Driving Discovery in Astronomy Using Scalable Computing and Fast Algorithms

Steven Stetzler

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2025

Reading Committee:

Mario Jurić, Chair

Andrew Connolly

Thomas Quinn

Program Authorized to Offer Degree:
Astronomy

University of Washington

Abstract

Driving Discovery in Astronomy Using Scalable Computing and Fast Algorithms

Steven Stetzler

Chair of the Supervisory Committee:
Mario Jurić
Astronomy

The Vera C. Rubin Observatory's Legacy Survey of Space and Time (LSST) will produce and publicly release an imaging and catalog dataset larger than any before it. At a total data volume of 500 PB, it will only be possible to realize the full scientific potential of this survey if tools and algorithms are available that can scale to the size of the dataset. This thesis contributes to this effort through three key advancements. First, I demonstrate how cloud-based science platforms can be used to access and analyze large catalogs using a distributed computing framework. The distributed computing tools utilized are user-friendly and accessible, allowing astronomers to scale their workflows to the entire LSST catalog. Second, I illustrate how the LSST Science Pipelines–a set of tools and algorithms for processing astronomical images–can be applied to a wide-field imaging survey, verifying their scalability and applicability in processing the LSST imaging dataset. I also provide tools that enable users and research groups to perform their own image processing campaigns. Finally, I introduce an optimized version of the shift-and-stack algorithm, enhancing its efficiency for detecting faint solar system objects in multi-epoch imaging surveys. This improved algorithm provides speedup relative to a naive implementation. This optimization lays the foundation for application of shift-and-stack to the entire LSST dataset, particularly in the search for faint inner solar system objects.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Number Page

LIST OF TABLES

ACKNOWLEDGMENTS

This dissertation represents six and a half years of graduate study in the astronomy department at the University of Washington. While this dissertation was written alone, the work it represents could not have been performed without the assistance and support of many people to whom I am deeply grateful.

I'd first like to extend my appreciation to my advisor Mario Jurič for guiding me patiently through my work and always doing his best to keep me on track. I'd additionally like to thank my secondary advisors and faculty mentors—Andrew Connolly, Thomas Quinn, and Jessica Werk—for guiding me through the astronomy graduate program.

My graduate studies were supported in-part by the Krell Institute's Computational Science Graduate Fellowship program. Through this fellowship, I had the invaluable opportunity to work at Los Alamos National Laboratory. I am deeply grateful to the fellowship program for providing me with this opportunity and to my research mentor, Michael Grosskopf, for his guidance during the program and continued support afterward.

I am profoundly grateful to the teachers and mentors who shaped my academic path and made my pursuit of a graduate degree possible. This includes my undergraduate research and academic advisors—Craig Group, Craig Dukes, and Kevin Stovall—as well as my high school teachers whose influence I carry with me: Mrs. Sandler, Mrs. Westgate, Mr. Bricker, and Mrs. Lubinski.

I also want to thank the generation of YouTube content creators and their videos that sparked my love of learning and led me on my quest for knowledge about the universe: Henry Reich's *MinutePhysics*, Michael Stevens' *Vsauce*, and Brady Haran's *Sixty Symbols* and *Numberphile* channels.

My eternal and deepest thanks go to my family—Mom, Dad, and Victoria—for their unconditional love, encouragement, and support. You guys rock, and I'm so glad to have

you in my life. Thank you Mom and Dad, for making those long drives to take me to college, and thank you, Mom, for making sure I woke up to get to school in the first place back in high school. Thank you, Victoria, for always having my back. I couldn't have done any of this without you all.

Finally, I am endlessly grateful for the friendships that sustained me throughout graduate school. To my friends, both old and new, who offered daily support and companionship—I truly appreciate you. To my graduate student colleagues, thank you for your solidarity through this program. A special thanks to Dino Bektešević for being an exceptional research partner and to Hannah Bish for her wisdom, kindness, and incredible friendship.

It takes a village to raise a scientist. And I am blessed to have been raised well by my family, friends, teachers, and mentors. Thank you all.

DEDICATION

To my parents: Diane and Steve Stetzler.

Chapter 1

INTRODUCTION

1.1 The Big Data Problem

Astronomy has historically been an effort of individuals and small groups studying the cosmos via privately owned and operated telescopes and instruments. An individual investigator formulates a hypothesis, makes a plan to test their hypothesis via one or more observations of the sky, sources funding and resources to execute that plan, and reports what they find. As the questions posed become more ambitious, teams of investigators are required to collaborate to build larger instruments, collect more data, and dig deeper in their analyses in order to advance our understanding of the universe. One result of this evolution is the advent of all-sky telescope surveys in which a team of scientists and engineers design and operate a single telescope to observe the sky in a systematic fashion. These surveys produce archival datasets from which scientific insights are gleaned by collaborations and individual investigators, answering an array of questions simultaneously from one telescope and one survey. The scale of these archival datasets has become truly vast: the largest of which currently comes from the Panoramic Survey Telescope and Rapid Response System (Pan-STARRS; Chambers et al. (2019)), which has produced and released to the Astronomy community a 1.6 petabyte (PB) dataset. Over the next decade, a new telescope and survey—the Vera C. Rubin Observatory's Legacy Survey of Space and Time (LSST; Ivezić et al. (2019))—will produce and publicly release a dataset that will dwarf all others, at a total data volume of 500 PB.

