The Astronomical Journal, 168:174 (29pp), 2024 October https://doi.org/10.3847/1538-3881/ad6a0f

Efficient Point-spread Function Modeling with ShOpt.jl: A Point-spread Function Benchmarking Study with JWST NIRCam Imaging

Edward M. Berman[1], Jacqueline E. McCleary[1], Anton M. Koekemoer[2], Maximilien Franco[3], Nicole E. Drakos[4], Daizhong Liu[5], James W. Nightingale[6], Marko Shuntov[7], Diana Scognamiglio[8], Richard Massey[9], Guillaume Mahler[6,9], Henry Joy McCracken[10], Brant E. Robertson[11], Andreas L. Faisst[12], Caitlin M. Casey[3,13], and Jeyhan S. Kartaltepe[14]

COSMOS-Web: The JWST Cosmic Origins Survey

[1] Northeastern University, 100 Forsyth St., Boston, MA 02115, USA; berman.ed@northeastern.edu
[2] Space Telescope Science Institute, 3700 San Martin Dr., Baltimore, MD 21218, USA
[3] The University of Texas at Austin, 2515 Speedway Blvd. Stop C1400, Austin, TX 78712, USA
[4] Department of Physics and Astronomy, University of Hawaii, Hilo, 200 W Kawili St., Hilo, HI 96720, USA
[5] Max-Planck-Institut für Extraterrestrische Physik (MPE), Giessenbachstr. 1, D-85748 Garching, Germany
[6] Department of Physics, Institute for Computational Cosmology, Durham University, South Rd., Durham DH1 3LE, UK
[7] Institut d'Astrophysique de Paris, CNRS, Sorbonne Université, 98bis Blvd. Arago, 75014, Paris, France
[8] Jet Propulsion Laboratory, California Institute of Technology, 4800, Oak Grove Dr., Pasadena, CA 91109, USA
[9] Department of Physics, Centre for Extragalactic Astronomy, Durham University, South Rd., Durham DH1 3LE, UK
[10] Institut d'Astrophysique de Paris, UMR 7095, CNRS, and Sorbonne Université, 98 bis Blvd. Arago, F-75014 Paris, France
[11] Department of Astronomy and Astrophysics, University of California, Santa Cruz, 1156 High St., Santa Cruz, CA 95064, USA
[12] Caltech/IPAC, 1200 E. California Blvd., Pasadena, CA 91125, USA
[13] Cosmic Dawn Center (DAWN), Denmark
[14] Laboratory for Multiwavelength Astrophysics, School of Physics and Astronomy, Rochester Institute of Technology, 84 Lomb Memorial Dr., Rochester, NY 14623, USA

Received 2024 January 21; revised 2024 July 21; accepted 2024 July 29; published 2024 September 24

Abstract

With their high angular resolutions of 30–100 mas, large fields of view, and complex optical systems, imagers on next-generation optical/near-infrared space observatories, such as the Near-Infrared Camera (NIRCam) on the James Webb Space Telescope, present new opportunities for science and also new challenges for empirical point-spread function (PSF) characterization. In this context, we introduce `ShOpt`, a new PSF fitting tool developed in Julia and designed to bridge the advanced features of PSFs in the full field of view (PIFF) with the computational efficiency of PSF Extractor (PSFEx). Along with `ShOpt`, we propose a suite of nonparametric statistics suitable for evaluating PSF fit quality in space-based imaging. Our study benchmarks `ShOpt` against the established PSF fitters PSFEx and PIFF using real and simulated COSMOS-Web Survey imaging. We assess their respective PSF model fidelity with our proposed diagnostic statistics and investigate their computational efficiencies, focusing on their processing speed relative to the complexity and size of the PSF models. We find that `ShOpt` can already achieve PSF model fidelity comparable to PSFEx and PIFF while maintaining competitive processing speeds, constructing PSF models for large NIRCam mosaics within minutes.

Unified Astronomy Thesaurus concepts: Computational methods (1965); Astronomy image processing (2306); Astronomy data analysis (1858); James Webb Space Telescope (2291)

1. Introduction

The inherent limitations of optical systems introduce artifacts into telescope imaging. Effects like diffraction, optical aberrations, atmospheric turbulence (if applicable), and telescope jitter are summarized in the telescope's point-spread function (PSF): the response of the optical system to an idealized point of light. Good PSF modeling during image processing reduces the impact of things like atmospheric turbulence and optical aberrations during scientific analysis. Failure to model the PSF correctly can lead to inaccurately measured positions, sizes, and shapes of small targets like galaxies.

The central importance of PSF characterization to astrophysics means that there is a wealth of PSF fitters available. These generally fall into two classes: forward-modeling approaches, which use physical optics propagation based on models of optical elements, and empirical approaches, which use observed stars as fixed points to model and interpolate the PSF across the rest of the image. In both cases, the PSF model may be validated by comparing a set of reserved stars to the PSF model's prediction.

Empirical characterization tools like PSF Extractor (PSFEx; Bertin 2011) and PSFs in the full field of view (PIFF; Jarvis et al. 2020)[15] are widely popular in astrophysics. However, the quality of PIFF and PSFEx models tends to be quite sensitive to the values of hyperparameters used to run the software, with optimal parameter selection sometimes relying on brute-force guess-and-check runs. PIFF, using the modeling and interpolation scheme used for the Dark Energy Survey Year 3 observations, is also notably inefficient for large, well-sampled images, taking hours in the worst cases.

Because space telescopes are unimpeded by the atmosphere, their PSFs are also often characterized by forward-modeling approaches like Tiny Tim (Krist et al. 2011) and WebbPSF

[15] https://github.com/astromatic/psfex and https://github.com/rmjarvis/Piff.

(Perrin et al. 2012, 2014; Ji et al. 2023). WebbPSF models are continually updated based on telescope telemetry, ensuring high accuracy in all bandpasses and for all instruments regardless of image noise. While robust, forward modeling is not infallible and may occasionally miss short-timescale variations and other "unknown unknowns" that can be captured with empirical PSF models, albeit at the cost of much higher noise.

The James Webb Space Telescope (JWST) represents a giant leap forward in our ability to explore the cosmos. Equipped with groundbreaking instruments like the Near Infrared Camera (NIRCam) and the Mid-Infrared Imager (MIRI), the telescope is poised to unlock unprecedented insights into the early universe (Robertson 2022). At the same time, these advances usher in a new set of complexities for PSF characterization:

1. Modeling the PSFs of space observatories like JWST is nontrivial since they generally operate near their diffraction limits and exhibit intricate optical patterns that defy the analytic approximations acceptable for PSFs of ground-based imaging. In particular, the PSFs exhibit steep spatial gradients, making them highly sensitive to the hyperparameters of commonly used analytic profiles, where slight variations can lead to substantial inaccuracies. The assessment of PSF models for space-like imaging presents an additional challenge, as most metrics of fit quality, e.g., FWHM or second moments of intensity, treat the PSF as an elliptical Gaussian—a poor approximation for the "spiky" NIRCam and MIRI PSFs. While there are some useful moment-based metrics for assessing fit quality (e.g., Zhang et al. 2023), there are no universally adopted nonparametric diagnostics for assessing pixel-level model biases.
2. The NIRCam detector pixel scales are $0\farcs031\,\mathrm{pixel}^{-1}$ and $0\farcs063\,\mathrm{pixel}^{-1}$ for the short- and long-wavelength channels, respectively (Rieke et al. 2003, 2005; Beichman et al. 2012). At these fine scales, fully capturing the intensity profile of the NIRCam PSF requires a much larger number of pixels than is needed for typical surveys, whose detectors have pixel scales 3–10 times larger than NIRCam (York et al. 2000; Jarvis et al. 2020; Fu et al. 2022; McCleary et al. 2023). As a consequence, the number of pixels needed to model the full size of the PSF is much greater. This new regime prompts a necessary evaluation of how current PSF fitters perform at this new scale. That is, can the PSF fitters still capture the full dynamic range of the distortion and do so in a reasonable time.

To meet these challenges, we introduce `ShOpt`,[16] a new PSF modeling tool that strives to retain the best of existing PSF modeling software while advancing their mathematical formulation and increasing their computational efficiency. Written in the high-level Julia language using a functional programming style,[17] `ShOpt` offers both accessibility and speed, positioning it as a valuable tool for the astrophysical community. `ShOpt` introduces manifold-based algorithms for enhanced efficiency in analytic profile fitting. `ShOpt` also employs three distinct techniques for pixel-basis fitting: principal component analysis (PCA), an autoencoder, and kernel smoothing.

Along with `ShOpt`, we introduce a suite of nonparametric PSF fit statistics appropriate for space-based imaging, namely reduced χ^2, mean relative error, and mean absolute error. Our proposed PSF characterization statistics are inspired by strong gravitational lensing analyses and move beyond the conventional metrics based on elliptical Gaussians. In this study, we evaluate the performance of `ShOpt`, PSFEx, and PIFF using data from the COSMOS-Web survey (Casey et al. 2023), using our proposed PSF fit statistics to gauge their relative performance. In addition, we time the PSF fitters to measure the computational efficiency of each tool.

To summarize, our contributions are the following:

1. We introduce `ShOpt`, and with it, a number of methods for efficient empirical PSF characterization.
2. We benchmark the model fidelity of different PSF fitters using nonparametric approaches. We use real and simulated catalogs from the COSMOS-Web Survey and measure χ^2, mean relative error, and mean absolute error. We supplement these statistics with conventional second-moment Hirata-Seljiak-Mandelbaum (HSM) fits (Hirata & Seljak 2003; Mandelbaum et al. 2005).
3. We benchmark the computational efficiency of different PSF fitters.

The remainder of this paper is structured as follows. In Section 2, we establish the workflow of `ShOpt` and the notation used in this paper. In Section 3, we develop our methods for Gaussian and pixel-basis PSF fits, and in Section 4 our methods for fitting their variation across the field of view. In Section 5, we describe our benchmarking methods and data sets. Our algorithmic choices for `ShOpt` are justified using big-$\mathcal{O}$ time complexity analysis in Section 6, specifically detailing `ShOpt`'s speed variation with different input parameters. In Section 7, we describe our benchmarking methods and data sets. We present our results in Section 8, and discussion and conclusions in Section 9.

2. ShOpt Notation, Workflow, and Overview

2.1. Notation and Preliminaries

We represent star locations in pixel coordinates as (x, y) and in celestial coordinates (expressed in degrees) as (u, v). In terms of a nominal position source, positive u is west, and positive v is to the north. "Vignette" refers to small, localized images, centered around individual stars or celestial objects, that are extracted from larger astronomical images.

We use the following notation for working with manifolds. For two sets A, B,

$$B_2 \equiv \{[x, y]\colon x^2 + y^2 < 1\} \subset \mathbb{R}^2, \tag{1}$$

$$\mathbb{R}_+ \equiv \{x\colon x > 0\} \subset \mathbb{R}, \tag{2}$$

$$A \times B \equiv \{(a, b)\colon a \in A, b \in B\}. \tag{3}$$

Throughout this paper, we describe the shape of a smoothly varying (analytic) PSF profile in terms of the variables $[s, g_1, g_2]$, where (g_1, g_2) are the polarization states of an elliptical source with reduced shear $\boldsymbol{g} = g_1 + ig_2 = ge^{i2\theta}$, where θ is the angle of the major axis from [west/some

[16] The name ShOpt is a contraction of "shear optimization." That is, we are finding the best-matching PSF by formulating optimization problems over the space of all possible shears. The name was inspired by the `manopt` library, a contraction of manifold optimization. It is an apt comparison given the manifold learning we describe in Section 3.1.

[17] By functional, we mean the functional programming paradigm. While Julia provides support for object-oriented design patterns with `structs`, the Julia code is generally written with design patterns that put an emphasis on reusable functions.

fiducial orientation] and

$$g \equiv \frac{1-q}{1+q}, \tag{4}$$

for an ellipse with (major/minor) axis ratio q, such that $0 \leqslant g < 1$ (Bernstein & Jarvis 2002). While shear is typically treated as a number in the complex plane $\mathbb{C}$, we make an equivalent characterization that shear is a vector $[g_1, g_2] \in \mathbb{R}^2$. This vector representation corresponds to the real and imaginary parts of the complex shear. The free parameter s represents the size of the ellipse (the geometric mean of the major and minor axis lengths). We introduce free parameters $[\sigma, e_1, e_2]$ that reparameterize $[s, g_1, g_2]$, noting that we adopt a slightly different relationship between ellipticity e and shear g than may be seen elsewhere in the literature, dropping by a factor of 2 in Equation (13) for the purposes of an easier calculation of the inverse map (details in Section 3).

Additionally, we are often concerned with a more realistic pixel basis, where each pixel in an $n \times n$ star image is a basis element.

For our nonparametric summary statistics defined in Equations (5)–(8), we use $v_{i,j,k}$ to represent the $i \times j$ pixels in vignette k, and $p_{i,j,k}$ to represent the $i \times j$ pixels in PSF model k. $\sigma_{i,j,k}$ represents the uncertainty in the model at pixel (i, j) and vignette k. The total number of vignettes is K. The number of pixels in a vignette is denoted N_{pix}

$$\overline{\chi^2}_\nu = \frac{1}{K}\sum_k \frac{1}{\text{d.o.f.}} \sum_{(i,j)}^{N_{\text{pix}}} \frac{(v_{i,j,k} - p_{i,j,k})^2}{(\sigma_{i,j,k})^2}, \tag{5}$$

$$\text{median } \chi^2_\nu = \text{median}_K \left[\frac{1}{\text{d.o.f.}} \sum_{(i,j)}^{N_{\text{pix}}} \frac{(v_{i,j,k} - p_{i,j,k})^2}{(\sigma_{i,j,k})^2} \right], \tag{6}$$

$$\text{Mean Relative Error} = \frac{1}{N_{\text{pix}}} \sum_{(i,j)}^{N_{\text{pix}}} \frac{1}{K}\sum_k \frac{v_{i,j,k} - p_{i,j,k}}{v_{i,j,k}}, \tag{7}$$

$$\text{Mean Absolute Error} = \frac{1}{N_{\text{pix}}} \sum_{(i,j)}^{N_{\text{pix}}} \frac{1}{K}\sum_k \frac{|v_{i,j,k} - p_{i,j,k}|}{|v_{i,j,k}|}. \tag{8}$$

We shall henceforth refer to the mean relative error as MRE and the mean absolute error as MAE. To enable flux-based comparisons between PSF vignettes, which most fitters normalize to unity, and star vignettes, we add the appropriate star flux to the PSF vignettes and then add Gaussian noise with mean and variance taken from the sky background in the vicinity of the star.

2.2. Code Overview

`ShOpt` takes inspiration from robotics algorithms, such as SE-Sync (Rosen et al. 2019), that run on manifold-valued data. The manifold properties of shears are described in Bernstein & Jarvis (2002); we expand on their work to provide more robust multivariate analytic fits to PSF intensity profiles. We specifically use multivariate Gaussians, as they are cheap to compute, but for a more rigorous treatment of optimization methods on manifold-valued data, see Absil & Mahony (2008) and Boumal (2023). `ShOpt` also provides three modes for fitting PSFs on a pixel basis: `PCA` mode, `autoencoder` mode, and `smoothing` mode detailed in Section 3. `PCA` mode approximates the original image by summing the first n principal components, where n is supplied by the user. We also introduce `autoencoder` mode, which uses a neural network with an autoencoder architecture to learn the PSF. A square image of side length n can be thought of as vectors in $\mathbb{R}^{n^2}$, where each pixel is a basis element. The architecture is built so that the image is encoded into a vector space represented by a basis with dim $(V) < n^2$ before being decoded back into the dimension of our original image. The nonlinearity of the network ensures that the key features are learned instead of some linear combination of the sky background. Both `PCA` and `autoencoder` modes provide the end user with tunable parameters that allow for accurate reconstruction of the model vignettes without overfitting to noise. The `smoothing` mode applies a Lanczos kernel to the input vignettes and uses the smoothed output for the pixel-basis fit.

Why Julia?

`ShOpt` is written in Julia. The Julia programming language is a high-level and functional language like Python, which makes it accessible to a community of open-source developers. At the same time, Julia is equipped with a just-in-time compiler, which helps Julia code execute quickly by recycling compiled code. This offers a speed advantage over Python, which is first transpiled to C before being translated into machine code. Julia also offers some of the most sophisticated tools for problems at the intersection of numerical linear algebra and optimization, such as the `ForwardDiff.jl` and `Optim.jl` (Revels et al. 2016; Mogensen & Riseth 2018) libraries. Julia also has an abundance of support for working with manifolds such as `manopt`, which may be pertinent in future releases of `ShOpt` (Bergmann 2022). Finally, much of today's production code is written with the help of program synthesis tools such as GitHub Copilot. It is clear that these tools will soon make Julia more accessible than ever before to astrophysicists and other potential future `ShOpt` contributors. Work is being done to strengthen these tools for programming languages with much less training data available, such as `Julia`. Cassano et al. (2023) successfully demonstrated how to transfer knowledge from programming languages with lots of publicly available training data (e.g., Python) to programming languages with much less training data available (e.g., Racket, OCaml, Lua, R, and most importantly, Julia).

The source code for `ShOpt` can be found on GitHub[18] and is detailed in our companion paper (Berman & McCleary 2024).

`ShOpt` accepts as input a FITS-format catalog containing star vignettes, positions, and signal-to-noise ratios (SNRs). One concession `ShOpt` makes is that it makes more sense to build on the `Astropy` infrastructure than to rebuild everything from scratch. As such, `PyCall.jl` is used to handle FITS input and output. Since this is only done once and not iterated over in any main loop, this does not significantly affect performance.

After an initial quality check based on SNR, the star vignettes are fit with a multivariate Gaussian to remove outlier stars. While these Gaussian fits are not used to create a PSF model, they are helpful in screening out stars that are saturated, too bright, or too faint. In situations where the image noise is high, the SNR threshold must be kept low to include enough stars for fitting. After these preprocessing steps, stars are fit in the pixel basis using either the `PCA` or autoencoder modes. The output learned stars are then fit with a multivariate Gaussian, which serves both to reject bad pixel-basis fits and to record the rough size and shape of the PSF. Surviving stars act as fixed

[18] https://github.com/EdwardBerman/shopt and https://edwardberman.github.io/shopt/.

points for polynomial interpolation of the PSF's spatial variation across the field of view. Finally, the learned data are saved to a summary FITS file, along with diagnostic plots and logging statements. This is summarized in Algorithm 1.

