1}]_{ij} \left(-\frac{1}{2}B_{\alpha j} \right)$$



IMCOM vs. Drizzle: With vs. without control over output PSF

- Hirata *et al.* (2024) Figure 9
- Drizzle (Fruchter & Hook 2002): What your telescope sees is what you get
- IMCOM (Rowe *et al.* 2011): What you get is closest to what you want to see



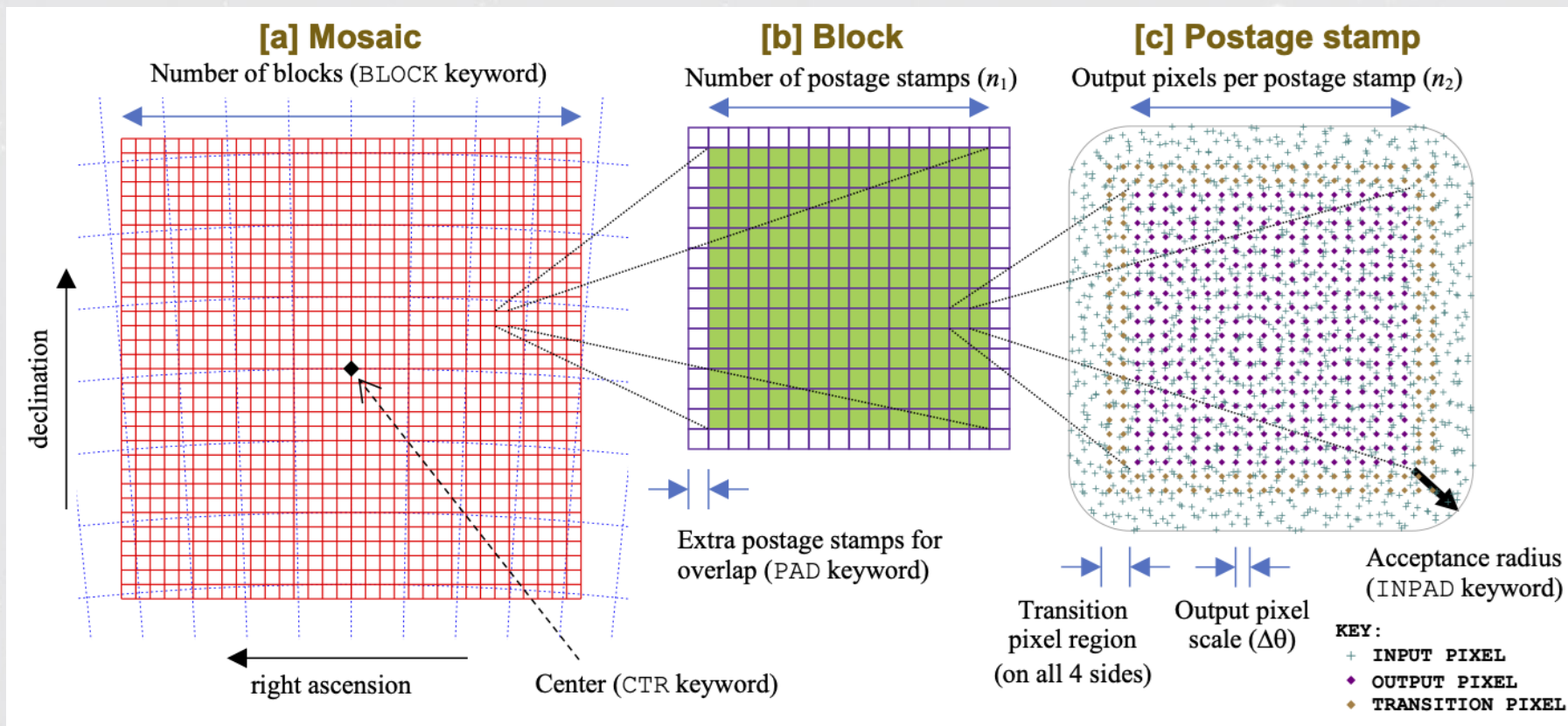
CONFIGURATION SECTION I: INPUT FILES

- **[input]** OBSFILE: input observation list
- **[input]** INDATA: reference input file directory and naming convention
 - Currently supported formats: dc2_imsim, anlsim, L2_2506
- **[input]** FILTER: 1 = F184, 2 = H158, 3 = J129, 4 = Y106, 10 = K213
- **[input]** INPSF: input PSF file directory, format, and oversampling rate
 - Currently supported formats: dc2_imsim, anlsim, L2_2506
- **[technical]** PSFSPLIT: PSF splitting (feature under development?)