Scientific insights are derived from these datasets through a process of data reduction: an abstract set of input data, typically a digital image of the night sky, is transformed through algorithms expressed as code and executed by computers into a form that has physical meaning. Bits of information (sequential values of ones and zeros) in the digital image of a star are transformed into a measure of the number of photons of a certain energy

that came from it at a particular moment: the star's apparent brightness (Heasley, 1999). The location of the star in the digital image is calculated relative to other stars in the image, providing a location on the celestial sphere, a way to find the star at a later date (Lang et al., 2010). These measurements enter a catalog or database of stars found in the images. Over sequences of these images, the star is found and its brightness measured again and again and entered into the catalog. A query against the catalog allows us to construct a "light curve" for this particular star: how its brightness has varied over time. The physical properties of the star can be derived from this brightness variation, for example through physical equations that model how stars expand and contract. Perhaps this star has a funny shape, and doesn't look like other stars: instead we've found a galaxy, for which we can still measure its brightness and shape. Or perhaps this star isn't found at exactly the same position as before and if we look closely, we notice that it seems to be moving over time: instead we've found an asteroid, an object orbiting the Sun in our own solar system. From digital bits of ones-and-zeros, knowledge about the physical reality of a single star, galaxy, or asteroid is generated. And through the simultaneous analysis of thousands, millions, or even billions of these stars, galaxies, and asteroids we can uncover the mysteries of our universe. From its matter density (Percival et al., 2007), to its expansion rate (Riess et al., 1998), to the properties and history of our own galaxy (Helmi, 2020), to the contents and history of our solar system (Tsiganis et al., 2005; Nesvorný, 2011; Batygin & Brown, 2016).

One of the main challenges in deriving scientific insights from the large datasets produced by all-sky surveys is that of scale. One byte is 8 bits of information, and a petabyte is approximately 1 quadrillion bytes. Existing computers can execute approximately 1 billion instructions per second. If we ask a computer one at a time to turn every zero into a one in the 500 PB dataset LSST would produce, it would take approximately $8 \times 500 \times 2^{50}/10^9 = 4.5 \times 10^9$ seconds or 142 years. The reality is a lot less bleak than this: a modern computer can flip multiple bits in one operation, and can perform multiple instructions at once; however, it takes more than a single operation to convert a digital image into knowledge. In either case, the point remains: the scale of the dataset is vast from the perspective of code execution. The code and algorithms that operate on the images from LSST and their derived catalogs must then be made to execute quickly and efficiently.

1.2 Speeding Up Code

There are four options for speeding up code execution: the code is executed on a faster computer, the code is implemented in a more computationally efficient manner, the algorithm the code implements is made more efficient, or the algorithm/code takes advantage of parallel computing. The most straightforward of these options is the first; however, the rate with which instructions are executed on computers has not surpassed 2-3 billion per second in the past 20 years. For the second, code compilers typically already act to optimize code before it is executed on a computer. Additionally, the gains from this level of optimization can be quite small for the effort required. For the third: algorithmic changes require a lot of human effort and time, but can pay off enormously. If an algorithm's "complexity" can be changed, it can reduce the number of instructions that need to be executed by orders of magnitude. While the payoff can be enormous, many algorithms in use by astronomers are what they are out of necessity: there simply isn't a different algorithm that would suffice. The last solution of utilizing parallel programming is often the most accessible and applicable to the problems of scientific computing.

In the case of converting LSST images into scientifically useful catalogs, parallel computing is directly applicable: the same algorithms can be applied to separate units of the LSST imaging dataset on multiple computers at once. While it might take a single computer 142 years to flip all of the bits in the LSST dataset, it would take 142 computers a year, 1,704 computers a month, and 51,830 computers just a day. When many computers work together, the scale of the big data problem becomes manageable.

The Rubin Observatory has developed the LSST Science Pipelines (LSP; Bosch et al. (2018, 2019)) to solve this problem. The LSP provides a core set of image processing algorithms as well as a robust and scalable data management and task execution system (Jenness et al., 2022). The LSP can be used to scale the image processing effort to thousands of computers with ease.

Most modern computer hardware already supports parallel execution of instructions through multi-core architectures, in which multiple central processing units (CPUs) are available on a single computing chip. Additionally, frameworks exist that provide the ability

to scale codes across multiple machines, or nodes. Hardware based on different computing architectures can unlock another level of parallelism and provide significant acceleration of scientific codes. This hardware most commonly comes in the form of field-programmable gate arrays (FPGAs) and graphics processing units (GPUs). GPUs in particular have seen a recent explosion in their use as general-purpose computing units due to their wide availability and ability to accelerate fundamental operations, such as matrix multiplication, that many scientific algorithms rely on. GPUs are able to execute thousands of instructions at once on a limited pool of data, unlocking an order of magnitude more parallelism than is available from CPUs if utilized efficiently. At present, it is possible through a variety of methods to achieve the computing scale required to address the big data problem. 51,830 computers is not as large a number as it seems.

1.3 Finding Faint Solar System Objects: A Computational Challenge

One of the main goals of the LSST is to inventory the solar system. In addition to the planets, known moons, and the Sun, our solar system contains innumerable small bodies composed of rock and ice tracing bound orbits around the Sun. These small bodies—also known as minor planets—are categorized into many distinct populations of interest depending on their orbital properties. Main Belt Asteroids (MBAs) orbit at ~ 3 AU from the sun, in a belt between Mars and Jupiter. Asteroids with orbits that cross the Earth's are classified as Near Earth Objects (NEOs). Past the planet Neptune, which orbits at 30 AU, lies the Trans-Neptunian Objects (TNOs), a reservoir of icy bodies in the distant solar system.