Algorithm 1. ShOpt Workflow

```
starCatalog ← loadStarCatalog()
filteredCatalog ← filterBySignalToNoise(starCatalog)
for each vignette in filteredCatalog do
    gaussianFit ← fitMultivariateGaussian(vignette)
    if not isGoodFit(gaussianFit) then
        remove(vignette)
    end if
end for
pixelGrid ← createPixelGrid(filteredCatalog)
reducedData ← dimensionalityReduction(pixelGrid)
for each grid in reducedData do
    gaussianFit ← fitMultivariateGaussian(grid)
    if not isGoodFit(gaussianFit) then
        remove(grid)
    end if
end for
for num iterations do
    for each point in FieldOfView do
        interpolatedStar ← interpolate(point, reducedData)
        if isOutlier(interpolatedStar) then
            remove(Star)
        end if
    end for
end for
plotAndSaveData()
```

3. Parameter Estimation for PSF Modeling

3.1. Analytic Profile Fitting

`ShOpt`'s analytic model has two components, a shear transformation and a radial function. We elect to fit an elliptical Gaussian $f_{\text{Gaussian}}(r)$ for our radial function:

$$f_{\text{Gaussian}}(r) = Ae^{-r^2/2}, \tag{9}$$

where A makes the image sum to unity. There are other radial profiles that we could choose from, however, any radial profile parameterized by $[s, g_1, g_2]$ contains some azimuthal symmetry that makes the "wings" of the PSF difficult to model. Thus, it is not worth the computational cost to fix a radial profile that is any more elaborate than an elliptical Gaussian as there are diminishing returns for accuracy.

For any analytic model, the shear is always the same,

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \frac{s}{\sqrt{1-(g_1)^2-(g_2)^2}} \begin{bmatrix} 1+g_1 & g_2 \\ g_2 & 1-g_1 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{10}$$

and

$$r = \sqrt{(u')^2 + (v')^2}. \tag{11}$$

The shear matrix in Equation (10) is known to be positive definite, and s is strictly positive. Therefore, our parameters are constrained to $\mathbb{R}_+ \times B_2$. The parameter space can be visualized as a solid cylinder extending infinitely far from the origin in one direction, as seen in Figure 1.

The finite domain of g poses a problem for fitting. As such, we adopt a fitting parameter e that allows us to map any value

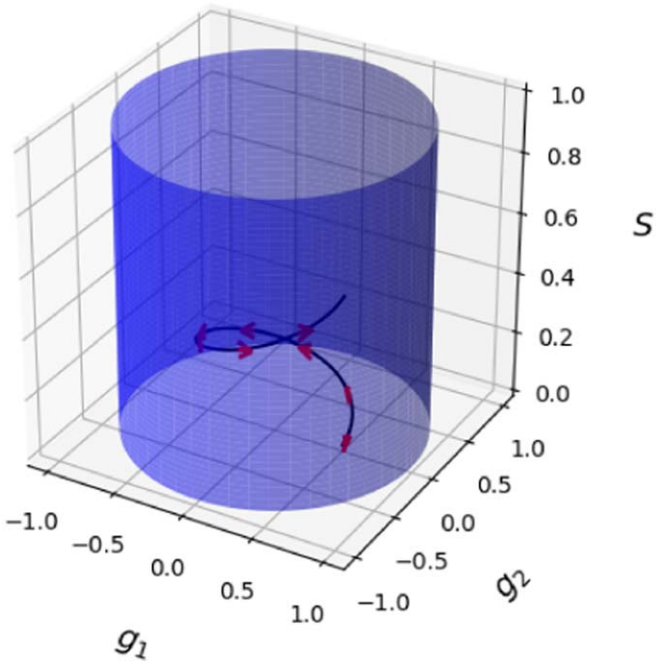


Figure 1. Parameter space for fitting an analytic profile, with sample iterations of the algorithm converging toward a learned $[s, g_1, g_2]$. Note that the cylinder extends upward toward infinity but is bounded from below by 0.

in $\mathbb{R}^2$ onto g:

$$e_i = \frac{g_i}{1 - ||g||^2}. \tag{12}$$

This map[19] from $B_2 \to \mathbb{R}^2$ is relatively smooth so that any automatic differentiation software can handle it during the fitting process. We can construct an inverse map via

$$||g||^2 = \frac{2||e||^2}{1 + 2||e||^2 + \sqrt{1 + 4||e||^2}}. \tag{13}$$

The inverse map allows us to constrain our update steps inside of our parameter space, which leads to quicker convergence and the ability to handle noisier data more effectively. Equation (13) also keeps the nice properties of smoothness and monotonicity desirable for activation functions; see Figure 2. Equation (13) is not the canonical map from $\mathbb{R}^2$ to $B_2(r)$; two sigmoid functions could also have been used. We chose this particular function because it avoids the finite domain problem, is easily differentiable by the tools we use, and has a loose geometric interpretation described in Bonnet & Mellier (1995).

Note that this inverse only specifies what the new norm should be; the components still need to be adjusted accordingly. We reparameterize[20] as follows:

$$s \equiv \sigma^2, \tag{14}$$

$$g_i \equiv e_i \frac{||g||}{||e||}. \tag{15}$$

[19] We could look at this as a map from $B_n \to \mathbb{R}^n$ but in this case, we are only concerned about vectors $g = [g_1, g_2]$ and $e = [e_1, e_2]$.

[20] Technically, for s to be strictly positive we would set $s := \sigma^2 + \varepsilon$, where ε is some small positive perturbation. Numerically, it makes little difference: stars with very small s are removed during preprocessing.

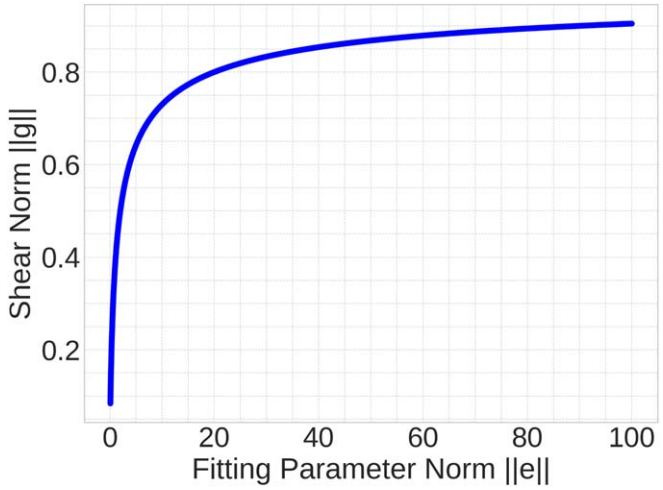


Figure 2. Reparameterization function. For any ellipticity vector $[e_1, e_2]$ the associated shear vector $[g_1, g_2]$ has a norm in [0, 1).

The values of σ, e_1, e_2 are obtained from each update step and $\|e\|$ is then determined from the usual L_2 norm. The parameter s is obtained from Equation (14) and $\|g\|$ from Equation (13). Finally, $\|g\|$ is used with $\|e\|$ to calculate g_1 and g_2.

We have that r is a function of $[u, v, s, g_1, g_2]$ and f is a function of r such that

$$f(r) \equiv I_p(u, v, s, g_1, g_2). \tag{16}$$

Even though we solve for parameters $[\sigma, e_1, e_2]$, the loss is still dictated by $[s, g_1, g_2]$

$$\text{cost}(s_t, g_{1_t}, g_{2_t}) = \frac{1}{2}\sum_{u_{\text{pix}}}\sum_{v_{\text{pix}}}(I_p(u, v)^* - I_p(u, v, s_t, g_{1_t}, g_{2_t}))^2. \tag{17}$$

Here $*$ denotes the ground truth (i.e., star) in the star vignette and the subscript t denotes iteration t in an LBFGS run. Parameters are found using Julia's `Optim.jl` library (Mogensen & Riseth 2018) and the gradient is computed using Julia's `ForwardDiff.jl` library (Revels et al. 2016).

3.2. Pixel-grid Fits

`PCA` mode, `autoencoder` mode, and `smoothing` mode all provide "pixel-grid" mechanisms for reconstructing the PSF on the image pixel basis. We outline each of these modes below.

3.2.1. PCA Mode

`PCA` mode approximates a star image with a rank-n approximation of the original image using PCA, where n is supplied by the user. Modeling the vignette in this way gives the user a method of determining how much they want their pixel-grid model to represent the original star vignette. Choosing a lower rank approximation will yield less detail on the shape of the PSF, but will do so quickly and without much noise contamination. On the other hand, a higher rank approximation will capture more of the fine details, but at the risk of capturing some of the noise. It should be noted that the choice of n has little overhead cost on the `ShOpt` computation time.

To minimize aliasing effects that might appear in low-rank approximations, the output image is convolved with a smoothing Lanczos kernel $\mathcal{L}$:

$$\mathcal{L}(x, a) = \begin{cases} 0 & \text{if } |x| \geqslant a \\ 1 & \text{if } x = 0 \\ a\text{sinc}(\pi x)\text{sinc}\left(\frac{\pi x}{a}\right) & \text{otherwise,} \end{cases} \tag{18}$$

where a is a tunable parameter for the size of the kernel and x is the distance from the center of the kernel.

3.2.2. Autoencoder Mode

`Autoencoder` mode uses deep learning to reconstruct PSF vignettes pixel by pixel. We adopt an autoencoder architecture because the projection into a latent space forces the machine learning algorithm to learn the key features of the image regardless of the noise present. The operations are carried out by Julia's `Flux.jl` machine learning library (Innes 2018).

In our specific architecture, the input star vignette is first flattened and then passed through the network, which has an encoder with one layer containing 128 nodes and a second layer containing 64 nodes. The encoder feeds into a latent space with 32 nodes, which is then decoded back into a layer of 64 nodes, 128 nodes, and finally back into an image (vignette) of the same dimension as the input vignette. We enforce both the input and output vignettes to sum to unity so that the relative distributions of intensities can be compared. The loss function is a mean squared error:

$$\text{cost}(x) = \frac{1}{N_{\text{pix}}}\sum_{N_{\text{pix}}}(x_p - \hat{x}_p)^2, \tag{19}$$

where $\hat{x}$ denotes the image after it has been put through the autoencoder.

The network consists of two activation functions. A `leakyrelu` (Maas 2013) function is used for all layers except for the last one. This choice reflects that most of the vignette pixel values are positive, and the ones that are not are usually close to zero or eliminated in data preprocessing. The final layer uses a `tanh` activation function to ensure that output values stay bounded between $(-1, 1)$. The network trains until it either hits a specified number of epochs or until it hits a stopping gradient. We encourage exploration of the number of epochs and the minimum stopping gradient to find an appropriate middle ground between accuracy and time of completion. Stopping gradients between 10^{-5} and 10^{-6} are usually sufficient to get a good approximation in a reasonable time and in 100 epochs or less. This was revealed through the use of diagnostic plots that are introduced in Section 8. The activation functions and number of nodes are not tunable by default; changing those is not recommended.

3.2.3. Smoothing Mode

In the case of well-sampled images like the NIRCam data considered in this analysis, we find that basic smoothing is sufficient before interpolation. This is implemented in ShOpt as the `smoothing` mode, which uses only the Lanczos kernel introduced in Equation (18) before producing the pixel-grid fit. While `PCA` and `Autoencoder` can yield denoised PSF models from low-dimensional reconstructions, the `smoothing` mode is a simpler technique that avoids some of their limitations, albeit at the expense of noisier models.

4. Interpolation Across the Full Field of View

To fit for the spatial variation of the size and shape of the PSF, `ShOpt` first produces a rough estimate by interpolating the analytic fit parameters $[s, g_1, g_2]$ across the field of view using a polynomial of order n, where n is supplied by the user. For an order-n polynomial $p(u, v)$), the cost function takes the form

$$\begin{aligned}&\text{cost}_p([a_0,\ldots,a_{(n+1)(n+2)\,//\,2}]) \\ &\quad=\left(\left[\sum_{(i,j),i+j\leqslant n} a_{ij}u^i v^j\right]-p(u,v)^*\right)^2,\end{aligned} \tag{20}$$

where $*$ denotes the ground truth obtained by pixel-basis fits. This gives us different polynomials in u and v for $[s, g_1, g_2]$. There is a tunable parameter for the stopping gradient that leaves the tradeoff between speed and accuracy of the polynomial interpolation to the end user. As elsewhere, the minimum of the cost function is found with LBFGS and the `Optim.jl` library.

Subsequently, the pixel-grid model is interpolated across the field of view. By definition, each pixel in pixel-grid models is a basis element, so the natural thing to do is to give each pixel in an $n \times n$ vignette its own polynomial. However, we found that this approach for solving for the polynomial coefficients produced systematically biased models, possibly due to the conditioning of the pixel intensity values. Thus, we opt for an alternative approach. To solve for the coefficients of the polynomial in the pixel-basis fit, we may specify a matrix $m \times n$ matrix, where m is the number of stars we are interpolating over and n is still the order of the polynomial. We denote this design matrix by A. If vector x represents the coefficients of the polynomials and vector b represents the intensity values, then we may use the known least squares solution to the matrix equation

$$Ax = b. \tag{21}$$

Not all stars are good exemplars of the PSF due to things like saturation, color effects, and noise. To combat this, `ShOpt` does its polynomial interpolation over several iterations according to Algorithm 2.

Algorithm 2. Iterative Polynomial Interpolation and Star Filtering Process

```
NumIterations ← [define number of iterations]
for i ← 1 to NumIterations do
    PSFModel ← PolynomialInterpolate(TrainingStars)
    RenderedStars ← RenderPSF(PSFModel)
    MSE ← ComputeMSE(TrainingStars, RenderedStars) ▷ MSE: Mean Squared Error
    WorstStars ← GetWorstPerformingStars(MSE, 10% or N sigma)
    TrainingStars ← TrainingStars—WorstStars
end for
    FinalPSFModel ← PolynomialInterpolate(TrainingStars)
return TrainingStars, FinalPSFModel
```

The number of iterations to refine the polynomial interpolation is specified by the user. After each iteration, the predicted PSF is rendered at each star location and the mean squared error (Equation (22)) is computed using the training stars. The worst 10% of training stars are filtered out

$$\text{Mean Squared Error} = \frac{1}{N_{\text{pix}}}\sum_{(i,j)}^{N_{\text{pix}}}(v_{i,j}-p_{i,j})^2. \tag{22}$$

Alternatively, we provide a mode that removes $N \times \sigma$ outliers from the interpolation. Polynomial interpolation for high-degree polynomials can be the most expensive part of the whole PSF fitting procedure. For this reason, `ShOpt` is strict about filtering outliers in polynomial interpretation. Training stars are excluded based on the value of s that was found during the analytic profile interpolation step, which eliminates the need for additional iterations to clean the training data. Additionally, we make the conscious decision not to continue filtering our training stars until we reach a given $n\sigma$ confidence. The JWST PSFs contain lots of background noise between the wings of the stars we are trying to model. Consequently, an excessive number of iterations may be required to reach the desired level of confidence.

5. Data Preprocessing and Outlier Rejection

In `ShOpt`'s outlier rejection process, three distinct phases are employed: (1) SNR-based filtering on input stars; (2) a Gaussian fit size filter on the input stars; and (3) a separate Gaussian fit size filter on pixel-grid models. Following these phases, the process advances to the iterative refinement for polynomial fitting, as detailed in Section 4.

Before doing anything else, `ShOpt` filters out bad stars on the basis of SNR; the default SNR metric is the SExtractor `SNR_WIN` parameter. By default, `ShOpt` filters out the noisiest 33% of entries, a percentage determined through experimentation. Users can adjust this threshold to fit their specific data sets.

The remaining vignettes are then fit with a multivariate Gaussian, as described in Section 3. We filter again based on the object size s. A very small or very large s probably corresponds to objects that are not point sources. By default, vignettes with $s < 0.075$ or $s > 1$ are filtered out. This ensures further computation time is not wasted on bad data. We apply the same size filtering after fitting Gaussians to the pixel-grid models. This prevents poor pixel-grid models from being used in our polynomial interpolation, as mentioned in Section 4.

`ShOpt` also offers several methods for efficiently cleaning useful data without discarding them. Given that SExtractor sets the flux of interloping sources to a sentinel value of -10^{32}, `ShOpt` sets pixels less than -1000 to `NaN`. Before doing any analytic or pixel-grid fits, we also smooth the image according to the kernel introduced in Equation (18) to avoid any hot pixels in the vignettes. Finally, we recommend that users select `true` in the YAML configuration for the setting `sum_pixel_grid_and_inputs_to_unity`. This enforces intensity to sum to unity in each of our model vignettes and their pixel-grid fits.

6. Runtime Analysis

In this section, we compare the algorithmic complexity and convergence properties of `ShOpt`, PIFF, and PSFEx, examining the analytic profile fits, pixel-grid fits, and polynomial interpolation steps separately. Runtime analysis of the three PSF fitters allows us to argue for the algorithmic choices implemented in `ShOpt` without needing to factor in a programming language or computing power. The results will serve as predictions for the speed tests of Section 8.3.

6.1. Analytic Profile Fit Runtime

Let n denote the number of pixels on one side of a square vignette so that there are n^2 total pixels. In our optimization scheme, we compute the loss between every pixel in the vignette and the analytic profile prediction an average of I times. Our nominal runtime is thus $\mathcal{O}(n^2 \times I)$.

The PIFF configuration used in this work does not solve for $[s, g_1, g_2]$ using a nonlinear process. Instead, this configuration uses iterative linear least squares with slight updates to the centroid and total flux over a grid of pixels. However, there are other PIFF configurations that use nonlinear processes to find $[s, g_1, g_2]$ based on first-order approximation methods. On the other hand, `ShOpt` uses LBFGS, which uses superlinear approximations to calculate the direction of improvement to find $[s, g_1, g_2]$. The LBFGS algorithm and the reparameterization step introduced in Section 3.1 allow us to argue that $I_{\mathrm{ShOpt}} < I_{\mathrm{PIFF}}$. While we refrain from claiming that `ShOpt` will always converge to the correct $[s, g_1, g_2]$ faster than PIFF, we have designed `ShOpt` with the hopes that using a more memory-expensive technique to compute its update steps while keeping the solution within a constraint gives us better convergence.

6.2. Pixel-grid Fit Runtime

6.2.1. PCA Runtime

PCA relies on computing the singular value decomposition of the covariance matrix of a given data set. Singular value decomposition is $\mathcal{O}(n^3)$ for an $n \times n$ matrix, but the fit is typically much cheaper to compute for an approximation using the first k principal components. While this does not lower the big-O complexity to the χ^2 minimization pixel-grid fitting implemented in PIFF and PSFEx (Bertin 2011; Jarvis et al. 2020), we do not encounter any noticeable speed bottlenecks at this step.[21] We will explore this more in Section 8.3.

6.2.2. Autoencoder Runtime

As in Section 6.1, we may observe that the complexity is $\mathcal{O}(n^2 \times I)$ because the loss is computed an average of I times over n^2 pixels. The number of parameters between layers as a function of neurons m in layer i in our network is given by

$$N_{\mathrm{params}} = (m_{i-1} \times m_i) + m_i, \tag{23}$$

where the first term corresponds to the number of weights and the second term corresponds to the biases. In our network, the number of nodes for the input and output is set by a flattened version of the input image. Therefore, the number of parameters to learn grows as $[(n^2 \times 128) + 128] + [(128 \times n^2) + n^2]$. On the other hand, PIFF and PSFEx employ a form of χ^2 minimization, wherein each pixel in the image is a learnable parameter; for an (n, n) image there are only n^2 learnable parameters and n^2 pixels computed in the loss function. Even though the loss function in `ShOpt` is also computed over n^2 pixels, there are many more than n^2 learnable parameters, so we expect the autoencoder to take more iterations to converge on average than PIFF and PSFEx. For this reason, `autoencoder` is not the default mode for pixel-grid fits in `ShOpt`. It should be reserved for cases that demand precision and where the images are small enough to be learned efficiently so the transfer learning effect is more pronounced.