CONFIGURATION SECTION II: MASKS AND LAYERS

- [technical] PMASK: permanent mask file
- [technical] CMASK: CR hit probability for stochastic mask
- [input] EXTRAINPUT: input images to stack at once
 - Images: science (`None` in `extrainput`), truth
 - Noise frames: white, 1/f, lab, romanimpreprocess
 - Injected stars: GalSim, internal; noisy, transient, field dependent
 - Injected galaxies: GalSim
- [technical] LABNOISETHRESHOLD: threshold for lab noise mask

Figure 4 in [Hirata et al., MNRAS 528:2533 \(2024\)](#)



CONFIGURATION SECTION III: WHAT AREA TO COADD

- [output] CTR: tile center in degrees RA, DEC
- [output] LONPOLE: longitude of the celestial pole
- [output] BLOCK: number of blocks on each side of the mosaic
- [output] OUTSIZE: number of postage stamps on each side of a block, number of output pixels on each side of a postage stamp, output pixel scale in arcsec

CONFIGURATION SECTION IV: MORE ABOUT POSTAGE STAMPS

- [technical] FADE: fading kernel width
- [output] PAD: pad this many IMCOM postage stamps around the edge
 - Objects centering in padding regions usually need to be excluded.
- [output] PADSIDES: on which side(s) to pad IMCOM postage stamps
 - For production runs, use 'auto' to claim a 10%-level speed up for free! (Sharing repeated postage stamps basically takes no time.)
- [technical] STOP: stop after this number of postage stamps

CONFIGURATION SECTION V: WHAT AND WHERE TO OUTPUT

- [output] OUTMAPS: which IMCOM diagnostics to report
 - U = PSF leakage map (U_alpha/C)
 - S = noise amplification map
 - [technical] K = kappa (Lagrange multiplier map)
 - T = total weight (sum over all input pixels)
 - N = effective coverage
- [output] OUT: output stem
- [technical] TEMPFILE: temporary storage
- [technical] INLAYERCACHE: cache input layers here

CONFIGURATION SECTION VI: TARGET OUTPUT PSF(S)

- [technical] NOUT: number of target output PSF(s)
- [output] OUTPSF: target output PSF type
 - Currently supported types: GAUSSIAN, AIRY(UN)OBSC
- [output] EXTRASMOOTH: target output PSF extra smearing
 - For GAUSSIAN, this is just its width (so not extra).
 - For AIRY(UN)OBSC, this is the width of the Gaussian component.
- [technical] OUTPSF2, EXTRASMOOTH2, ...
 - If you want to run more than one target output PSFs at once.

CONFIGURATION SECTION VII: BUILDING LINEAR SYSTEMS

- [technical] NPIXPSF: width of PSF sampling/overlap arrays
- [technical] PSFCIRC, PSFNORM: experimental features
- [technical] AMPPEN, FLATPEN: experimental features
- [input] INPAD: input stamp size padding (aka acceptance radius)
 - This determines number of input pixels and time consumption.

CONFIGURATION SECTION VIII: SOLVING LINEAR SYSTEMS

- [technical] LAKERNEL: kernel to solve linear systems
 - The Cholesky kernel is recommended.
- [technical] ITERRTOL, ITERMAX: parameters for the iterative kernel
- [technical] EMPIRNQC: parameter for the empirical kernel
- [technical] KAPPAC: list of kappa/C values, ascending order
 - Using a single value is recommended.
- [technical] UCMIN: target (minimum) leakage
- [technical] SMAX: maximum allowed value of Sigma

References (from [README.rst](#))

- Optimal Linear Image Combination. [Rowe et al. ApJ 741:46 \(2011\)](#)
- Simulating image coaddition with the Nancy Grace Roman Space Telescope - I. Simulation methodology and general results. [Hirata et al. MNRAS 528:2533 \(2024\)](#)
- Simulating image coaddition with the Nancy Grace Roman Space Telescope - II. Analysis of the simulated images and implications for weak lensing. [Yamamoto et al. MNRAS 528:6680 \(2024\)](#)
- Analysis of Biasing from Noise from the Nancy Grace Roman Space Telescope: Implications for Weak Lensing. [Laliotis et al. PASP 136:124506 \(2024\)](#)
- Simulating image coaddition with the Nancy Grace Roman Space Telescope: III. Software improvements and new linear algebra strategies. [Cao et al., ApJS accepted, arXiv: 2410.05442](#)
- OpenUniverse2024: A shared, simulated view of the sky for the next generation of cosmological surveys. [OpenUniverse et al. arXiv: 2501.05632](#)