Solar system bodies are discovered through single- and multi-epoch imaging surveys of the sky. Objects in our solar system move with appreciable proper motion across the sky due to their orbital motion around the sun in addition to parallax induced by the Earth's own orbit. In single epoch-imaging, these objects appear as either point sources or streaks in an image depending on their proper motion at the time of observation and the exposure time of the image. In exposures with ~ 1 minute durations, MBAs and TNOs appear only as point sources. Multi-epoch imaging is required to recover coherent motion in the position of these point sources over time. Detections are collected across a single night to form a

track or "tracklet" of linear motion across the sky, followed by the application of a linking algorithm to group tracklets with coherent motion over nights (Kubica et al., 2007; Denneau et al., 2013; Holman et al., 2018). This process is computationally expensive, with typical linking algorithms scaling with the cube of the number of tracklets—$\mathcal{O}(N^3)$. Advances in algorithmic complexity from Holman et al. (2018) have brought that scaling to $\mathcal{O}(N \log N)$, an enormous and necessary improvement to apply the linking problem at the scale of LSST.

Small solar system bodies are typically cold and emit no light of their own in the visual wavelengths. This means that their brightness in an image depends on the amount of light they reflect from the Sun. This value is proportional to the object's linear size squared, inversely proportional to its heliocentric distance squared, and inversely proportional to its geocentric distance squared; in other words, the brightest objects are both large and are close to us in the solar system while small and distant objects rapidly become too faint to observe. One may consider extending the telescope exposure time to recover faint objects; however, the proper motion of these objects always acts to lessen the signal-to-noise of a point source in an image through streaking/trailing losses making this an infeasible method for the discovery of faint objects. Additionally, non-sidereal tracking of the telescope cannot correct for more than the mean motion of a collection of unknown objects, inevitably leading to reduced signal-to-noise of any detected objects.

Fortunately, computational methods already exist to find faint solar system objects in multi-epoch imaging. These faint moving objects can be recovered through the application of the shift-and-stack algorithm—also known as digital tracking/synthetic tracking/track-before-detect—to survey images. Shift-and-stack aligns pixels (shift) from a set of multi-epoch images along a candidate moving object trajectory before coadding them (stack) (Gladman et al., 1998; Bernstein et al., 2004; Heinze et al., 2015). Trajectories that align with real moving objects lead to coherent signal accumulation across epochs, enabling a high-confidence detection even when the object appears at low-SNR in individual images. In stacks of 100 images, this algorithm can discover objects 10× (2.5 magnitudes) fainter than would otherwise be possible.

The main barrier to broad application of shift-and-stack is its enormous computational expense: the number of pixel stacks required for a complete search can easily exceed 10^{10} for

small scale searches and take hours or days of compute time. This computational expense scales with the velocity of the objects targeted for search, limiting the application of shift-and-stack mostly to searches for slow moving objects in the outer solar system. Speedup can be achieved through improved parallelism, for example application of shift-and-stack on GPUs. Whidden et al. (2019) showed it is possible to search the necessary 10^{10} trajectories in under a minute with a single GPU. With the use of high levels of parallelism, this extremely computationally intensive problem can be tackled.

1.4 Realizing the Potential of LSST

The scientific potential of LSST can only be realized when tools and algorithms are available to scale to the size of its dataset. This can be achieved by improving existing algorithms and exploiting parallel execution to provide speedup in their application.

In Chapter 2 of this thesis, I address the challenge of accessing and analyzing the PB-sized LSST catalog. I demonstrate how a cloud-based science platform can enable next-to-the-data analyses of the LSST catalog among other terabyte-scale astronomical tabular datasets. The presented platform is built on Amazon Web Services, utilizes Apache Spark and the Astronomy eXtensions for Spark for parallel data analysis and manipulation, and provides a JupyterHub web-accessible front-end for user access (Zaharia et al., 2010; Zečević et al., 2019). I demonstrate the usability of this platform through an example science analysis of data from the Zwicky Transient Facility's 1 billion+ light-curve catalog. I show how this system enables an end-user to iteratively build analyses (in Python) that transparently scale processing with no need for end-user interaction. This kind of system will be necessary for investigators to apply their scientific workflow to the LSST dataset.

In Chapter 3 of this thesis, I describe an effort to perform large-scale image processing for a TNO discovery survey using the LSST Science Pipelines. I describe a set of tools for performing the processing in an automated and reproducible manner. I additionally perform experiments to optimize the image processing for the discovery of slow-moving solar system objects. I show it is possible to recover ~ 0.25 mag of lost flux in bright TNOs through the construction of an optimized survey template. This work verifies the scalability of the LSST Science Pipelines and provides tools that users and research groups can use to perform their

own experiments and processing campaigns on the LSST imaging dataset.

In Chapter 4 of this thesis, I present an alternative implementation of the shift-and-stack algorithm that trades off sensitivity to the faintest objects for improved computational performance. My algorithm applies the shift-and-stack algorithm to low-SNR detection catalogs derived from single-epoch imaging. "Stacking" detection catalogs instead of pixels reduces the number of numerical operations performed, and the sparsity of the detection catalog enables key approximations that reduce the number of candidate trajectories that need to be searched. Together, these enable real-world speedups of $10 - 10^3\times$ over image-based shift-and-stack while retaining the ability to find objects fainter than would be possible otherwise. I explore the depth-compute time trade-off of this algorithm and find that, when searching for Trans-Neptunian Objects, my algorithm can achieve wall-clock runtime speedups of at least $\sim 30\times$ with 88% of the memory usage while sacrificing 0.25 mag in depth in comparison with an image-based shift-and-stack approach. The attained speedups enable the broad application of shift-and-stack to the LSST imaging dataset. It additionally provides a path forward for performing shift-and-stack searches to find faster moving inner solar system objects.