[21] A χ^2 minimization pixel-grid fit will be incorporated in a future `ShOpt` release.

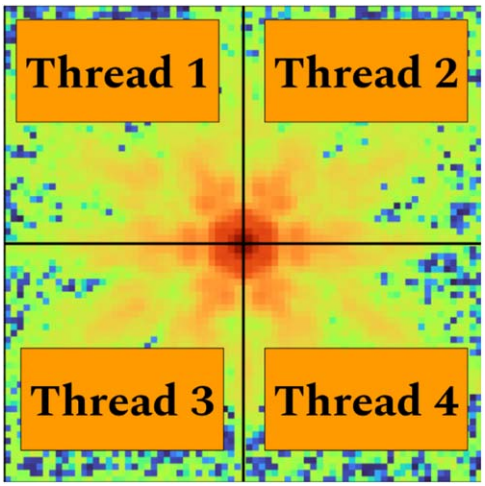


Figure 3. Pictoral representation of multithreading in `ShOpt`. In this example, all of the pixels in the upper left of the pixel-grid renderings are interpolated across the field of view in thread 1, and all of the pixels in the upper right of the pixel-grid renderings are interpolated across the field of view in thread 2, and so on.

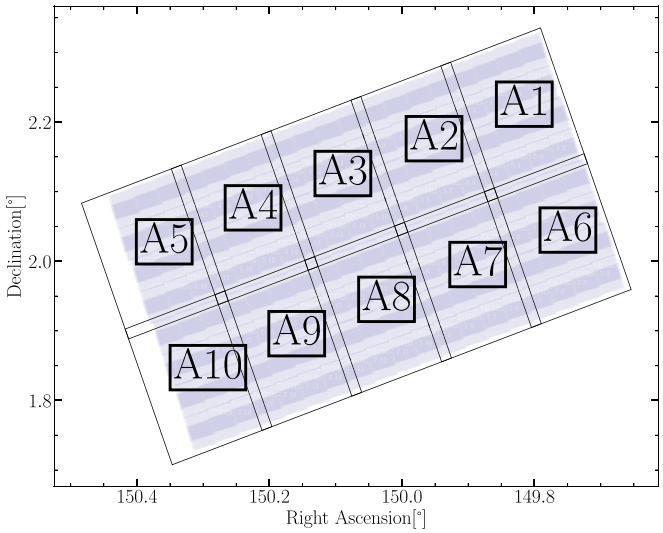


Figure 4. April data tiling scheme.

6.3. Polynomial Interpolation Runtime

Almost all PSF fitting software makes use of polynomial interpolation to model the variation of the PSF across the camera's field of view. Any performance gains in this area are primarily derived from the efficiency of the polynomial fitting process to the data. In PSFEx, these fits are implemented using PCA, which Bertin (2011) argues requires the lowest number of basis vectors to approximate an image if the data are sufficiently well-behaved.

In `ShOpt`, we use the design matrix and the known least squares solution to the matrix equation $Ax = b$ to fit each pixel with an independent polynomial. This process is inherently parallelizable because the polynomial found for one pixel is independent of the polynomial found for any other pixel (see

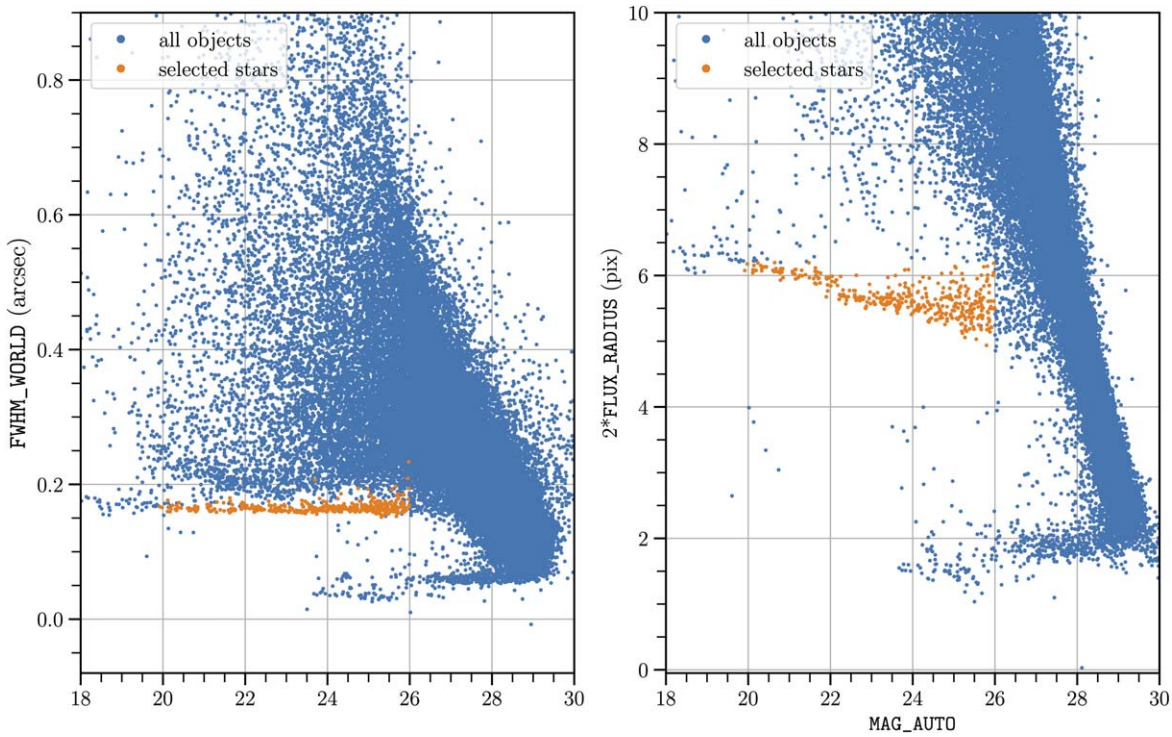


Figure 5. Size–magnitude diagram for objects detected in the A6 mosaic in F444W A6. Blue points represent all sources; orange points show stars selected by stellar locus parameter cuts.

Figure 3). `ShOpt` will automatically use as many threads are made available to it. The idea to break these operations into computation blocks is inspired by Tiny Machine Learning problems described in Sabot et al. (2023) and the CAKE algorithm outlined in Kung et al. (2021). Currently, the number of threads is not configurable because it has to be specified before the program is run. On UNIX, we ran `export JULIA_NUM_THREADS = auto` before running the program, on Windows you can similarly run `set JULIA_NUM_THREADS = auto`.

6.4. Order of Operations

We are also deliberate about the order of operations in `ShOpt`. The initial analytic profile fitting serves as a data cleaning method, which allows us to not waste compute time fitting pixel-grid models to outlier stars. A second round of analytic profile fits to the resulting PSF models further refines the data set, ensuring that the (computationally expensive) polynomial interpolation is applied only to the highest quality PSF models. By contrast, PIFF adopts an integrated approach where analytic and pixel-grid fits are interwoven with iterations of polynomial interpolation. While this method is thorough, our approach is designed to prioritize computational expediency.

7. Benchmarking, Data, and Analysis

Our PSF analysis utilizes NIRCam imaging from the COSMOS-Web survey (Casey et al. 2023), which we have chosen for its expansive coverage (more than three times the area of all other JWST surveys combined) in four bandpasses as well as the fact that many of its science cases require careful PSF characterization. This section presents an overview of the three COSMOS-Web data sets employed in our PSF benchmarking analysis—simulated single exposures, simulated observation mosaics, and real observation mosaics—followed by a description of the benchmarking methods we apply to each.

We employ simulated observations for PSF code unit testing because they provide a controlled environment with fully known input parameters. Unlike real observations, simulations allow us to control for observational imperfections such as saturation, noise, or defective pixels; this allows for the establishment of a ground truth against which the accuracy of PSF model fits can be measured directly. Although real observations feature the true PSF and are integral to our study, their inherent uncertainties and the absence of a definitive PSF ground truth make them less suitable for initial code validation. Instead, the real data serve as a stress test for the PSF fitting codes, ensuring they remain effective when confronted with the vagaries of real observations.

1. *Simulated Data:* These simulations are based on the COSMOS 2020 data (Weaver et al. 2022), and contain all the known galaxies and stars in this field. Galaxy fluxes in JWST filters were calculated from `Stardust`SEDs if they existed (Kokorev et al. 2021); otherwise, fluxes were found by interpolating across existing photometry in other filters. Star fluxes were similarly modeled by interpolating available photometry. Galaxies with counterparts in the Zurich Structure & Morphology catalog (Sargent et al. 2007; Scarlata et al. 2007) were assigned the according morphological parameters (Sérsic indices, ellipticity, position angle, size), or from observed distributions if they did not.

   COSMOS-Web-like NIRCam images were generated using the Multi-Instrument Ramp Generator (MIRAGE; Hilbert et al. 2019), which includes PSF modeling with WebbPSF (Perrin et al. 2014), sky background, detector noise, dark current, and Poisson

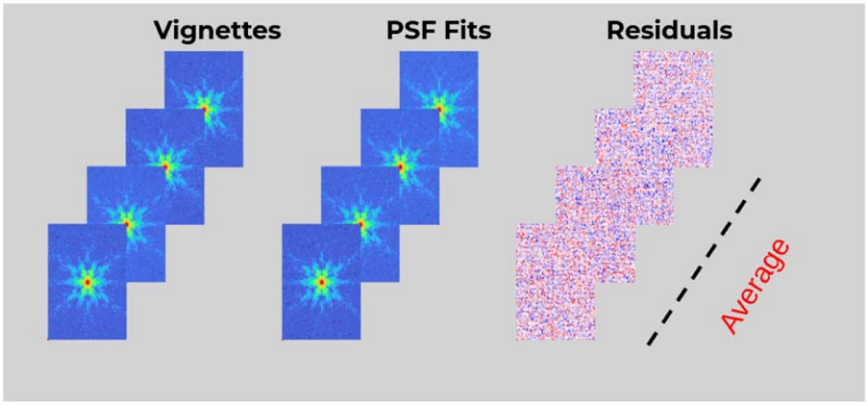


Figure 6. Schema illustrating the computation of summary statistics for PSF model assessment. Individual star-PSF model pairs are compared to produce residual images, which are then averaged into a mean residual image for the ensemble.

noise. After the images were generated, the JWST calibration pipeline (Bushouse et al. 2023) was used to reduce the raw NIRCam data and create mosaics, with some modifications, like $1/f$ noise and subtraction of low-level background (e.g., Bagley et al. 2022; Finkelstein et al. 2022). We employ two sets of simulated images:

(a) *Simulated Single Exposures:* Stars are generated by convolving a MIRAGE WebbPSF model with idealized point sources, so these images allow us to measure success directly by comparing the learned PSF to the input PSF. We employ images from stage 2 of the JWST calibration pipeline, so there are no effects from dithering or distortion (i.e., `tweakreg`). The main challenge in using these data is the low star density, particularly for the short-wavelength channel. With a training/validation split of 90%, this led to an average of 1 validation star for each subarray in the F115W filter, 1 validation star for the F150W filter, 2.2 stars for the F277W filter, and 2.3 stars for the F444W filter. To gather enough data for meaningful summary statistics, we combine exposures from 156 different visits.

(b) *Simulated Mosaics:* Image mosaics are `i2d`-format data cubes built from single exposures that have passed through stage 3 of the JWST science calibration pipeline.

As such, the mosaics do reflect normal dithering and distortion effects. The simulated mosaics used in our study cover a contiguous area of 76 arcmin^2 at the full COSMOS-Web depth (∼27th magnitude) and provide a reasonable 0.5 star per square arcminute.

For an in-depth discussion of the simulation process, refer to N. Drakos (2024, in preparation).

2. *Real Mosaics:* The real data used in our analysis were taken in 2023 April and include visits 77–152, covering 0.28 deg^2 in the bottom right of our allocated area of the COSMOS field (Casey et al. 2023), JWST program ID 1727.[22] Three of the planned visits were skipped, so the data include a total of 72 visits. As with the simulated mosaics, we use the `i2d`-format data cubes produced in stage 3 of the JWST science calibration pipeline. We restrict ourselves to an approximately 0.11 deg^2 area of the full field of view, corresponding to tiles {*A*1, *A*5, *A*6, *A*10} in Figure 4; we chose these particular tiles to test the relative performance of the PSF fitters where we expect astrometric distortions to be the most severe.

[22] Available from MAST at STScI, http://mast.stsci.edu.

Table 1
Number of Reserved Validation Stars for the Three Imaging Data Sets Under Consideration in Each of the Four COSMOS-Web NIRCam Bandpasses

Data Type	Filter	Validation Stars
Simulated single exposures	F115W	3373
	F150W	7186
	F277W	2573
	F444W	740
Simulated mosaics	F115W	37
	F150W	79
	F277W	35
	F444W	35
Real mosaics	F115W	155
	F150W	156
	F277W	148
	F444W	136

Our benchmarking procedure has four steps:

1. Run Source Extractor (SExtractor) on our images to generate star catalogs (Bertin & Arnouts 1996). For the simulated data, stars are identified by matching the SExtractor catalogs with the input point source catalogs. For real data, star catalogs are created using cuts on the stellar locus, an example of which is shown in Figure 5. The config file for SExtractor is given in the Appendix. Note that we run the source extractor on the individual tiles and aggregate our results over all of them.
2. The resulting star catalogs are segregated into training (90%) and validation (10%) catalogs. We are careful to filter out saturated stars in our catalogs by searching for sentinel values of 0 in the `ERR` extension (for i2d-format mosaics) or 1, 2 in the `DQ` extension (for single exposures). Vignettes that contain any pixels set to a sentinel value are removed from the catalogs.
3. We use the training catalogs to get empirical PSF models for each fitter. We run `ShOpt` (Berman & McCleary 2024), PIFF (Jarvis et al. 2020), and PSFEx (Bertin 2011).

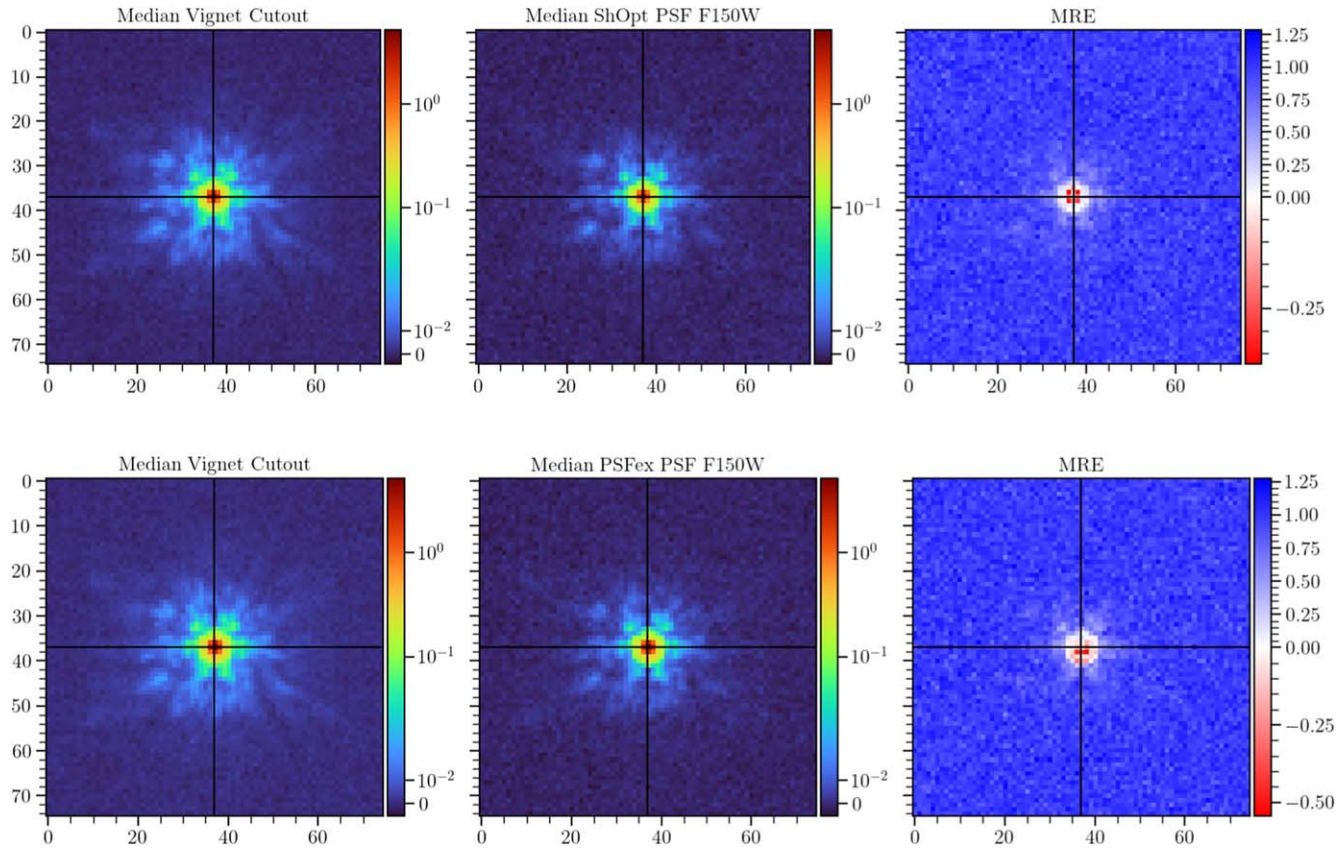


Figure 7. Mean relative error between MIRAGE input point source images and PSF models for the F277W filter. The left panels show the median MIRAGE cutout; the center panels show the median PSF cutout. The right panels show the average relative error between the MIRAGE and PSF cutouts. The top two rows are `ShOpt`, and the bottom two rows are PSFEx. Color bars for left and center panels show pixel intensity values in units of MJy sr^{-1}. The color bars in the right panels show the (dimensionless) relative error. The MREs defined in Equation (7) are displayed in the titles of the mean residual images.

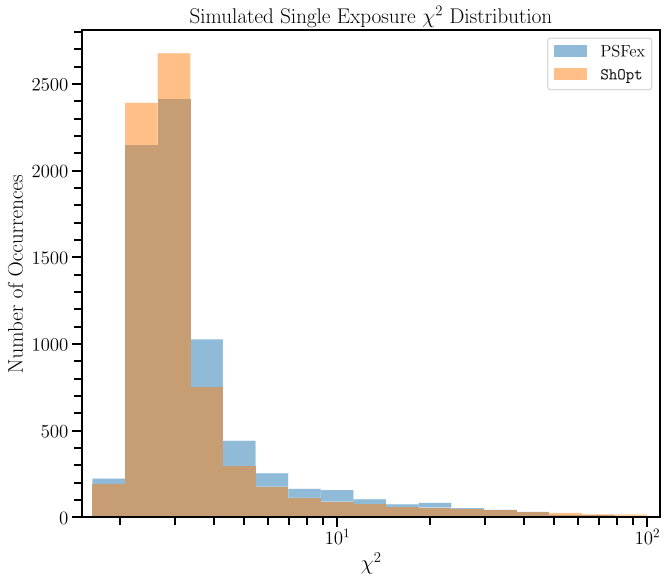


Figure 8. Distribution of χ^2_ν for simulated single exposure images. Not shown are 42 PSFEx χ^2_ν values greater than 100 and 100 `ShOpt` χ^2_ν values greater than 100.

4. We then use the reserved validation star catalogs to calculate the summary statistics of Equations (5)–(8). Figure 6 illustrates this process: we calculate the residuals between the star vignettes (or MIRAGE cutouts for simulations) and renderings of the PSF models, then take the mean. To obtain single residual scores, we compute the mean and standard error of all pixels in residual images.

The code for creating star catalogs, calculating statistics, and creating the associated PSF diagnostic figures can be found on GitHub.[23]

8. Results

In this section, we report the outcomes of the PSF benchmarking analysis detailed in Section 7. The comprehensive PSF model fidelity analysis is presented in Section 8.1, and the assessment of the computational efficiency of the different PSF fitters is presented in Section 8.3.

We note that `ShOpt` was run in `smoothing` mode throughout. We found that `PCA` mode and `Autoencoder` mode added additional complexity to the PSF fitting process that resulted in poorer fits. Appendix C contains representative configuration files supplied to each PSF fitter.

8.1. Nonparametric Model Fidelity

As described above, we evaluate the relative performance of PSF models produced by PIFF, PSFEx, and `ShOpt` using mean and median reduced χ^2 residuals, the MRE, and the MAE (see Equations (5)–(8)). These statistics are

[23] https://github.com/mcclearyj/cweb_psf

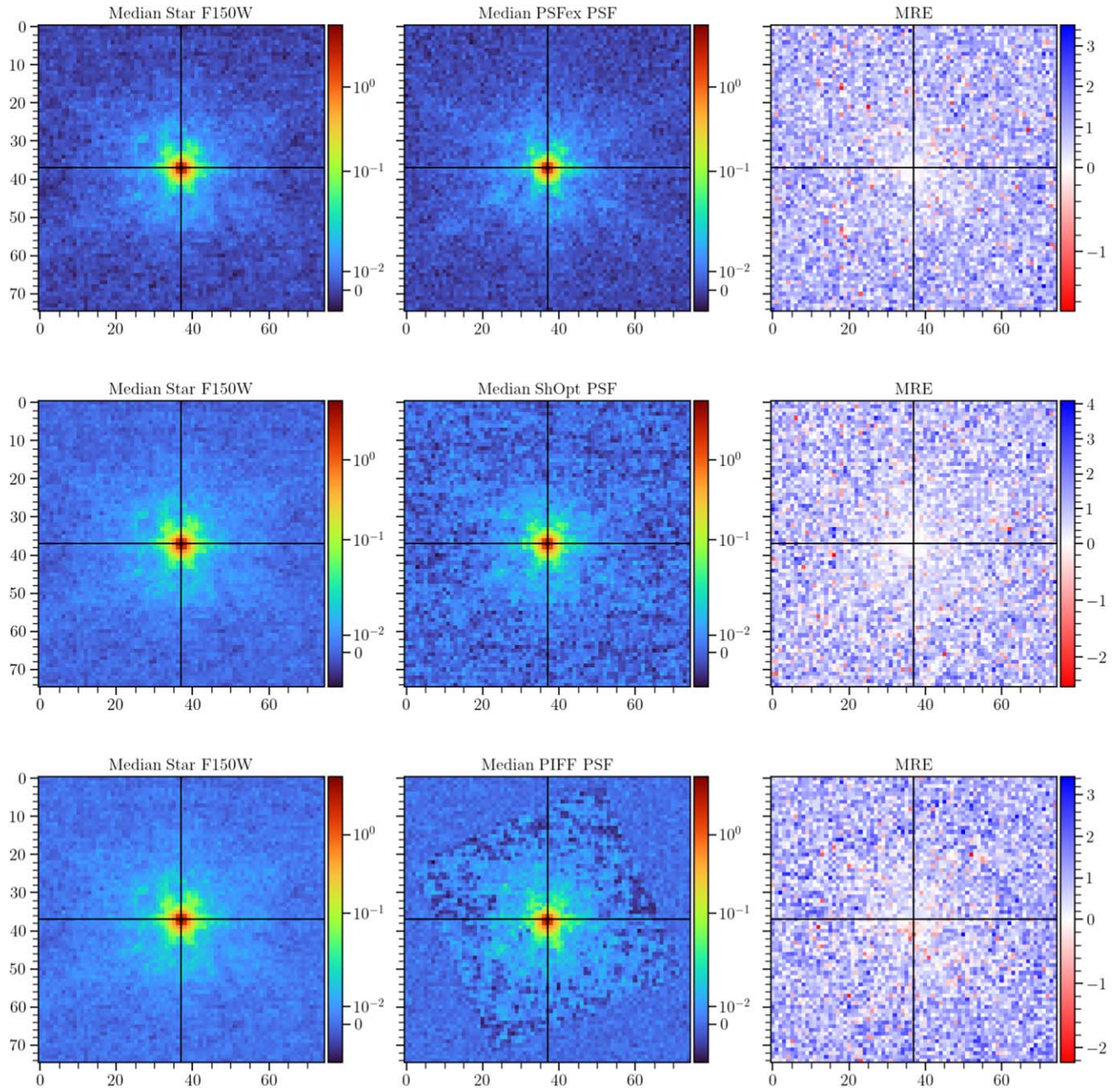


Figure 9. Evaluation of mean relative error between stars and PSF models for simulated mosaics in the F150W bandpass. The left panels show the median of the vignettes. Center panels show the median PSF cutout. The right panels show the average relative error between the vignette cutouts and the PSF cutouts. The top is PSFEx, the middle is `ShOpt`, and the bottom is PIFF.

computed using the reserved validation stars, which number from 35 for the simulated F277W and mosaics to 156 for the F150W real mosaics (Table 1). Statistics and diagnostic plots are based on PSF vignette sizes of (75, 75) pixels, which encloses the majority of the relevant star and PSF light profiles without including excessive sky background or a large number of interloping objects.

8.1.1. Simulated Single Exposures

While the simulated single exposures yield ample training data in aggregate, training data on individual detectors from individual visits are sparse. Despite this sparsity of training data, Figures 7, 16, and 17 show that both PSFEx and `ShOpt` produce reasonable models of the PSF. While both PSF fitters seem to slightly underfit the center of the PSF, they are otherwise able to model the finer details of the PSF.

The distribution of reduced χ^2 shown in Figure 8 illustrates that `ShOpt` produces a fit that is just as strong, if not stronger than PSFex.

8.1.2. Simulated Mosaics

The simulated data mosaics have a higher density of training stars, as evinced by significantly better fits for PSFEx and

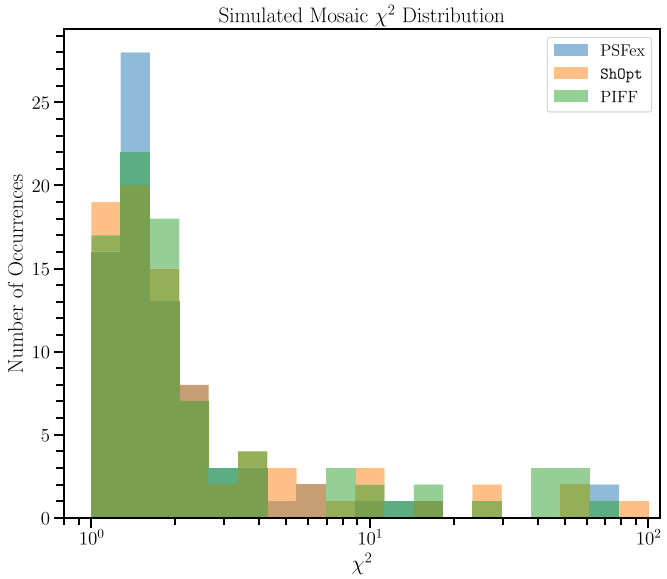


Figure 10. Distribution of χ^2_ν for simulated data mosaics. PSFEx values are shown in blue, `ShOpt` values are shown in orange, and PIFF values are shown in green. For clarity, the plot excludes two PSFEx χ^2_ν values greater than 100, one `ShOpt` χ^2_ν value greater than 100, and two PIFF χ^2 values greater than 100.

`ShOpt` than the simulated single exposures. We also supply PIFF fits for these images. Figures 9 and 18–20 do not show the same overconcentration visible in Figures 7, 16, and 17.

Figure 10 suggests minimal statistical difference in performance among the PSF fitters, with the same heavy-tailed distribution of χ^2_ν for each.

Table 2 shows MRE and MAE values are consistent with zero for all `ShOpt` and PSFEx models, suggesting minimal bias. While the PIFF model residuals are also consistent with zero, the 90% bounds tend to be larger than `ShOpt` and PSFEx, indicating an increased incidence of catastrophic fits. We also point out that the 10% errors tend to be close to the median error among each of the PSF fitters.

In general, the presence of outlier fits obscures a clear ranking based on MAE, MRE, and $\overline{\chi^2}$ alone. The median χ^2 and its distribution, as shown in Figure 10 and Table 2, seem to be more indicative of performance, demonstrating no significant difference in the reliability of the PSF fitters for the simulated mosaics.

8.1.3. Real Mosaics

We find similar results for the real data mosaics as for the simulated mosaics exposures, namely that each PSF fitter produces similarly high-quality models. Figures 11 and 21–22 suggest that in the main, both `ShOpt` and PSFEx are able to model the finer details of the PSF at all bandpasses analyzed.

The high model fidelity for each fitter is further supported by the values of median and mean reduced χ^2 in Table 3, as well as the distributions of reduced χ^2 in Figure 12. We do not run PIFF on the real mosaics due to timing costs; the real data mosaics cover tens of thousands of pixels, and PIFF fits did not reliably converge.

8.2. Size and Shape Analysis

Although the NIRCam PSFs are obviously not elliptical Gaussians, adaptive second moments are a common way of evaluating the quality of PSF fits (Hirata & Seljak 2003; Mandelbaum et al. 2005), and so in this section, we explore residuals in the second-moment fits of stars and PSF models. Specifically, we use the the GalSim software's (Rowe et al. 2015) `FindAdaptiveMom` function to measure the size ($\sigma_{\rm HSM}$) and shape ($(g1, g2)$) of the best-fit elliptical Gaussians of all validation stars and PSF models, and then compute the average of the differences between the two. As demonstrated by Table 4, `ShOpt` is able to produce fits that are just as good if not better than PSFEx and PIFF across the different data sets and wavelengths. The only area where the `ShOpt` models struggled was with the shape adaptive moments on some of the simulated single exposures. Otherwise, `ShOpt` produced just as good if not better size and shape statistics compared to the other PSF fitters.

8.3. Program Speed and Scalability

We investigate how the different PSF fitters handle the simulated mosaic data as the number of pixels in the star vignettes and the degree of the polynomial used to model spatial variations increase. Each PSF fitter is run multiple times to get an average time in seconds for pixel-based fits. For `ShOpt`, we set the size of the side length n of the analytic fit stamp to be 30. That is, we use 30×30 pixels to fit the multivariate Gaussian. This is separate from the number of pixels we use for the pixel basis. We do not use any GPU compute power to accelerate the PSF fitters even though the `ShOpt autoencoder` mode can be GPU-accelerated. All of these tests are run on the same hardware using Northeastern's Discovery Cluster on the zen2 CPUs (Northeastern University Research Computing 2024). These tests aim to test our calculations in Section 6.

8.3.1. Variation with the Vignette Size

While the runtime for both `ShOpt` and PSFEx scales modestly as the output PSF size increases, the PIFF runtime does not. For larger PSF sizes, PIFF's average `BasisInterp` execution time is an order of magnitude longer than `ShOpt` and PSFEx. While `ShOpt` and PSFEx required approximately 5–10 minutes to process PSF sizes of (137, 137) pixels for each of the four wavelengths, PIFF consistently took between 1 and 3 hr for PSF sizes of (64, 64) (Figure 13). In the worst case, PIFF can take as much as an entire day to finish. While most PSF fitters had consistent completion times, PIFF exhibits greater variability for (64, 64)-pixel PSF output sizes, particularly with shorter wavelengths. It is worth noting that PIFF was developed using DECam imaging, for which PSF sizes of (17, 17) are sufficient. For those vignette sizes, PIFF is actually faster than the other PSF fitters, see Figure 13. It is also worth noting that PIFF contains alternatives to the `BasisInterp` algorithm benchmarked here and that these algorithms may be faster.

8.3.2. Polynomial Degree

As in the case of vignette size, while both `ShOpt` and PSFEx scale as the degree of the spatial interpolation polynomial increases, PIFF does not. For a degree-one interpolation, all three PSF fitters have similar performance

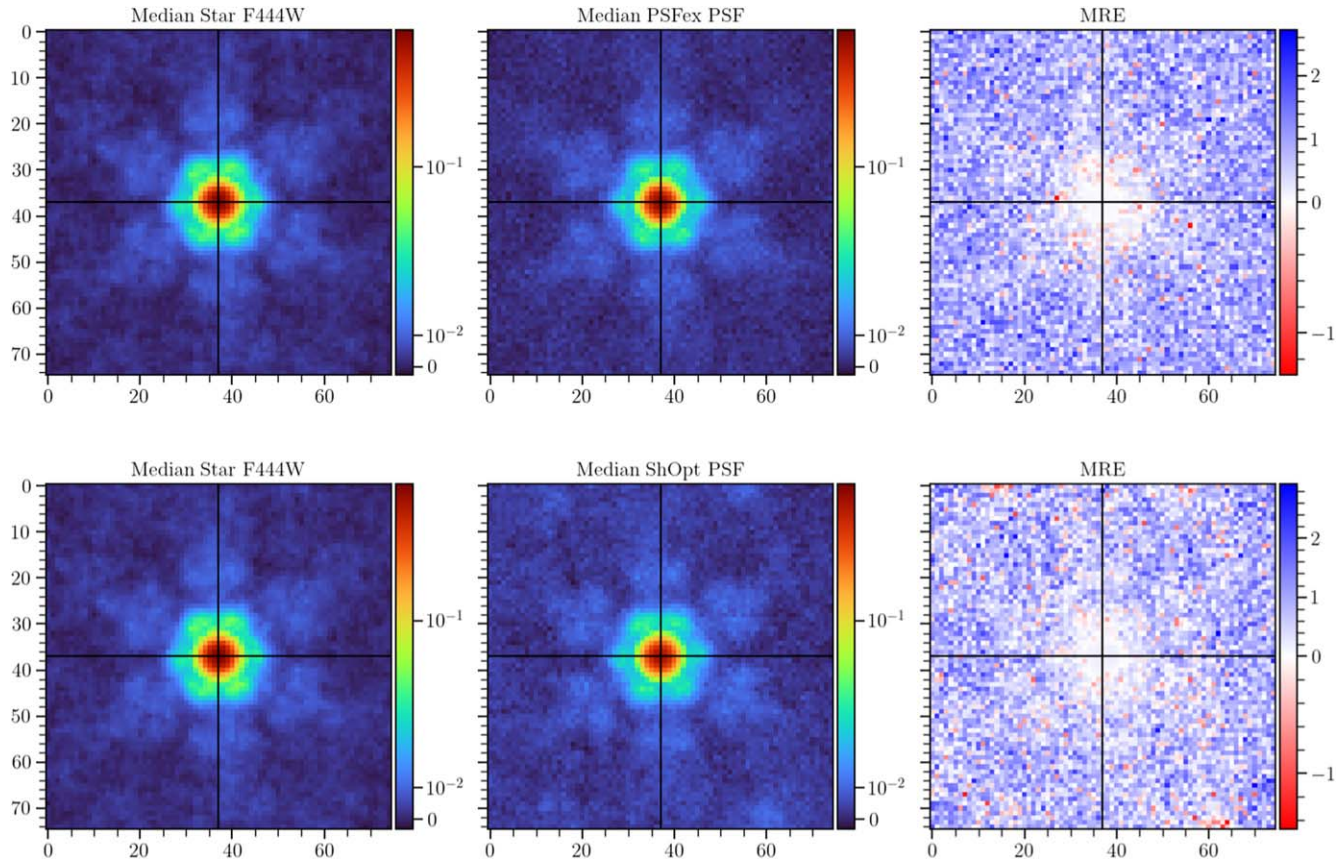


Figure 11. Evaluation of mean relative error between stars and PSF models for real data mosaics in the F444W bandpass. The left panels show the median star vignette; the center panels show the median PSF model; and the right panels show the mean residuals of individual star-PSF vignettes. The top row is PSFEx, and the bottom row is `ShOpt`.

Table 2
Simulated Mosaic Summary Statistics

Filter	PSF Fitter	MAE	MRE	$\overline{\chi^2_\nu}$	Median χ^2_ν
F115W	ShOpt	$3.41^{+5.17}_{-1.79}$	0.80 ± 1.18	$8.89^{+25.85}_{-1.16}$	1.78
	PIFF	$3.11^{+4.78}_{-1.72}$	0.87 ± 1.20	$12.76^{+49.40}_{-1.26}$	1.82
	PSFEx	$2.67^{+3.90}_{-1.54}$	0.81 ± 0.94	$3.81^{+10.90}_{-1.26}$	1.88
F150W	ShOpt	$3.22^{+4.39}_{-2.08}$	0.88 ± 0.79	$5.82^{+3.73}_{-0.93}$	1.42
	PIFF	$2.87^{+3.83}_{-2.01}$	0.88 ± 0.72	$5.93^{+4.90}_{-0.94}$	1.39
	PSFEx	$2.70^{+3.54}_{-1.90}$	0.87 ± 0.64	$5.40^{+2.07}_{-0.92}$	1.30
F277W	ShOpt	$3.60^{+5.49}_{-1.83}$	0.77 ± 1.26	$15.21^{+20.29}_{-1.25}$	1.98
	PIFF	$2.75^{+4.01}_{-1.68}$	0.85 ± 1.06	$36.97^{+48.55}_{-1.38}$	2.09
	PSFEx	$2.67^{+3.92}_{-1.58}$	0.83 ± 0.96	$16.99^{+37.48}_{-1.12}$	1.88
F444W	ShOpt	$3.08^{+4.43}_{-1.86}$	0.80 ± 1.07	$6.11^{+6.14}_{-1.03}$	1.70
	PIFF	$2.84^{+4.10}_{-1.76}$	0.89 ± 1.05	$13.49^{+15.76}_{-1.12}$	1.71
	PSFEx	$2.76^{+3.99}_{-1.70}$	0.86 ± 1.00	$2.01^{+3.31}_{-1.03}$	1.46

Note. The MAE and $\overline{\chi^2_\nu}$ statistics are reported with 10% and 90% errors.

for (33, 33) PSF sizes. However, as we increase the degree to two and three, PIFF takes an order of magnitude longer on average to execution, see Figure 14. For larger degree sizes, PIFF often did not converge to a PSF model with 4σ confidence after its maximum of 30 iterations. `ShOpt` and PSFEx maintain a consistent runtime of about 5 minutes, whereas PIFF experiences a dramatic increase in runtime, going up to hours for short wavelengths with higher degrees of the interpolating polynomial, and by around 30 minutes for longer wavelengths.