Chapter 2

ENABLING ACCESS TO SCALABLE COMPUTING IN THE CLOUD

2.1 Introduction

Today's astronomy is undergoing a major change. Historically a data-starved science, it is being rapidly transformed by the advent of large, automated, digital sky surveys into a field where terabyte and petabyte data sets are routinely collected and made available to researchers across the globe.

The Zwicky Transient Facility (ZTF; Bellm et al. (2019); Graham et al. (2019); Dekany et al. (2020); Masci et al. (2019)) has engaged in a three-year mission to monitor the Northern sky. With a large camera mounted on the Samuel Oschin 48-inch Schmidt telescope at Palomar Observatory, the ZTF is able to monitor the entire visible sky almost twice a night. Generating about 30 GB of nightly imaging, ZTF detects up to 1,000,000 variable, transient, or moving sources (or alerts) every night, making them available to the astronomical community (Patterson et al., 2018). Towards the middle of 2025, a new survey, the Legacy Survey of Space and Time (LSST; Ivezić et al., 2019), will start operations on the NSF Vera C. Rubin Observatory. Rubin Observatory's telescope has a mirror almost seven times larger than that of the ZTF, which will enable it to search for fainter and more distant sources. Situated in northern Chile, the LSST will survey the southern sky taking $\sim 1,000$ images per night with a 3.2 billion-pixel camera with a $\sim$10 deg^2 field of view. The stream of imaging data ($\sim$6PB/yr) collected by the LSST will yield repeated measurements ($\sim$100/yr) of over 37 billion objects, for a total of over 30 trillion measurements by the end of the next decade. These are just two examples, with many others at similar scale either in progress (Kepler, Pan-STARRS, DES, GAIA, ATLAS, ASAS-SN; Kaiser et al., 2010; Dark Energy Survey Collaboration et al., 2016; Gaia Collaboration et al., 2016a; Tonry et al., 2018; Shappee et al., 2014) or planned (Roman, Euclid; Spergel et al., 2015; Scaramella et al., 2014). They are being complemented by numerous smaller projects ($\lesssim$\$1M scale),

contributing billions of more specialized measurements.

This 10-100x increase in survey data output has not been followed by commensurate improvements in tools and platforms available to astronomers to manage and analyze those catalogs. Most survey-based studies today are performed by navigating to archive websites, entering (very selective) filtering criteria to download "small" (∼10s of millions of rows; ∼10GB) subsets of catalog products. Those subsets are then stored locally and analyzed using custom routines written in high-level languages (e.g., Python or IDL), with the algorithms generally assuming in-memory operation. With the increase in data volumes and subsets of interest growing towards the ∼100GB-1TB range, this mode of analysis is becoming infeasible.

One solution is to provide astronomers with access to the data through web portals and *science platforms* – rich gateways exposing server-side code editing, management, execution and result visualization capabilities – usually implemented as notebooks such as Jupyter (Kluyver et al., 2016) or Zeppelin (Cheng et al., 2018). These systems are said to *bring the code to the data*, by enabling computation on computational resources co-located with the data and providing built-in tools to ease the process of analysis. For example, the Rubin Observatory/LSST has designed (Jurić et al., 2017; Dubois-Felsmann et al., 2017) and implemented a science platform suitable for accessing and visualizing data from the LSST, with deployments hosted on both on-premises hardware and Google Cloud (O'Mullane et al., 2021).[1] While such science platforms are a major step forward in working with large datasets, they still have some limitations when deployed on on-premises hardware or traditional HPC systems. These systems can suffer from having insufficient computing next to the data: all users of shared HPC resources are familiar with "waiting in the queue" due to over subscription. Science platforms built on cloud computing resources will find it easier to provide computing resources according to user demand: this is the promise of "elastic" computing in the cloud.

Secondly, even when surveys deploy distributed SQL databases for serving user queries (e.g. Qserv in the case of LSST; Wang et al., 2011), user analysis is still not easily parallelized

[1] See as well https://data.lsst.cloud/

– query requests and results are bottlenecked at one access point which severely limits scalability. In contrast, the system we describe and implement provides direct, distributed access to data for a user's analysis code. Finally, current science platforms do not tackle the issue of working on multiple large datasets at the same time – if they're in different archives, they still have to be staged to the same place before work can be done. In other words, they continue to suffer from availability of computing, being I/O-bound, and geographic dislocation.

We therefore need to not only bring the code to the data, but also *bring the data together*, co-locate it next to an (ideally limitless) reservoir of computing capacity, with I/O capabilities that can scale accordingly. Furthermore, we need to make this system *usable*, by providing astronomer-friendly frameworks for working with extremely large datasets in a scalable fashion. Finally, we need to provide a user interface which is accessible and familiar, with a shallow learning curve.