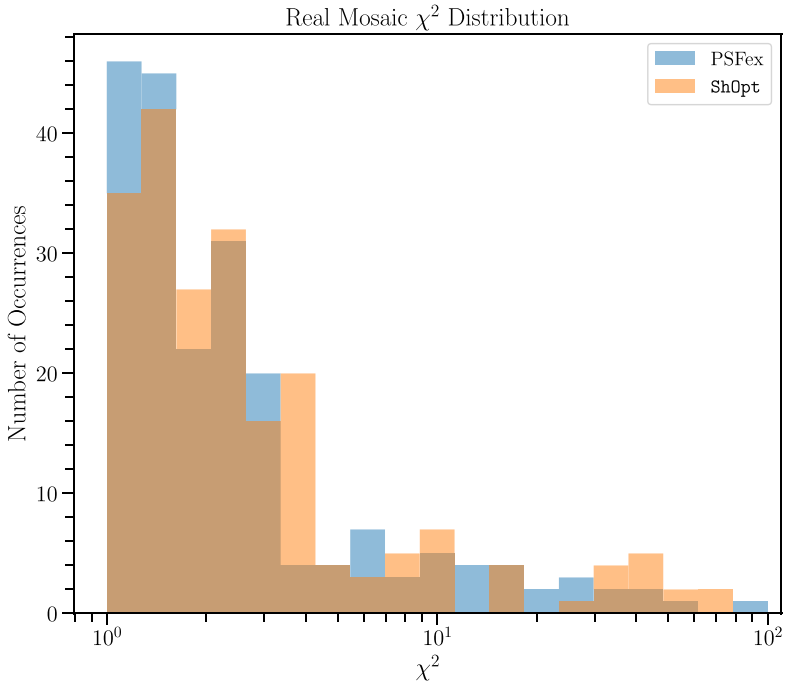


Figure 12. Distribution of χ^2_ν for real data mosaics. Not shown are two PSFEx χ^2_ν values greater than 100 and four ShOpt χ^2_ν values greater than 100.

Table 3
Real Mosaic Summary Statistics

Filter	PSF Fitter	MAE	MRE	$\overline{\chi^2}_\nu$	Median χ^2_ν
F115W	ShOpt	$2.96^{+3.65}_{-2.25}$	0.88 ± 0.50	$3.12^{+3.96}_{-0.99}$	1.49
	PSFEx	$2.76^{+3.38}_{-2.15}$	0.90 ± 0.49	$4.49^{+5.30}_{-0.96}$	1.45
F150W	ShOpt	$2.96^{+3.69}_{-2.24}$	0.86 ± 0.51	$2.05^{+3.00}_{-1.02}$	1.43
	PSFEx	$2.76^{+3.42}_{-2.15}$	0.87 ± 0.48	$1.65^{+2.09}_{-0.93}$	1.27
F277W	ShOpt	$2.96^{+3.92}_{-1.87}$	0.63 ± 0.55	$29.28^{+92.44}_{-2.59}$	4.15
	PSFEx	$2.56^{+3.27}_{-1.81}$	0.75 ± 0.50	$13.55^{+34.46}_{-2.46}$	3.60
F444W	ShOpt	$2.89^{+3.83}_{-1.91}$	0.70 ± 0.60	$10.28^{+32.84}_{-1.67}$	2.86
	PSFEx	$2.58^{+3.30}_{-1.84}$	0.78 ± 0.52	$7.34^{+15.22}_{-1.71}$	2.49

9. Discussion and Conclusions

We have presented `ShOpt`, a new empirical PSF characterization tool, and with it, a methodology for benchmarking the accuracy and computational efficiency of PSF fitting software.

In our assessment of PSF model quality, we produced a series of mean residual images, including normalized mean error, mean absolute error, and χ^2 error. We further condense this information into aggregate statistics that quantify the pixel-level discrepancies between the modeled and actual PSFs across the ensemble of stars being modeled. We supplement these statistics with second-moment HSM fit statistics because of their wide use in the literature.

Though convenient, single-number figures of merit can mask discrepancies between a PSF model and the true PSF. Accordingly, we recommend a holistic approach to evaluating the quality of a particular PSF model, using both aggregate statistics and mean residual images. Among the various single-number figures of merit we examined, the average and median χ^2 values were the most illustrative of mismatched PSF models. Additionally, the distributions of these statistics are usually indicative of the performance of the PSF modeling.

In Section 6, we predicted that `ShOpt`'s algorithmic design would allow it to scale as the PSF model size and the degree of the interpolating polynomial increased; this assertion was borne out in our speed testing. Our analysis does not distinguish between the contributions of architectural choices and the Julia language itself to `ShOpt`'s fast execution time. Regardless, we find that `ShOpt` delivers speed performance on par with PSFEx, with only marginal differences in processing times for PSF models (137, 137) pixels in size (large enough to enclose most of the NIRCam PSF). `ShOpt`'s competitive speed comes from a combination of multithreading, imposed geometric constraints, the implementation of the LBFGS algorithm, and a thorough data cleaning pipeline. Conversely, PIFF—optimized

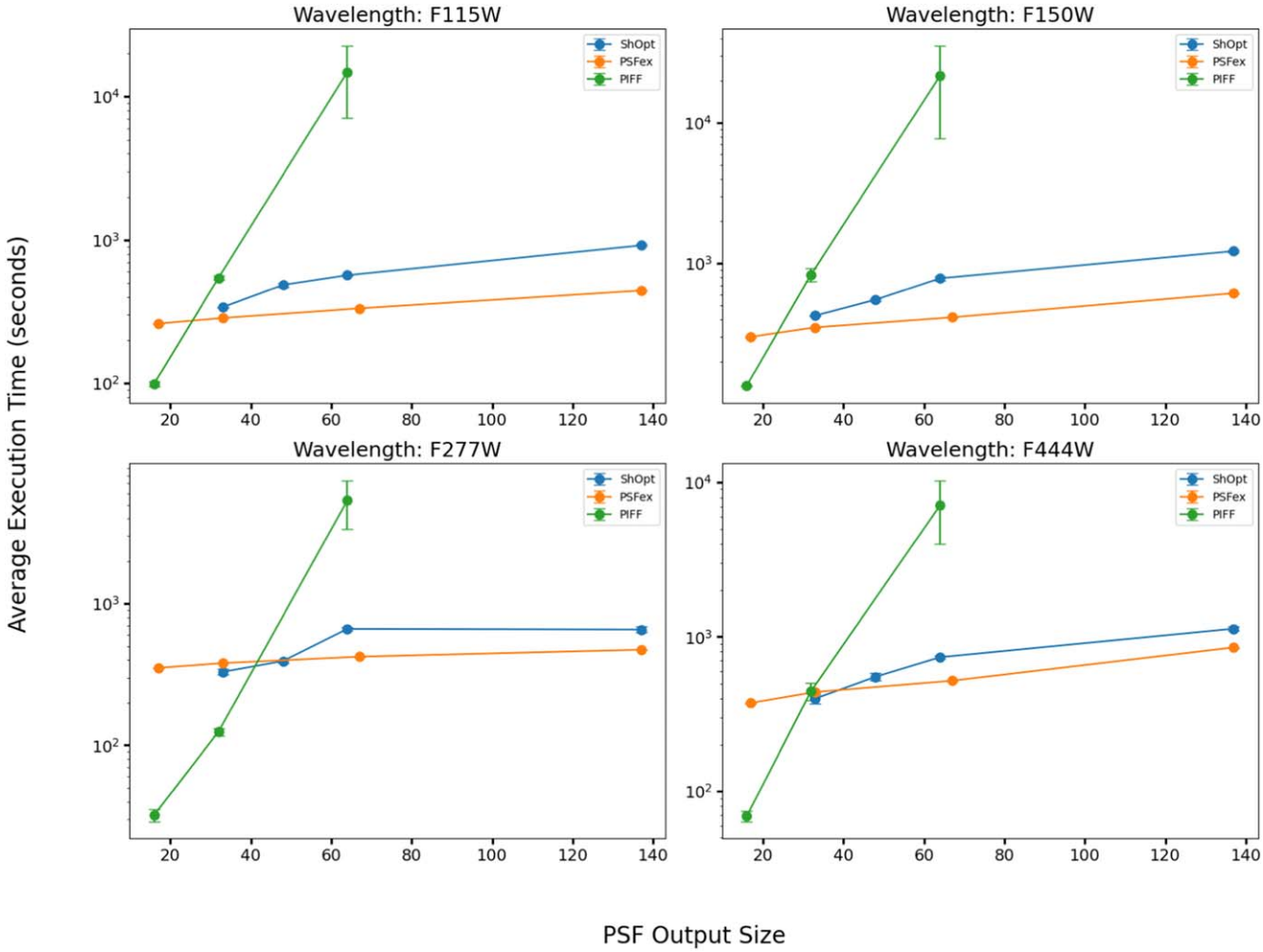


Figure 13. Average execution time as a function of the output size of the PSF side length plotted on a log scale.

Table 4
Residuals in the Adaptive Second Moments of Stars and PSF Models Across all Data Sets Used in This Work

			MSE		
Data	Fitter	Wavelength	$\sigma_{\rm HSM} \times 10^{-2}$	$g_1 \times 10^{-2}$	$g_2 \times 10^{-2}$
Simulated Single Exposure	PSFEx	F115W	0.134 ± 0.138	0.06 ± 0.01	0.17 ± 0.05
Simulated Single Exposure	ShOpt	F115W	0.248 ± 0.032	0.37 ± 0.01	0.23 ± 0.01
Simulated Single Exposure	PSFEx	F150W	0.46 ± 0.12	0.17 ± 0.01	0.24 ± 0.01
Simulated Single Exposure	ShOpt	F150W	0.56 ± 0.13	0.35 ± 0.01	0.23 ± 0.01
Simulated Single Exposure	PSFEx	F277W	2.48 ± 4.30	0.12 ± 0.02	0.10 ± 0.02
Simulated Single Exposure	ShOpt	F277W	15.3 ± 10.90	0.61 ± 0.03	1.68 ± 0.06
Simulated Single Exposure	PSFEx	F444W	1.75 ± 0.67	0.16 ± 0.02	0.29 ± 0.04
Simulated Single Exposure	ShOpt	F444W	3.46 ± 0.89	0.67 ± 0.05	1.12 ± 0.09
Simulated Mosaics	PSFEx	F115W	17.70 ± 9.65	1.63 ± 0.63	0.86 ± 0.28
Simulated Mosaics	ShOpt	F115W	18.44 ± 10.60	1.57 ± 0.62	0.83 ± 0.26
Simulated Mosaics	PIFF	F115W	18.80 ± 10.90	1.52 ± 0.60	0.93 ± 0.31
Simulated Mosaics	PSFEx	F150W	7.96 ± 5.58	0.72 ± 0.51	0.07 ± 0.02
Simulated Mosaics	ShOpt	F150W	7.35 ± 5.00	0.70 ± 0.50	0.07 ± 0.02
Simulated Mosaics	PIFF	F150W	7.53 ± 5.35	0.76 ± 0.53	0.07 ± 0.02
Simulated Mosaics	PSFEx	F277W	2.29 ± 0.58	0.42 ± 0.13	0.17 ± 0.05
Simulated Mosaics	ShOpt	F277W	2.83 ± 0.83	0.44 ± 0.13	0.16 ± 0.04
Simulated Mosaics	PIFF	F277W	5.19 ± 0.81	0.43 ± 0.17	0.13 ± 0.04
Simulated Mosaics	PSFEx	F444W	1.08 ± 0.33	0.17 ± 0.05	0.30 ± 0.17

Table 4
(Continued)

Data	Fitter	Wavelength	MSE		
			$\sigma_{\rm HSM} \times 10^{-2}$	$g_1 \times 10^{-2}$	$g_2 \times 10^{-2}$
Simulated Mosaics	ShOpt	F444W	0.98 ± 0.27	0.16 ± 0.05	0.32 ± 0.18
Simulated Mosaics	PIFF	F444W	2.28 ± 0.31	0.30 ± 0.08	0.30 ± 0.17
April Mosaics	PSFEx	F115W	0.67 ± 0.10	0.33 ± 0.05	0.19 ± 0.05
April Mosaics	ShOpt	F115W	0.78 ± 0.10	0.31 ± 0.05	0.20 ± 0.05
April Mosaics	PSFEx	F150W	0.38 ± 0.06	0.12 ± 0.02	0.04 ± 0.01
April Mosaics	ShOpt	F150W	0.56 ± 0.06	0.11 ± 0.02	0.03 ± 0.00
April Mosaics	PSFEx	F277W	1.19 ± 0.23	0.05 ± 0.01	0.04 ± 0.01
April Mosaics	ShOpt	F277W	0.90 ± 0.13	0.04 ± 0.01	0.03 ± 0.01
April Mosaics	PSFEx	F444W	2.02 ± 1.49	0.02 ± 0.01	0.01 ± 0.00
April Mosaics	ShOpt	F444W	2.00 ± 1.63	0.03 ± 0.01	0.01 ± 0.00

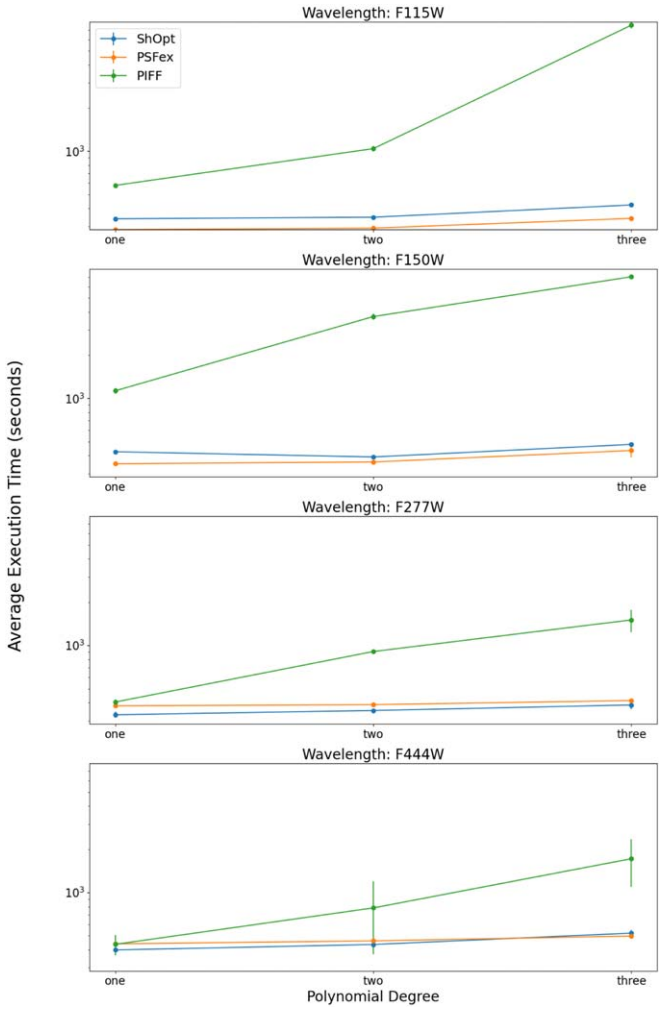


Figure 14. Average execution time as a function of the degree of the interpolating polynomial on (33, 33) PSF models plotted on a log scale.

for the larger pixel scale of the Dark Energy Camera—does not execute in a reasonable timeframe even for PSF sizes that are (64, 64) pixels in size.

In its current state, `ShOpt` produces an excellent model of the NIRCam PSF for all three data sets and is able to produce these models extremely fast.

Several enhancements are planned for `ShOpt` that may improve the quality of its PSF models. For example, `ShOpt` builds models strictly in the native pixel scale of the image; implementing super- or subsampling of the image pixel scale would align `ShOpt` with the standards of other PSF software. A more advanced PCA method for PSF reconstruction, proposed in Nie et al. (2023), is slated for future integration in our `PCA` mode. There is also a method of nonnegative PCA outlined in Bro & De Jong (1997) that may be useful to incorporate. These upgrades, among others, will be featured in future `ShOpt` releases.

`ShOpt` incorporates several innovative techniques that could benefit other PSF modeling efforts. These include leveraging manifold properties when fitting analytic profiles; using low-dimensional reconstructions for pixel-basis fits; and pixel-by-pixel parallelization during fitting of spatial variation. It has demonstrated the ability to bridge the speed of PSFEx with the advances of PIFF, making it a viable PSF fitter for next-generation astronomical observatories.

Acknowledgments

This work was supported by a Northeastern University Undergraduate Research and Fellowships PEAK Experiences Award. E.B. was also supported by a Northeastern University Physics Department Co-op Research Fellowship and by a Dean's College of Science Undergraduate Research Fellowship. This paper underwent internal review from the COSMOS-Web collaboration, and J.M. and E.B. thank everyone on this team for their insights. Additionally, E.B. thanks Professor David Rosen for valuable insights during the early stages of this work. Support for the COSMOS-Web survey was provided by NASA through grant JWST-GO-01727 and HST-AR-15802 awarded by the Space Telescope Science Institute, which is operated by the Association of Universities for Research in Astronomy, Inc., under NASA contract NAS 5-26555. This work was made possible by utilizing the CANDIDE cluster at the Institut d'Astrophysique de Paris, which was funded through grants from the PNCG, CNES, DIM-ACAV, and the Cosmic Dawn Center and maintained by Stephane Rouberol. Further support was provided by Research Computing at Northeastern University.

Appendix A
Additional PSF Diagnostic Figures

Figures 15–22 show additional PSF diagnostics. Figure 15 shows MAE and χ^2_ν errors for PSFex and Figures 16–22 show MRE plots for ShOpt and PSFex in the wavelengths not shown in the main text.

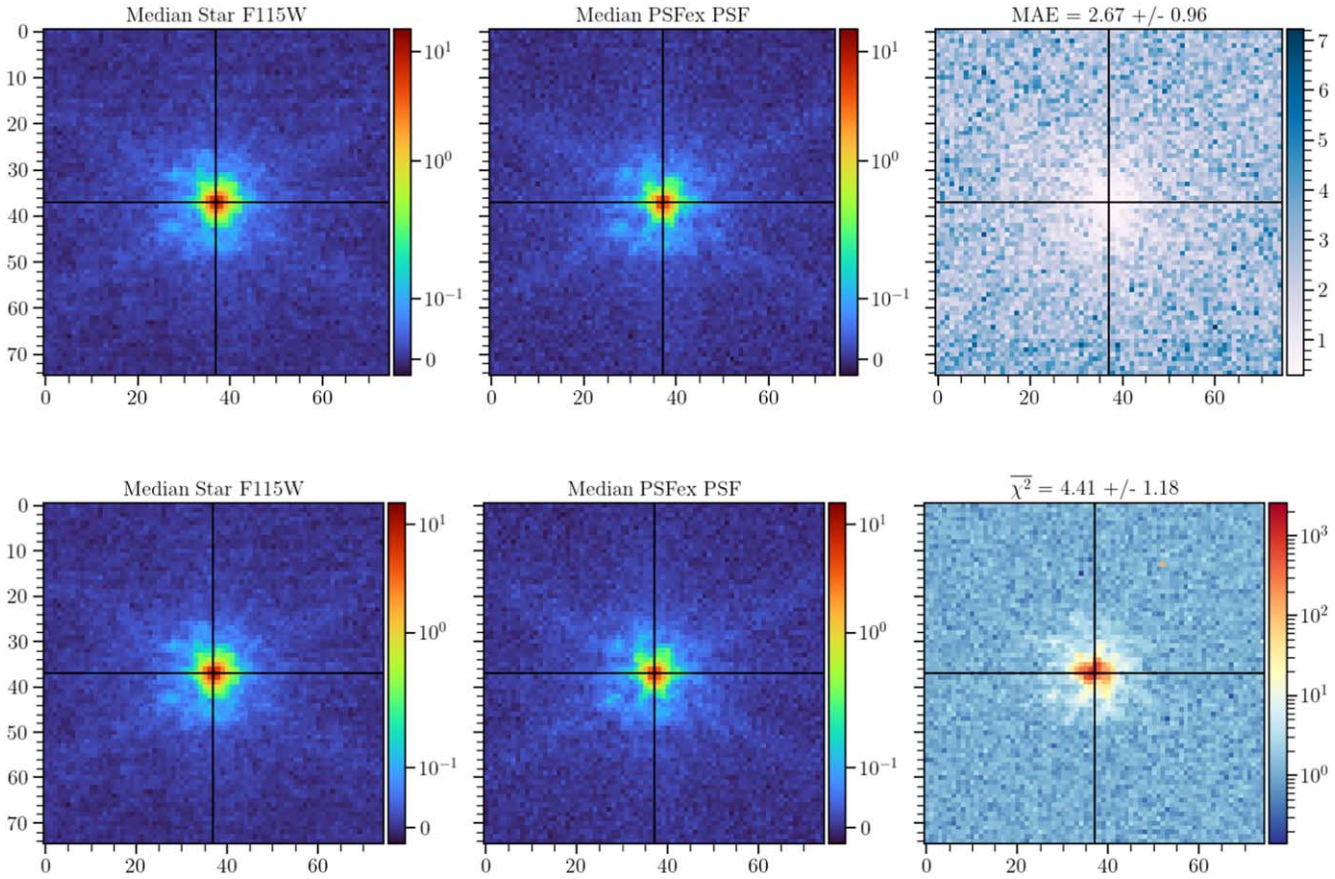


Figure 15. Examples of mean average error (top panel) and χ^2 residual figures (bottom panel), shown here for the simulated mosaics in F115W.

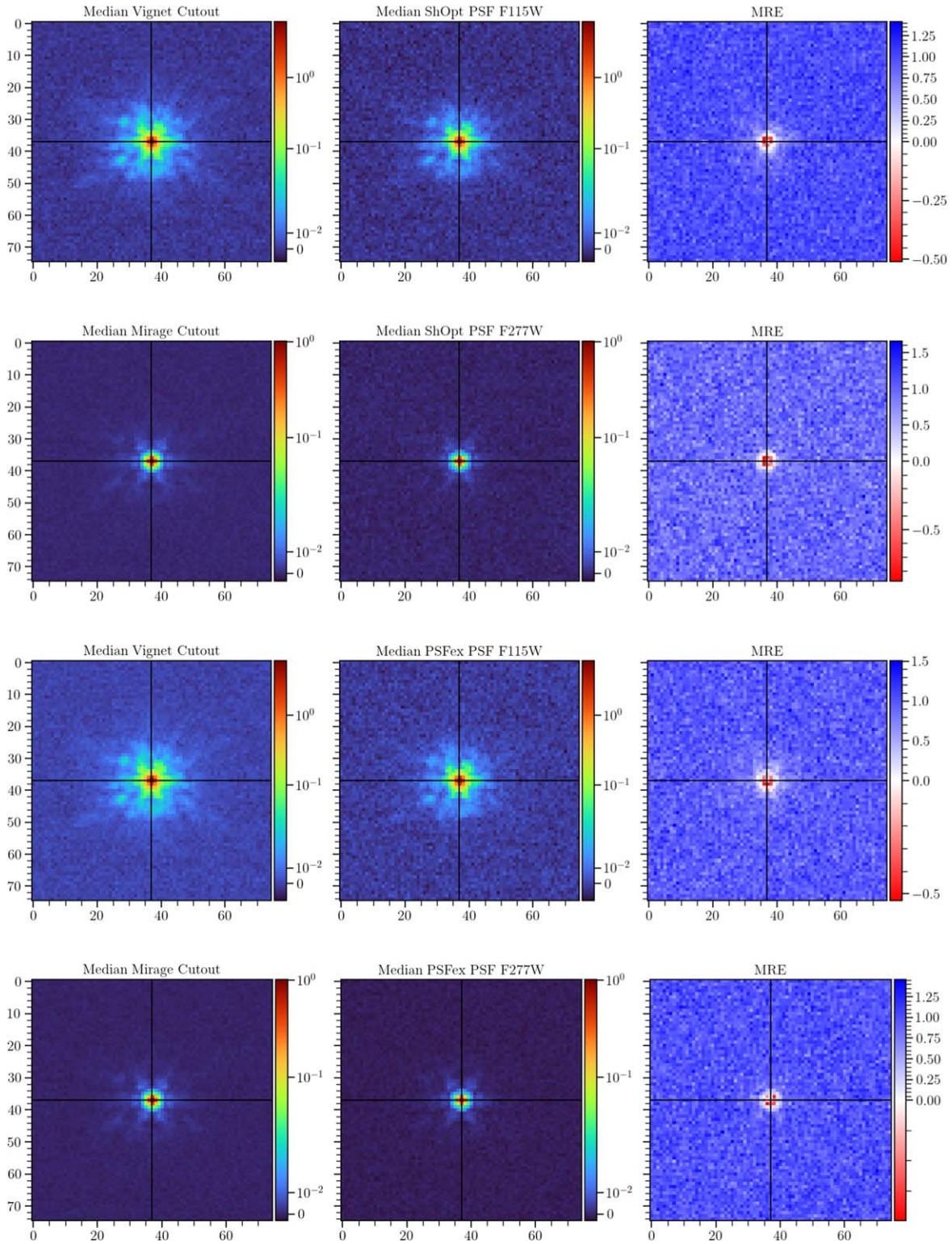


Figure 16. Mean relative error between MIRAGE input point source images and PSF models for the F115W and F150W bandpasses. Panels and color bars are the same as in Figure 7.

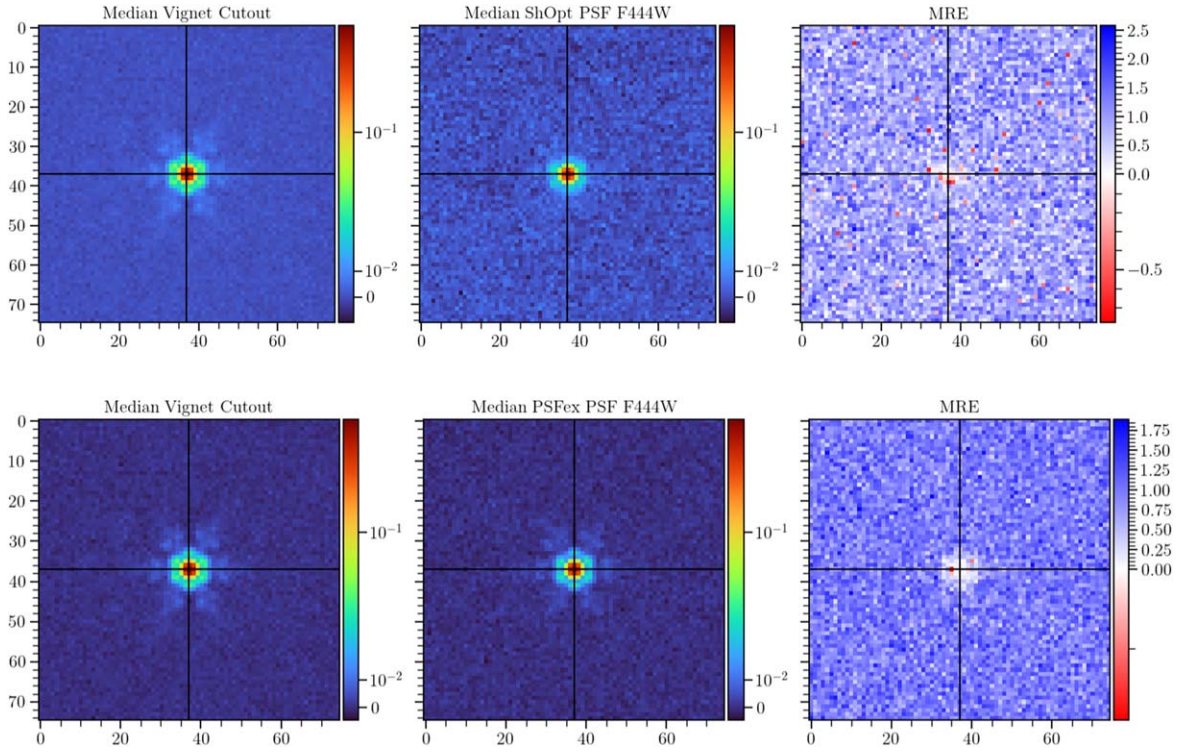


Figure 17. Mean relative error between MIRAGE input point source images and PSF models for the F444W filter. Panels and color bars are the same as in Figure 7.

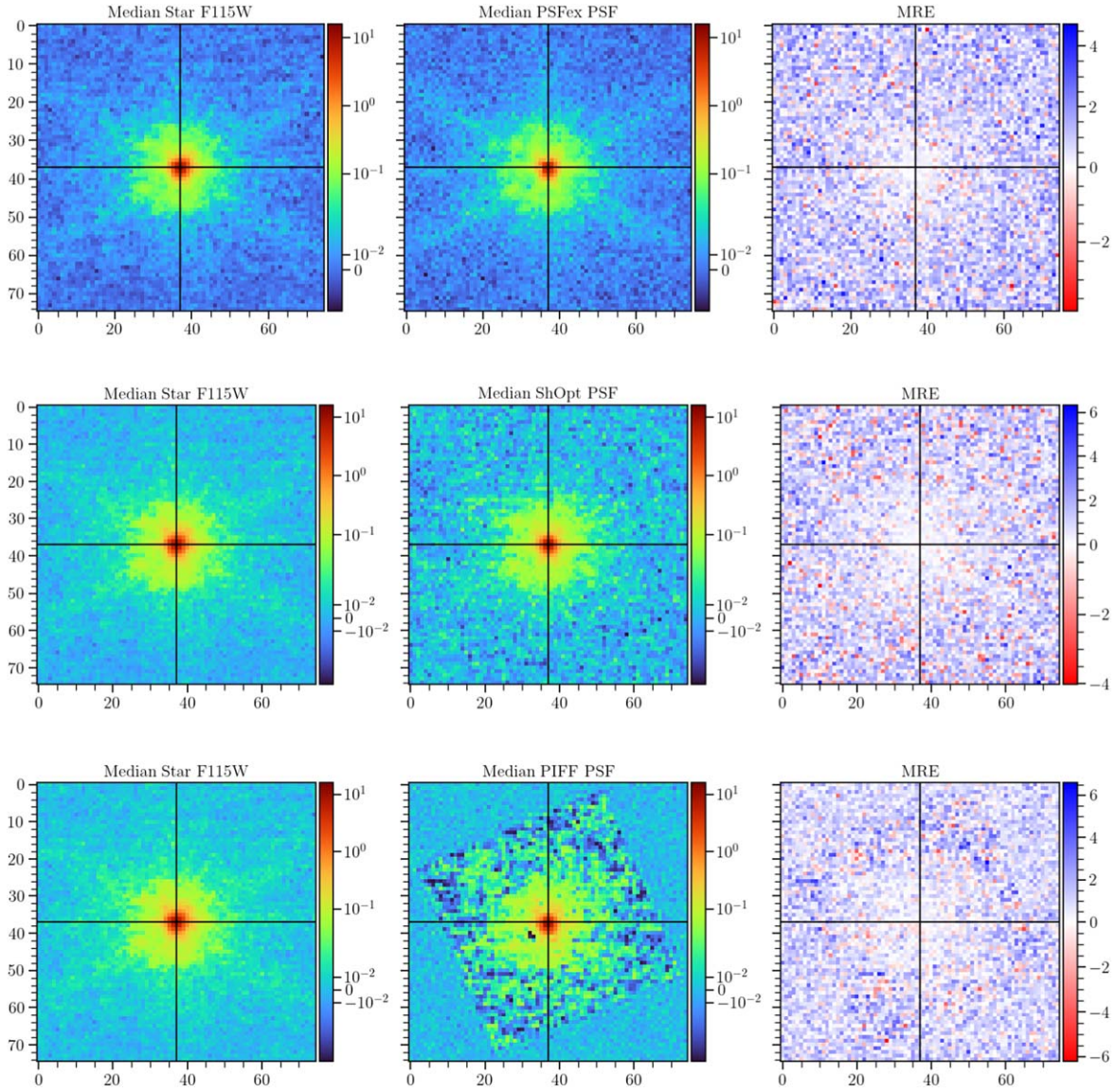


Figure 18. Mean relative error between stars and PSF models for simulated mosaics in the F115W bandpass. The left panels show the median of the vignettes. The panels and color bars are the same as in Figure 9.

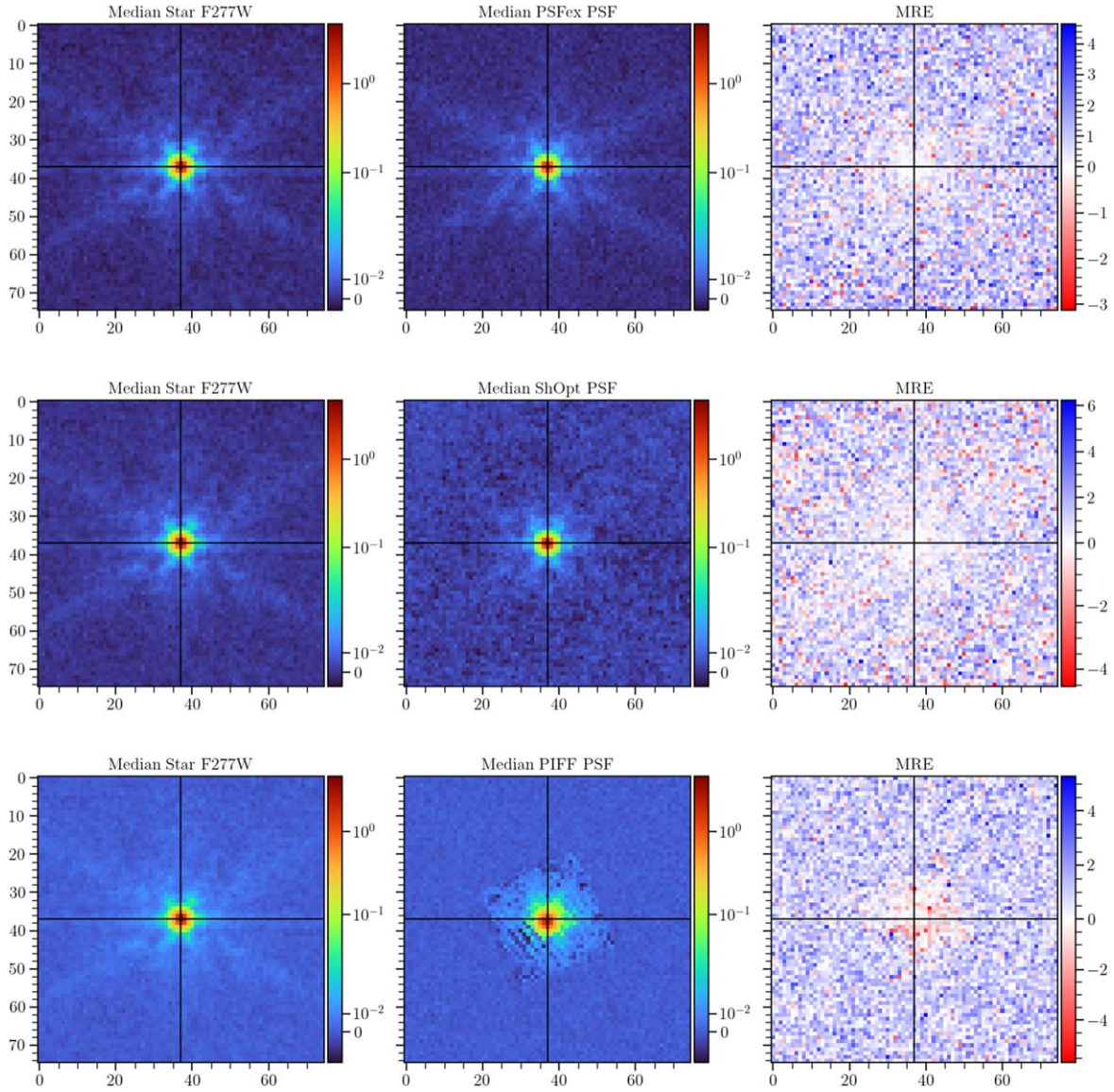


Figure 19. Evaluation of mean relative error between stars and PSF models for simulated mosaics in the F277W bandpass. The left panels show the median of the vignettes. The panels and color bars are the same as in Figure 9.