We address the first of these challenges by utilizing the Cloud to supply data storage capacity, effective dataset co-location, I/O bandwidth, and (elastic) compute capability. This work utilizes the Amazon Web Services (AWS) cloud, leveraging Amazon Simple Storage Service (Amazon S3) for storage and access to TB+ sized tabular data sets (catalogs) and Amazon Elastic Cloud Compute (Amazon EC2) for elastic computing. Bektesevic et al. (2020) have investigated using the same services for scalable storage, access, and processing of image data. The second challenge is addressed by extending the Astronomy eXtensions for Spark (AXS; Zečević et al., 2019), a distributed database and map-reduce like workflow system built on the industry-standard Apache Spark (Zaharia et al., 2010) engine, to work in this cloud environment. Spark allows the execution of everything from simple ANSI SQL-2011 compliant queries to complex distributed workflows, all driven from Python. When using Spark, data can be sourced from a number of storage solutions and a variety of formats, including FITS (Peloton et al., 2018). Finally, a JupyterHub facade provides a user-friendly entry-point to the system. Additionally, we make it possible for IT groups (or advanced users) to easily deploy this entire system for use within their departments, as an out-of-the-box solution for cloud-based astronomical data analysis.

The combination of these technologies allows the researcher to migrate "classic" subset-

download-analyze workflows with little to no learning curve, while providing an upgrade path towards large-scale analysis. We validate the approach by deploying a cloud-based platform for accessing and analyzing a 1 billion+ light-curve catalog from the ZTF (a precursor to LSST), and demonstrate it can be successfully used for exploratory science.

2.2 A Platform for User-Friendly Scalable Analysis of Large Astronomical Datasets

We begin by introducing the properties of cloud systems that make them especially suitable for scalable astronomical analysis platforms, discuss the overall architecture of our platform, its individual components, and performance.

2.2.1 The Cloud

Traditionally, computing infrastructure was acquired and maintained close to the group utilizing the resource. For example, a group led by a faculty member would purchase and set up one or more machines for a particular problem, or (on a larger scale) a university may centralize computing resources into a common cluster, shared with the larger campus community. These acquisitions – so-called "on-premise" computing – are capital heavy (require a large initial investment), require local IT knowledge, and allow for a limited variety of the systems being purchased (e.g., a generic Linux machine for a small group, or standardized types of nodes for an HPC cluster).

Cloud services move this infrastructure (and the work to maintain it) away from the user, and centralize it with the cloud provider. The infrastructure is provided as a service: individual machines, entire HPC clusters, as well as higher-order services (databases, filesystems, etc.) are *rented* for the time the resource is needed, rather than purchased.

They are billed proportional to usage; virtual machines are typically rented by the second, virtual networks priced by bandwidth usage, and virtual storage priced by storage size per unit time. These components are provisioned by the user on-demand, and are built to be "elastic." One can typically rent several hundred virtual machines and provision terabytes of storage space with an expectation that it will be delivered within minutes and then release these resource back to the cloud provider at will. This usage and pricing model

offers the unique benefit of providing access to affordable computing at scale. One can rent hundreds of virtual machines for a short period of time (just the execution time of a science workflow) without investing in the long-term support of the underlying infrastructure. In addition, cloud providers typically offer managed storage solutions to support reading/writing data to/from all of these machines. These so-called "object stores" are highly available, highly durable, and highly scalable stores of arbitrarily large data volumes. For example, Amazon Simple Storage Service (Amazon S3) provides scalable, simultaneous access to data through an Application Programming Interface (API) over a network.[2] S3 supports very high throughput at the terabit-per-second level, assuming storage access patterns are optimized.[3] Once a solution for scalable storage is added to the mix, cloud computing systems start to resemble the traditional supercomputers many scientists are already familiar with for running simulations and performing large-scale data analysis.

2.2.2 Orchestrating cloud applications: Kubernetes

The pain point that remains in managing and developing applications for the cloud is the problem of orchestration: it can become burdensome to write custom software for provisioning and managing cloud resources, and there is a danger of cloud "lock-in" occurring when software applications become too strongly coupled with the cloud provider's API. The open source community has developed orchestration tools, like Kubernetes, to address this issue.[4]

Kubernetes is used to schedule software applications packaged in Docker images and run as Docker containers on a cluster of computers (physical or virtual machines) while handling requests for, and provisioning, cloud resources to support running those containers.[5] Kubernetes provides a cloud-agnostic API, accessible over a network using HTTP(S), to describe

[2]Amazon S3 uses a REST API with HTTP.

[3]This is detailed in the S3 documentation: https://docs.aws.amazon.com/AmazonS3/latest/dev/optimizing-performance.html

[4]The Kubernetes documentation provides a thorough and beginner-friendly introduction to the software: https://kubernetes.io/docs/

[5]Docker isolates software programs at the level of the operating system, in contrast to virtual machines which isolate operating systems from one another at the hardware level. See https://www.docker.com/ and https://docs.docker.com/ for more information.

cloud resources as "representational state transfer" (REST) objects. For example, cluster storage is described using "Persistent Volume" objects, requests for that storage use "Persistent Volume Claim" objects, and networking utilities like routing, port-forwarding, and load balancing use "Service" objects. An application that runs using one or more containers is specified using a "Pod" object. If the application requires it, the Pod object can reference storage objects and service objects by name to link an application to these resources. In addition, the Pod object allows one to impose CPU and memory limits on an application or assign the application to a certain node, among other features. Each Kubernetes object is described using YAML, a human-readable format for storing configuration information (lists and dictionaries of strings and numbers).[6] Figure 2.1 shows an example set of YAML-formatted text describing Kubernetes objects that together would link a Jupyter notebook server backed by a 10 GiB storage device to an internet-accessible URL.