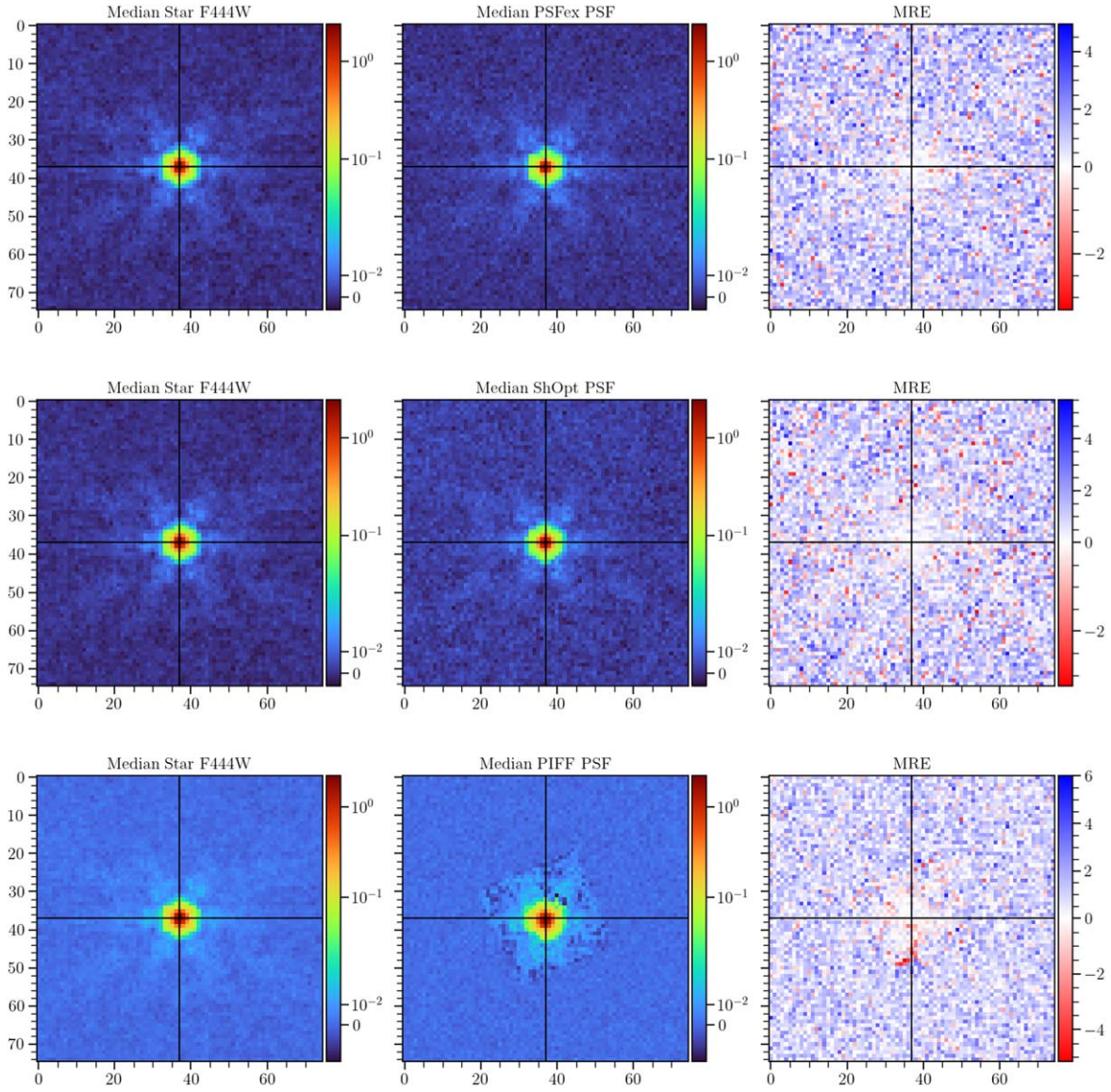


Figure 20. Evaluation of mean relative error between stars and PSF models for simulated mosaics in the F444W bandpass. The left panels show the median of the vignettes. The panels and color bars are the same as in Figure 9.

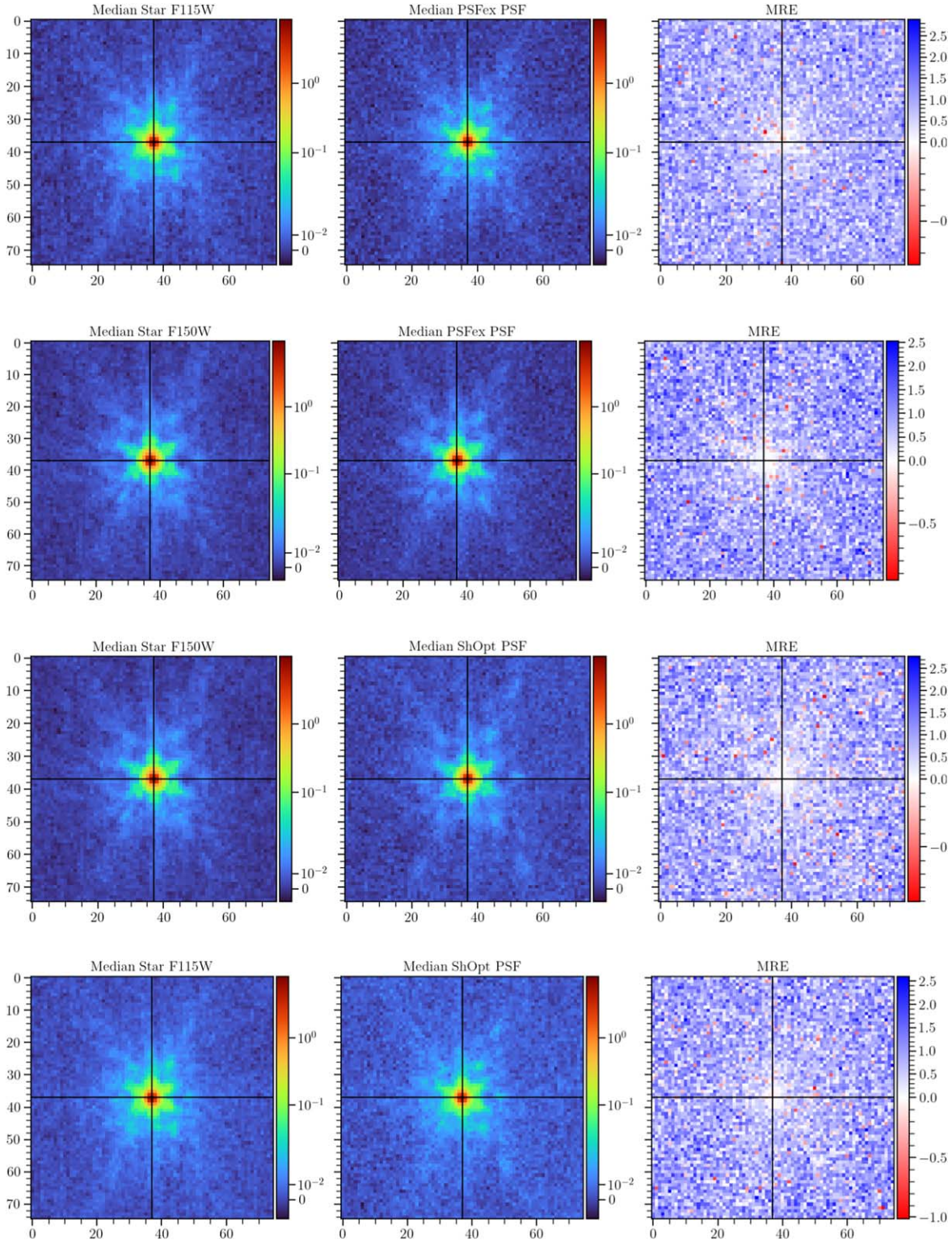


Figure 21. Mean relative error between stars and PSF models for real data mosaics in the F115W and F150W bandpasses. The color bars are the same as in Figure 11.

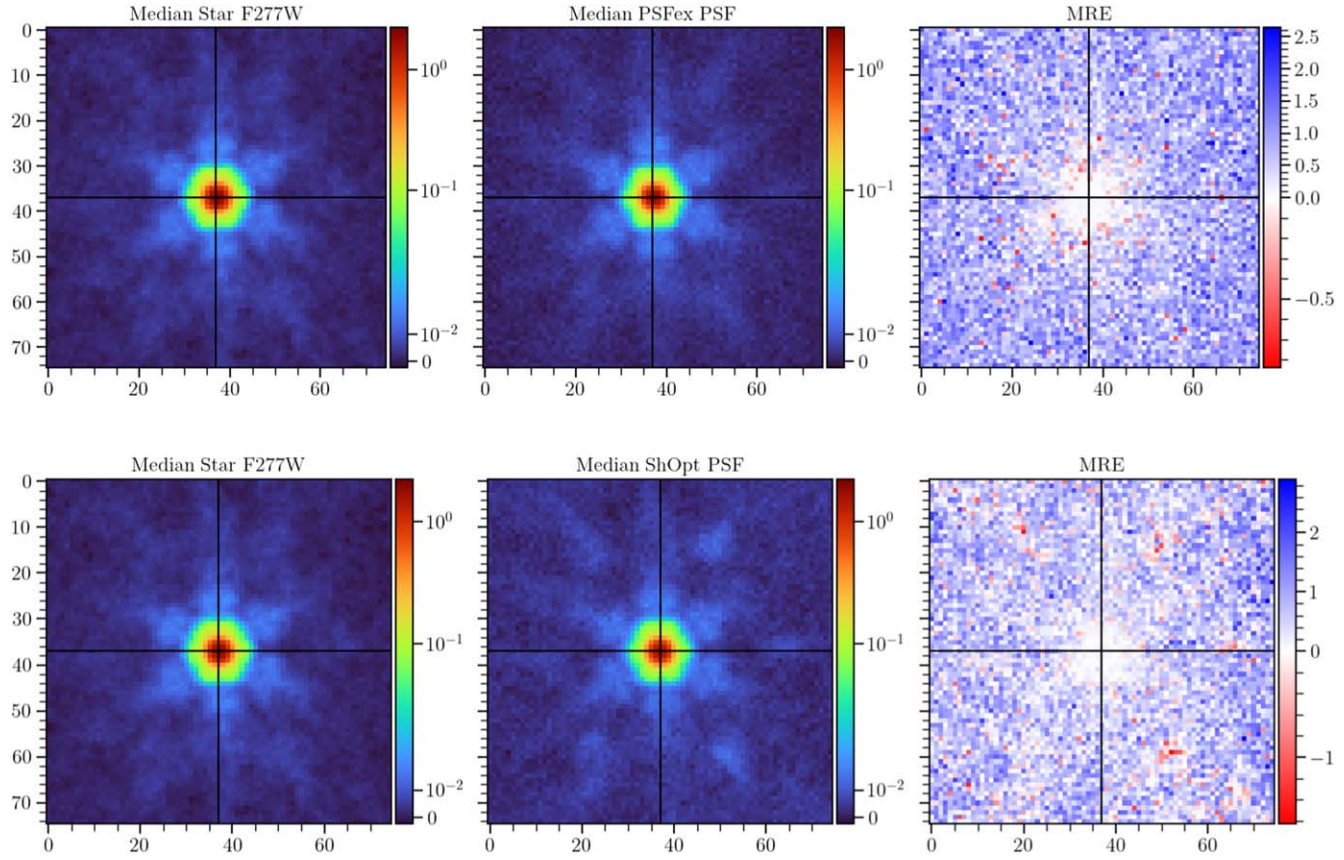


Figure 22. Evaluation of mean relative error between stars and PSF models for real data mosaics in the F277W bandpass. The panels and color bars are the same as in Figure 11.

Appendix B
Running ShOpt

`export JULIA_NUM_THREADS=auto # On Windows set JULIA_NUM_THREADS=auto julia shopt.jl [configdir] [outdir] [catalog.fits]`

See our `TutorialNotebook.ipynb` or our `README.md` for more detailed setup instructions on our GitHub.

Appendix C
Testing Configs

C.1. Source Extractor Config

```
# Default configuration file for SExtractor 2.25.0
# #--------------------------------Catalog ------------------------------

CATALOG_NAME catalog.fits # name of the output catalog
CATALOG_TYPE FITS_LDAC
PARAMETERS_NAME sextractor.param # name of the file containing
  catalog contents

#------------------------------- Extraction ----------------------------------

DETECT_TYPE CCD # CCD (linear) or PHOTO (with gamma correction)
DETECT_MINAREA 8 # min. # of pixels above threshold
DETECT_MAXAREA 0 # max. # of pixels above threshold (0=unlimited)
THRESH_TYPE RELATIVE # threshold type: RELATIVE (in sigmas)
# or ABSOLUTE (in ADUs)
DETECT_THRESH 2 # <sigmas> or <threshold>,<ZP> in mag.arcsec-2
ANALYSIS_THRESH 2 # <sigmas> or <threshold>,<ZP> in mag.
  arcsec-2

FILTER Y # apply filter for detection (Y or N)?
FILTER_NAME Gauss_2.5_5x5.conv # name of the file containing the filter
FILTER_THRESH # Threshold[s] for retina filtering

DEBLEND_NTHRESH 32 # Number of deblending subthresholds
DEBLEND_MINCONT 0.005 # Minimum contrast parameter for deblending
CLEAN Y # Clean spurious detections? (Y or N)?
CLEAN_PARAM 1.0 # Cleaning efficiency

MASK_TYPE CORRECT # type of detection MASKing: can be one of
# NONE, BLANK or CORRECT

#------------------------------WEIGHTing -----------------------------

WEIGHT_TYPE MAP_WEIGHT # type of WEIGHTing: NONE,
  BACKGROUND,
# MAP_RMS, MAP_VAR or MAP_WEIGHT
RESCALE_WEIGHTS Y # Rescale input weights/variances (Y/N)?
WEIGHT_IMAGE weight.fits # weight-map filename
WEIGHT_GAIN Y # modulate gain (E/ADU) with weights? (Y/N)
WEIGHT_THRESH 0 # weight threshold[s] for bad pixels

#------------------------------FLAGging ------------------------------

FLAG_IMAGE flag.fits # filename for an input FLAG-image
FLAG_TYPE OR # flag pixel combination: OR, AND, MIN, MAX
# or MOST

#-----------------------Differential Geometry Map ----------------------

DGEO_TYPE NONE # Differential geometry map type: NONE or PIXEL
```

```
DGEO_IMAGE dgeo.fits # Filename for input differential geometry image

#———————————— Photometry ————————————

PHOT_APERTURES 31 # MAG_APER aperture diameter(s) in pixels
PHOT_AUTOPARAMS 2, 2.5 # MAG_AUTO parameters: <Kron_fact>,
  <min_radius>
PHOT_PETROPARAMS 2.0, 3.5 # MAG_PETRO parameters:
  <Petrosian_fact>,
# <min_radius>
PHOT_AUTOAPERS 0.0,0.0 # <estimation>,<measurement> minimum
  apertures
# for MAG_AUTO and MAG_PETRO
PHOT_FLUXFRAC 0.5 # flux fraction[s] used for FLUX_RADIUS

SATUR_LEVEL 37000.0 # level (in ADUs) at which arises saturation
SATUR_KEY SATURATE # keyword for saturation level (in ADUs)

MAG_ZEROPOINT 28.086519392 # magnitude zero-point
MAG_GAMMA 4.0 # gamma of emulsion (for photographic scans)
GAIN 0.0 # detector gain in e-/ADU
GAIN_KEY GAIN # keyword for detector gain in e-/ADU
PIXEL_SCALE 0 # size of pixel in arcsec (0=use FITS WCS info)

#———————————— Star/Galaxy Separation ————————————

SEEING_FWHM 0.07 # stellar FWHM in arcsec
STARNNW_NAME default.nnw # Neural-Network_Weight table filename

#———————————— Background ————————————

BACK_TYPE AUTO # AUTO or MANUAL
BACK_VALUE 0.0 # Default background value in MANUAL mode
BACK_SIZE 128 # Background mesh: <size> or <width>,<height>
BACK_FILTERSIZE 3 # Background filter: <size> or <width>,<height>

BACKPHOTO_TYPE LOCAL # can be GLOBAL or LOCAL
BACKPHOTO_THICK 24 # thickness of the background LOCAL annulus
BACK_FILTTHRESH 0.0 # Threshold above which the background-
# map filter operates

#———————————— Check Image ————————————

CHECKIMAGE_TYPE -BACKGROUND, APERTURES # Check-image
  type(s)
CHECKIMAGE_NAME im.sub.fits, im.aper.fits # Filename for the check-
  image(s)
```

C.2. ShOpt Config

```
saveYaml: true #save this file with each run

# Options: auotoencoder, PCA, smoothing.
# Make sure mode is a string with double quotes
mode: "smoothing"

# If PCA mode is enabled, how many moments do you
# want to use for your pixel-grid fit
PCAterms: 50

# The size of the smoothing kernel
lanczos: 5

# For Autoencoder mode, when enabled

NNparams:
# max number of training epochs for each pixel-grid fit
epochs: 100
# The stopping gradient of the loss function for the pixel-grid fit
minGradientPixel: 1e-5

# For fitting analytic profile
AnalyticFitParams:
# Stopping gradient for LBFGS on vignettes
minGradientAnalyticModel: 1e-6
# Stopping gradient for LBFGS on pixel-grid models
minGradientAnalyticLearned: 1e-6
# The subset of pixels you wish to fit the analytic profile to
analyticFitStampSize: 64

dataProcessing:
# Filter this SnRPercentile: 0.33
# Filter stars with analytic profile fit size s exceeding this value
sUpperBound: 1
# Filter stars with analytic profile fit size s below this value
sLowerBound: 0.075

# What plots do you want?
plots:
unicodePlots: true
normalPlots:
parametersHistogram: true
parametersScatterplot: true
cairomakiePlots:
streamplots: false
pythonPlots: false

# Degree of polynomial for spatial interpolation of PSF model
polynomialDegree: 1
# Size of pixel-grid PSF model
stampSize: 130

# How many stars are you using the train versus to validate the PSF fit
training_ratio: 0.9
# Sum flux to unity true or false
sum_pixel_grid_and_inputs_to_unity: false

# stopping gradient for LFBGS used for polynomial interpolation
polynomial_interpolation_stopping_gradient: 1e-12

# Name to prefix summary.shopt
summary_name: ''
# Do you want to save storage by only storing essential information, how to
  reconstruct the PSF and analytic models
truncate_summary_file: true

CommentsOnRun: "** This is where you can leave comments or notes to self
  on the run! **"
```

C.3. PIFF Config

```
# modules and input.wcs fields, in which case, the code will use the (less
# accurate) WCS that ships with the image in the fits file.

input:

# Input file directory
dir: "./"

# Input filename(s) and HDU extensions
```

```
image_file_name: "mosaic_nircam_f277w_COSMOS-Web_i2d.fits"
image_hdu: 1
weight_hdu: 4

# Input catalog and HDU extension
cat_file_name: "mosaic_nircam_f277w_COSMOS-Web_i2d_starcat.fits"
cat_hdu: 2

# What columns in the catalog have things we need?
x_col: XWIN_IMAGE
y_col: YWIN_IMAGE
ra_col: ALPHAWIN_J2000
dec_col: DELTAWIN_J2000

# The telescope pointing
ra: 149.9303551903936
dec: 2.380272767453749

# Leave blank if you don't know what it is!
# gain: 1

# How large should the postage stamp cutouts of the stars be?
stamp_size: 100

# Use all cores for reading the input files
nproc: -1

select:

# For bright stars, weight them equivalent to snr=100 stars, not higher.
max_snr: 100

# Remove stars with snr < 10
min_snr: 10

# Reserve 15reserve_frac: 0.15

# Reject size outliers
hsm_size_reject: True

psf:

# This type of PSF will use a separate model/interp solution for each chip.
# But all the solutions will be given in a single output file.
type: SingleChip

# Also use all cores when finding psf
nproc: -1

outliers:

type: Chisq

# The threshold is given in terms of nsigma equivalent
nsigma: 4

# Only remove at most 3max_remove: 0.03

model:
# This model uses a grid of pixels to model the surface brightness distribution.
type: PixelGrid
scale: 0.03 # NIRCam ative pixel scale
size: 75

interp:

# This interpolator does some of the model solving when interpolating
# to handle degenerate information from masking
# and the fact that the pixels are smaller than native.
type: BasisPolynomial
order: 1
```

C.4. PSFEx Config

```
# Default configuration file for PSFEx 3.17.1
# EB 2016-06-28
#

#-------------------------PSF model -------------------------

BASIS_TYPE PIXEL # NONE, PIXEL, GAUSS-LAGUERRE or FILE
#BASIS_NUMBER 30 # Basis number or parameter
PSF_SAMPLING 0 # Sampling step in pixel units (0.0=auto)
PSF_SIZE 261 # Image size of the PSF model
PSF_RECENTER Y
#------------------- Point source measurements -------------------

CENTER_KEYS XWIN_IMAGE,YWIN_IMAGE # Catalogue parameters
  for source pre-centering
PHOTFLUX_KEY FLUX_APER(1) # Catalogue parameter for photo-
  metric norm.
PHOTFLUXERR_KEY FLUXERR_APER(1) # Catalogue parameter for
  photometric error

#---------------------PSF variability -------------------------

PSFVAR_KEYS XWIN_IMAGE,YWIN_IMAGE # Catalogue or FITS
  (preceded by:) params
PSFVAR_GROUPS 1,1 # Group tag for each context key
PSFVAR_DEGREES 1 # Polynom degree for each group

#---------------------Sample selection ------------------------

SAMPLE_AUTOSELECT Y # Automatically select the FWHM (Y/N)?
SAMPLEVAR_TYPE NONE # File-to-file PSF variability: NONE or
  SEEING
SAMPLE_FWHMRANGE 1,20 # Allowed FWHM range (2.7,3.2)
SAMPLE_VARIABILITY 0.3 # Allowed FWHM variability
  (1.0=100SAMPLE_MINSN 50 # Minimum S/N for a source to be used
SAMPLE_MAXELLIP 0.3 # Maximum (A-B)/(A+B) for a source to be used

#---------------------Output catalogs -------------------------

OUTCAT_TYPE FITS_LDAC # NONE, ASCII_HEAD, ASCII,
  FITS_LDAC
OUTCAT_NAME psfex_out.cat # Output catalog filename

#------------------------- Check-plots -------------------------

CHECKPLOT_DEV PDF # NULL, XWIN, TK, PS, PSC, XFIG, PNG,
# JPEG, AQT, PDF or SVG
CHECKPLOT_RES 0 # Check-plot resolution (0=default)
CHECKPLOT_ANTIALIAS Y # Anti-aliasing using convert (Y/N)?
CHECKPLOT_TYPE NONE
CHECKPLOT_NAME

#------------------------ Check-Images -------------------------

CHECKIMAGE_TYPE CHI,SAMPLES,RESIDUALS,SNAPSHOTS,
  -SYMMETRICAL
# or MOFFAT,-MOFFAT,-SYMMETRICAL
CHECKIMAGE_NAME chi.fits,samp.fits,resi.fits,snap.fits, minus_symm.fits
```

```
# Check-image filenames
CHECKIMAGE_CUBE Y
```

C.5. Shell Script

This is given to inform how much memory we requested from Discovery for purposes of speed testing.

```
#!/bin/bash
#SBATCH –nodes=1
#SBATCH –mem=16G
#SBATCH –cpus-per-task=4
#SBATCH –time=24:00:00
#SBATCH –partition=short
#SBATCH –job-name=150_A6
#SBATCH –ntasks=1
#SBATCH –constraint=zen2 # Requesting specific CPU architecture
pwd
source /work/mccleary_group/berman.ed/minicondaInstall/bin/activate
python get_galaxy_cutouts.py -config configs/box_cutter.yaml file #file is a
   placeholder for a fits file
```

Appendix D
Additional ShOpt Checkplots and Outputs

D.1. Diagnostic Material

`ShOpt` provides the following stream plots (Figure 23) to give the user an inclination toward how the PSF is changing across the field of view. We also have a Julia script, `reader.jl` that reads in the `summary.shopt` file and provides easy PSF reconstruction. If you want to do your analysis in Python, we also have Python code available for reading in `summary.shopt` files here: https://github.com/EdwardBerman/sigma-Eta-Shopt-Reader.

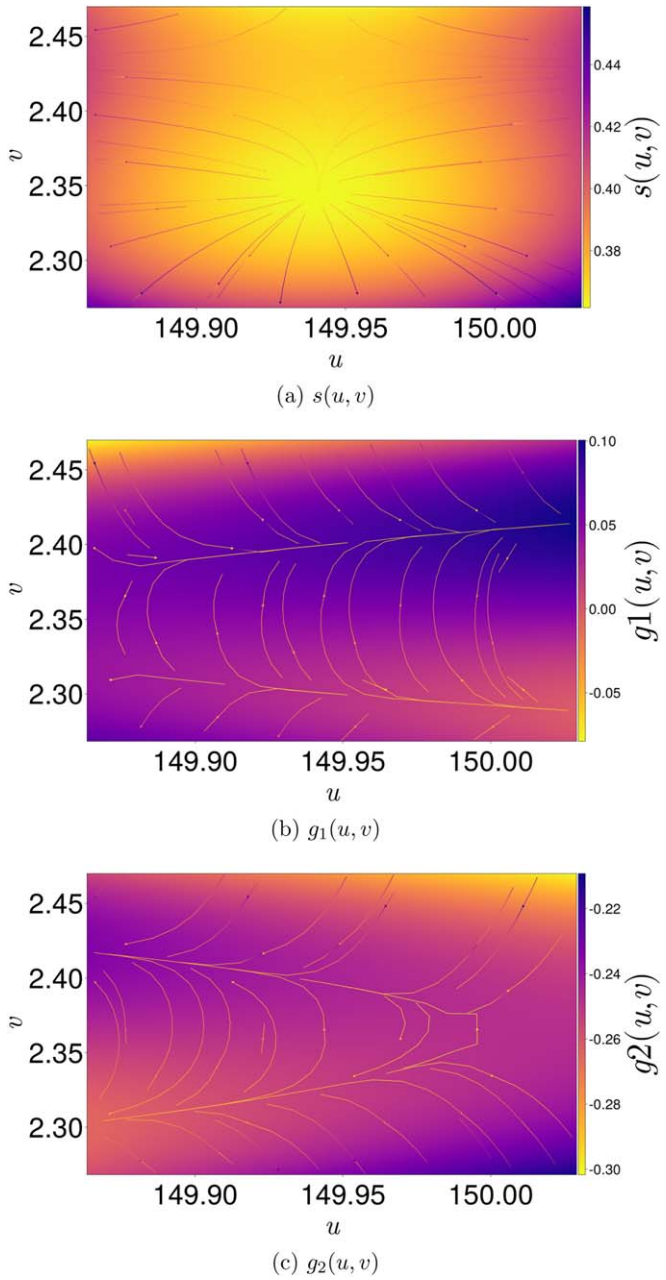
D.2. Summary.shopt

`summary.shopt` contains six relevant extensions. The first extension is named polynomial matrix, and it contains a three-dimension matrix. Two dimensions correspond to the dimensions of the input vignettes and the third dimension corresponds to the coefficients of the polynomial at that pixel. The second extension contains all data relevant to learned parameters $[s, g_1, g_2]$ as well as the (u, v) coordinates at each star. We also measure the mean relative error between stars and their pixel-grid fits before the polynomial interpolation step. Note that only stars that make it through all filters are contained. The third, and fourth extensions contain three-dimensional arrays corresponding to the input vignettes and the pixel-grid fits of the vignettes. The fifth extension provides flags that tell you the indices of stars that were filtered out of the final interpolation step. The sixth extension tells you how to find $[s, g_1, g_2]$ at an arbitrary (u, v). There is also a mode that only outputs the first, second, and sixth extensions for reasons related to storage concerns. This is enabled by default.

D.3. Command Line Outputs

As an extra convenience, we give users the option to display some of the diagnostic material to the terminal using `UnicodePlots.jl`. This may be useful for less scrupulous more exploratory runs of our software or for users looking for a quick sanity check that everything ran correctly without having to navigate to an output directory and open all of the saved checkplots.

We also print out to the terminal some key information about what is happening as the program runs, including what the program is doing, how many and which stars are failing or being filtered, progress on fitting, and how long particular portions of the code took to run.



(a) $s(u, v)$

(b) $g_1(u, v)$

(c) $g_2(u, v)$

Figure 23. Stream plots demonstrating how variables $[s, g_1, g_2]$ vary across the field of view (in astrometric coordinates (u, v)) for the F115W simulated mosaic image. Recall that s corresponds to size, and g_1, g_2 correspond to shear.

Appendix E
Petal Diagrams

We speculate that petal diagrams may be able to approximate the spiky natures of JWST PSFS. Consider $r = A\cos(k\theta + \gamma)$, shown below in Figure 24 for different $[A, k]$ values where $\gamma = 0$. In practice, $[A, k, \gamma]$ could be learnable parameters. We

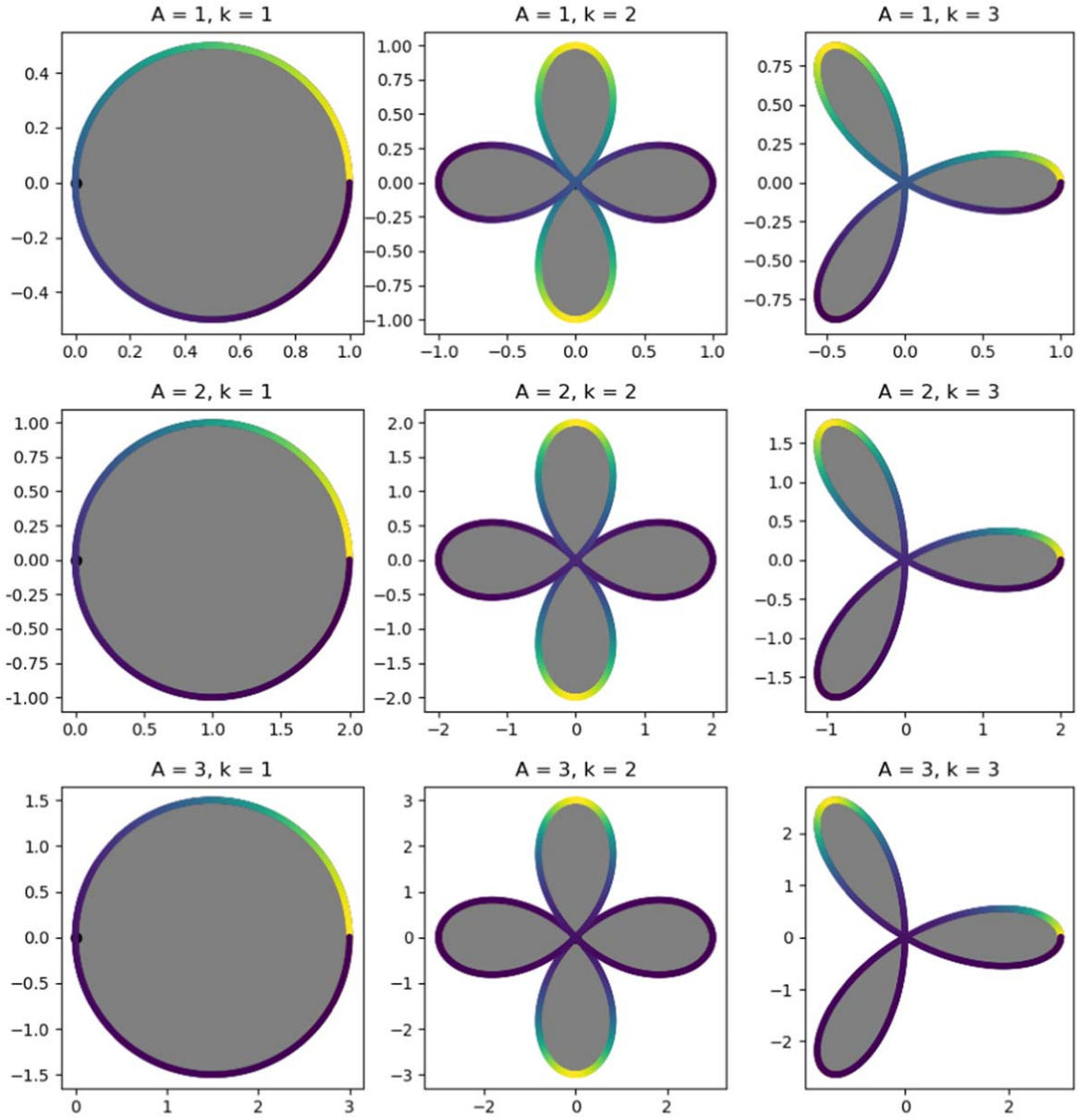


Figure 24. Petal diagrams approximating different PSFs. Different k values correspond to PSF with different numbers of spikes. The A value determines how long the spikes are. Here, all of the pixels inside the petals (gray) are set to a constant value, and everything outside is 0.

could then choose some $f(r) \propto \frac{1}{r}$ such that the gray fades from black to white. We would define $f(r)$ piecewise such that it is 0 outside of the petal and decreases radially with r inside the petal. The upshot of this approach is that we can just look at the learned k and immediately know if our PSF captures the correct number of wings. Alternatively, Bergé et al. (2019) introduced exponential shapelets with an orthogonal separation of r and θ, which may also be useful.

ORCID iDs

Edward M. Berman https://orcid.org/0000-0002-8494-3123
Jacqueline E. McCleary https://orcid.org/0000-0002-9883-7460
Anton M. Koekemoer https://orcid.org/0000-0002-6610-2048
Maximilien Franco https://orcid.org/0000-0002-3560-8599
Nicole E. Drakos https://orcid.org/0000-0003-4761-2197
Daizhong Liu https://orcid.org/0000-0001-9773-7479
James W. Nightingale https://orcid.org/0000-0002-8987-7401
Marko Shuntov https://orcid.org/0000-0002-7087-0701
Diana Scognamiglio https://orcid.org/0000-0001-8450-7885
Richard Massey https://orcid.org/0000-0002-6085-3780
Guillaume Mahler https://orcid.org/0000-0003-3266-2001
Henry Joy McCracken https://orcid.org/0000-0002-9489-7765
Brant E. Robertson https://orcid.org/0000-0002-4271-0364
Andreas L. Faisst https://orcid.org/0000-0002-9382-9832
Caitlin M. Casey https://orcid.org/0000-0002-0930-6466
Jeyhan S. Kartaltepe https://orcid.org/0000-0001-9187-3605

References

Absil, P.-A., & Mahony, R. 2008, Optimization Algorithms on Matrix Manifolds (Princeton, NJ: Princeton Univ. Press) xvi+224
Bagley, M. B., Finkelstein, S. L., Rojas-Ruiz, S., et al. 2022, ApJ, 961, 209
Beichman, C. A., Rieke, M., Eisenstein, D., et al. 2012, Proc. SPIE, 8442, 84422N

Bergé, J., Massey, R., Baghi, Q., & Touboul, P. 2019, MNRAS, 486, 544
Bergmann, R. 2022, JOSS, 7, 3866
Berman, E., & McCleary, J. 2024, JOSS, 9, 6144
Bernstein, G. M., & Jarvis, M. 2002, AJ, 123, 583
Bertin, E. 2011, in ASP Conf. Ser. 442, Astronomical Data Analysis Software and Systems XX, ed. I. N. Evans et al. (San Francisco, CA: ASP), 435
Bertin, E., & Arnouts, S. 1996, A&AS, 117, 393
Bonnet, H., & Mellier, Y. 1995, A&A, 303, 331
Boumal, N. 2023, An Introduction to Optimization on Smooth Manifolds (Cambridge: Cambridge Univ. Press)
Bro, R., & De Jong, S. 1997, J. Chemom., 11, 393
Bushouse, H., Eisenhamer, J., Dencheva, N., et al. 2023, JWST Calibration Pipeline, v1.13.1, Zenodo, doi:10.5281/zenodo.7038885
Casey, C. M., Kartaltepe, J. S., Drakos, N. E., et al. 2023, ApJ, 954, 31
Cassano, F., Gouwar, J., Lucchetti, F., et al. 2024, in Proc. ACM on Programming Languages (PACMPL) (New York: ACM)
Finkelstein, S. L., Bagley, M. B., Arrabal Haro, P., et al. 2022, ApJL, 940, L55
Fu, S., Dell'Antonio, I., Chary, R.-R., et al. 2022, ApJ, 933, 84
Hilbert, B., Sahlmann, J., Volk, K., et al. 2019, spacetelescope/mirage: First github release, v1.1.1, Zenodo, doi:10.5281/zenodo.3519262
Hirata, C., & Seljak, U. 2003, MNRAS, 343, 459
Innes, M. 2018, JOSS, 3, 602
Jarvis, M., Bernstein, G. M., Amon, A., et al. 2020, MNRAS, 501, 1282
Ji, Z., Williams, C. C., Tacchella, S., et al. 2023, arXiv:2305.18518
Kokorev, V. I., Magdis, G. E., Davidzon, I., et al. 2021, ApJ, 921, 40
Krist, J. E., Hook, R. N., & Stoehr, F. 2011, Proc. SPIE, 8127, 166
Kung, H., Natesh, V., & Sabot, A. 2021, in SC21: Int. Conf. for High Performance Computing, Networking, Storage and Analysis (New York: ACM), 1
Maas, A. L. 2013, in Int. Conf. Machine Learning (Atlanta, GA: ICML)
Mandelbaum, R., Hirata, C. M., Seljak, U., et al. 2005, MNRAS, 361 1287
McCleary, J. E., Everett, S. W., Shaaban, M. M., et al. 2023, AJ, 166, 134
Mogensen, P. K., & Riseth, A. N. 2018, JOSS, 3, 615
Nie, L., Shan, H., Li, G., et al. 2023, arXiv:2308.14065
Northeastern University Research Computing 2024, Hardware Overview, https://rc-docs.northeastern.edu/en/latest/hardware/hardware_overview.html
Perrin, M. D., Sivaramakrishnan, A., Lajoie, C.-P., et al. 2014, Proc. SPIE, 9143, 91433X
Perrin, M. D., Soummer, R., Elliott, E. M., Lallo, M. D., & Sivaramakrishnan, A. 2012, Proc. SPIE, 8442, 84423D
Revels, J., Lubin, M., & Papamarkou, T. 2016, arXiv:1607.07892
Rieke, M. J., Baum, S. A., Beichman, C. A., et al. 2003, Proc. SPIE, 4850, 478
Rieke, M. J., Kelly, D., & Horner, S. 2005, Proc. SPIE, 5904, 1
Robertson, B. E. 2022, ARA&A, 60, 121
Rosen, D. M., Carlone, L., Bandeira, A. S., & Leonard, J. J. 2019, IJRR, 38, 95
Rowe, B., Jarvis, M., Mandelbaum, R., et al. 2015, A&C, 10, 121
Sabot, A., Natesh, V., Kung, H. T., & Ting, W.-T. 2023, arXiv:2304.05544
Sargent, M. T., Carollo, C. M., Lilly, S. J., et al. 2007, ApJS, 172, 434
Scarlata, C., Carollo, C. M., Lilly, S., et al. 2007, ApJS, 172, 406
Weaver, J. R., Kauffmann, O. B., Ilbert, O., et al. 2022, ApJS, 258, 11
York, D. G., Adelman, J., Anderson, J. E., Jr., et al. 2000, AJ, 120, 1579
Zhang, T., Almoubayyed, H., Mandelbaum, R., et al. 2023, MNRAS, 520, 2328