The Kubernetes core service, a set of software called the control plane, is responsible for maintaining an API server accessible within (and potential externally to) the cluster, maintaining a database of the objects created so far, assigning pods (applications) to nodes in the cluster in a way that respects their constraints, keeping track of the general state of the cluster, and handling aspects of networking within the cluster and through the cloud provider. Additional components of the Kubernetes core code (or third-party plugins) handle provisioning of virtual hardware from the cloud provider to satisfy requirements that cannot be met by current cluster resources. As an example on AWS, an outstanding request for a Service requiring a load balancer will be fulfilled by creating an AWS Elastic Load Balancer (ELB) or Application Load Balancer (ALB). Similarly, an outstanding request for a Persistent Volume will be fulfilled by creating an Amazon Elastic Block Store (EBS) volume. Finally, applications can be scheduled on the cluster that modify the cluster state. In particular, the Kubernetes Cluster Autoscaler interacts with the cloud provider to terminate underutilized nodes or add new nodes when there are pods that cannot be scheduled given the current number of nodes.[7] The handling of hardware provisioning from the cloud

[6] See https://yaml.org/ for specification and implementations.

[7] https://github.com/kubernetes/autoscaler

Network

```
apiVersion: v1
kind: Service
metadata:
  name: my-load-balancer
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 8888
    protocol: TCP
    name: http
  selector:
    app: notebook-pod
```

Storage

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-volume-claim
spec:
  resources:
    requests:
      storage: 10Gi
      storageClassName: ebs
```

Application

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jupyter-notebook
spec:
  replicas: 1
  selector:
    matchLabels:
      app: notebook-pod
  template:
    metadata:
      labels:
        app: notebook-pod
    spec:
      containers:
      - name: notebook
        image: jupyter/scipy-notebook
        ports:
        - containerPort: 8888
        volumeMounts:
        - mountPath: "/home/jovyan"
          name: my-volume
      volumes:
      - name: my-volume
        persistentVolumeClaim:
          claimName: my-volume-claim
```

Figure 2.1: An illustration of the structure and composition of YAML-formatted text specifying Kubernetes objects that together create a functional and internet-accessible Jupyter notebook server. The Jupyter notebook application is created as a Pod on the cluster (right). Networking objects (top left) specify how a public-facing load balancer can be connected to the Jupyter notebook Pod (`notebook-pod`) on a certain port (`8888`). Storage objects trigger the creation of, for example, hard drive disk space from the cloud provider (bottom left). Colored text indicate how the files are linked to support one another: blue indicates how network and application are linked, orange how application and storage are linked, and green how storage volumes are mounted into the filesystem of the application.

provider by administrative software in the Kubernetes control plane, through Kubernetes plugins, and through applications running in the cluster allows additional user applications to remain decoupled from the cloud provider's API.

2.2.3 System Architecture

Cloud systems offer unique infrastructure elements that help support a system for scalable science analysis. Virtual machines can be rented in the hundreds or thousands to support large computations, each accessing data in a scalable manner from a managed service. Orchestration layers, like Kubernetes, ease the process of running science software on cloud resources. In this section, we discuss how we leverage cloud infrastructure to build such a platform. Underlying this platform are four key components:

1. An interface for computing. We use the Jupyter ecosystem: a JupyterHub deployment based on the `zero-to-jupyterhub` project that creates Jupyter notebook servers on our computing infrastructure for authenticated users. A Jupyter notebook server provides a web interface to interactively run code on a remote machine alongside a set of pre-installed software libraries.[8]

2. A scalable analytics engine. We use Apache Spark, an industry standard tool for distributed data querying and analysis, and the Astronomy eXtensions to Spark (AXS).

3. A scalable storage solution. We use Amazon Simple Storage Solution (S3). Amazon S3 is a managed object store that can store arbitrarily large data volumes and scale to an arbitrarily large number of requests for this data.

4. A deployment solution. We've developed a set of Helm charts and bash scripts automating the deployment of this system onto the AWS cloud.[9]

---

[8]See https://zero-to-jupyterhub.readthedocs.io/ and https://github.com/jupyterhub/zero-to-jupyterhub-k8s.

[9]For Helm, see https://helm.sh/.

Each of these components are largely disconnected from one another and can be mixed and matched with other drop-in solutions.[10] Aside from the deployment solution, each of these components are comprised of simple processes communicating with each other through an API over a network. This means that each solution for (1), (2), and (3) is largely agnostic to the choice of running on a bare-metal machine, inside a virtual machine (VM), inside a Linux container, or using a managed cloud service as long as each component is properly networked.

Figure 2.2 shows the state of the Kubernetes cluster during normal usage of a platform created with our Helm chart as well as the pathway of API interactions that occur as a user interacts with the system. A user gains access to the system through a JupyterHub, which is a log-in portal and proxy to one or more managed Jupyter notebook servers spawned by the JupyterHub. This notebook server is run on a node of the Kubernetes cluster, which can be constrained by hardware requirements and/or administrator provided node labels. A proxy forwards external authenticated requests from the internet to a user's notebook server. Users can use the Apache Spark software, which is pre-installed on their server, to create a Spark cluster using the Spark on Kubernetes API. The user can also access their running notebook server using a Secure Shell (SSH) client.

An Interface to Computing

The Jupyter notebook application, and its extension Jupyter lab, provide an ideal environment for astronomers to access, manipulate, and visualize data sets. The Jupyter notebook/lab applications, although usually run locally on a user's machine, can run on a remote machine and be accessed through a JupyterHub, a web application that securely forwards authenticated requests directed at a central URL to a running notebook server.[11]

---

[10] Zepplin notebooks, among other tools, compete with Jupyter notebooks for accessing remote computers for analysis and data visualization. Dask is a competing drop-in for Apache Spark that scales Python code natively. A Lustre filesystem could be a drop-in for Amazon S3. Amazon EFS, a managed and scalable network filesystem, is also an option. Kustomize is an alternative to Helm.

[11] As an example, one may access a JupyterHub at the URL `https://hub.example.com` which, if you are an authenticated user, will forward through a proxy to `https://hub.example.com/user/username`. When running a notebook on a local machine, there is no access to a JupyterHub and the single user server is served at (typically) `http://localhost:8888`.

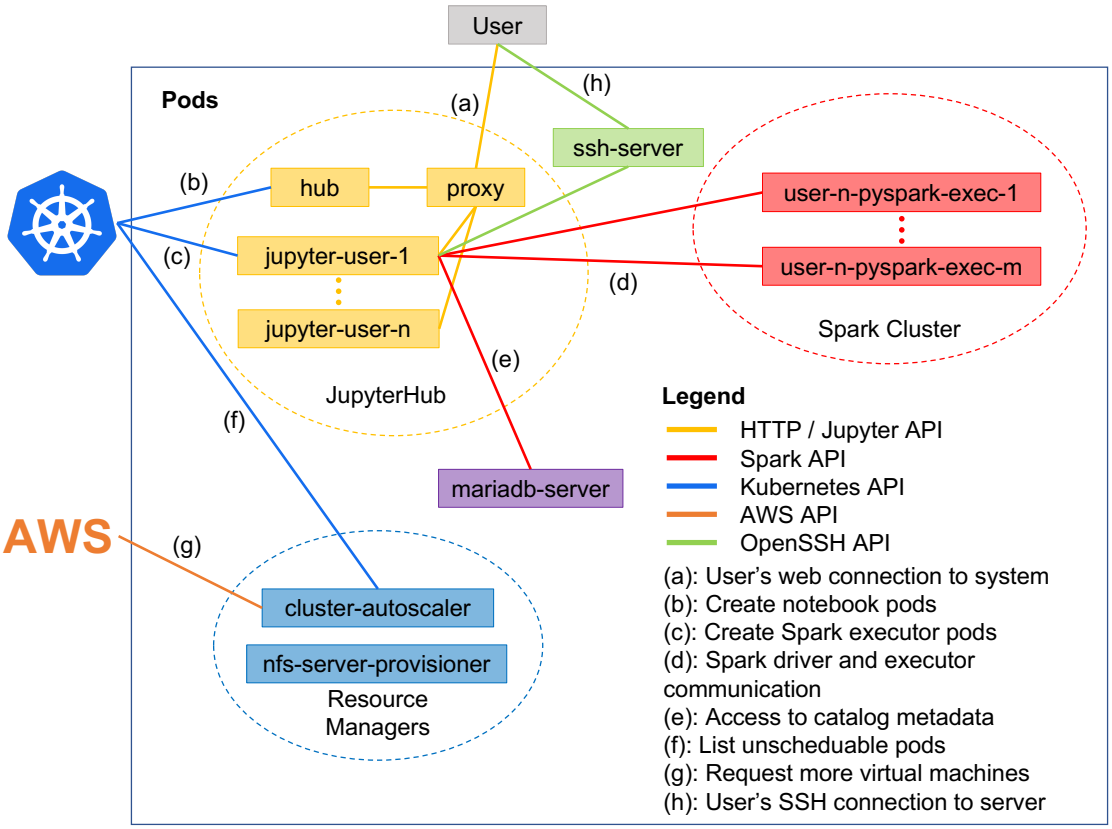


Figure 2.2: A diagram of the essential components of the Kubernetes cluster when the science platform is in use. Each box represents a single Kubernetes Pod scheduled on the cluster. The colors of the boxes and the dashed ovals surrounding the three groups are for visualization purposes only; each Pod exists as an independent entity to be scheduled on any available machines. The colored paths and letter markers indicate the pattern of API interactions that occur when users interact with the system. (a) shows a user connecting to the JupyterHub from the internet. The JupyterHub creates a notebook server (`jupyter-user-1`) for the user (b). The user creates a Spark cluster using their notebook server as the location for the Spark driver process (c). Scheduled Spark executor Pods connect back to the Spark driver process running in the notebook server (d). The Spark driver process accesses a MariaDB server for catalog metadata (e). In the background, the Kubernetes cluster autoscaler keeps track of the scheduling status of all Pods (f). At any point in (a)-(d), if a Pod cannot be scheduled due to a lack of cluster resources, the cluster autoscaler will request more machines from AWS to meet that need (g). Optionally, the user can connect to their running server with SSH (h).

The authentication layer of JupyterHub allows us to block non-authenticated users from the platform. Our science platform integrates authentication through GitHub, allowing us to authenticate both individual users by their GitHub usernames and groups of users through GitHub Organization membership. For example, the implementation of this science platform described in Section 2.3 restricts access to the platform and its private data to members of the `dirac-institute` and `ZwickyTransientFacility` GitHub organizations.

Users can choose to bypass the Jupyter computing environment by accessing their running notebook server with a SSH client. The SSH client also facilitates file transfers between the user and their notebook server when using utilities such as `scp` or `rsync`. Access through SSH is implemented using a "jump host" setup: a single, always-running container on the cluster runs an OpenSSH server that is networked to the internet. When the user's notebook server is running, it additionally runs an OpenSSH server in the background. The user adds a cryptographic public key to a filesystem shared between the notebook server and jump host. The user connects to the jump host with their username and a cryptographic private key stored on their local machine. From the jump host, the user can connect to their running notebook server with no additional configuration. A properly formatted invocation of the `ssh` command can do this in one step. Additional public and private keys are generated automatically for each host and each user and placed on a shared filesystem with correct permissions.

Finally, a Virtual Network Computing (VNC) desktop is made available to the user for using graphical applications outside of the Jupyter notebook. The VNC desktop is provided through the Jupyter Remote Desktop Proxy software, an extension to the Jupyter notebook server. The in-browser desktop emulation offers reasonable interaction latencies over a typical internet connection.

A Scalable Analytics Engine

Apache Spark (Spark) is a tool for general distributed computing, with a focus on querying and transforming large amounts of data, that works well in a shared-nothing, distributed computing environment. Spark uses a driver/executor model for executing queries. The

driver process splits a given query into several (1 to thousands) independent tasks which are distributed to independent executor processes. The driver process keeps track of the state of the query, maintains communication with its executors, and coalesces the results of finished tasks. Since the driver and executor(s) only need to communicate with each other over the network, executor processes can remain on the same machine as a driver, to take advantage of parallelism on a single machine, or be distributed across several other machines in a distributed computing context.[12] The API for data transformation, queries, and analysis remains the same whether or not the Spark engine executes the code sequentially on a local machine or in parallel on distributed machines, allowing code that works on a laptop to naturally scale to a cluster of computers.

To support astronomy-specific operations, Zečević et al. (2019) have developed the Astronomy eXtensions to Spark (AXS), a set of additional Python bindings to the Spark API to ease astronomy-specific data queries such as cross matches and sky maps in addition to an internal optimization for speeding up catalog cross matches using the ZONES algorithm, described in Zečević et al. (2019). AXS is included in our science platform to ease the use of Spark for astronomers and also provide fast cross-matching capability between catalogs.

AXS requires that tabular data is stored Apache Parquet file format, a compressed column-oriented data storage format.[13] The columnar nature and partitioning of the files in Parquet format allows for very fast reads of large tables. For example, one can obtain a subset of just the "RA" column of a catalog without scanning through all parts of all of the files. Apache Spark's flexible functionality for accessing data of different formats, exposed in Python through `pyspark.sql.DataFrame.read`, allows one to convert a broad range of catalogs in different formats – including FITS (Peloton et al., 2018) – to Parquet. AXS additionally requires that catalogs stored in Parquet be similarly partitioned in order to perform fast cross-matches. AXS provides a single function, exposed in Python as `AxsCatalog.save`, that will re-partition a data frame read using Spark, save it in Parquet format, and make the table available to a user through its Apache Hive metastore

[12] Creating executor processes on a single machine isn't done in practice; instead, Spark supports multi-threading in the driver process that replace the external executor process(es) when using local resources.

[13] See https://parquet.apache.org/

database.[14]

A Scalable Storage Solution

Amazon S3 is a scalable object store with built-in backups and optional replication across geographically distinct AWS regions. Files are placed into a S3 bucket, a flat filesystem that scales well to simultaneous access from thousands of individual clients. Files are accessed over the network using a REST API over HTTP, supporting actions to retrieve and create new objects in the bucket. The semantics of the S3 API are not compliant with the POSIX specification that typical filesystems adhere to. However, there are projects, such as `s3fs`, that allow for mounting of the S3 object store as a traditional filesystem and provide an interface layer that makes the filesystem largely POSIX compliant.[15] The names of S3 buckets are globally unique, which makes public and private sharing of data in a bucket easy: a user anywhere in the world can access public data from an S3 bucket by specifying only its name. To access private data, the user must additionally authenticate them self with AWS. Access control lists provide object-level permissions for read/write access to certain users and the public. Additionally, there is no limit to the amount of data that can be stored, although individual files must be no larger than 5 TB, and individual upload actions cannot exceed 5 GB. In this platform, we store and access TB+ tabular datasets stored in Parquet format with a common partitioning scheme, making the data AXS compatible.

A deployment solution

We have created a deployment solution for organized creation and management of each of these three components. The code for this is stored at a GitHub repository accessible at https://github.com/astronomy-commons/science-platform. Files referenced in the following code snippets assume access at the root level of this repository.

To create and manage our Kubernetes cluster, we use the `eksctl` software.[16] This

[14] See https://hive.apache.org/

[15] See https://github.com/s3fs-fuse/s3fs-fuse

[16] See https://eksctl.io/

software defines configuration of the Amazon Elastic Kubernetes Service (EKS) from YAML-formatted files. An EKS cluster consists of a managed Kubernetes master node that runs the control plane software along with a set of either managed or unmanaged nodegroups backed by Amazon Elastic Compute Cloud (EC2) virtual machines which the applications scheduled on the cluster.[17]

To help us manage large numbers of Kubernetes objects, we use Helm, the "package manager for Kubernetes." Helm allows Kubernetes objects described as YAML files to be templated using a small number of parameters or "values," also stored in YAML. Helm packages together YAML template files and their default template values in Helm "charts." Helm charts can have versioned dependencies on other Helm charts to compose larger charts from smaller ones. After cluster creation, we use Helm to install the `cluster-autoscaler-chart`, which deploys the Kubernetes Cluster Autoscaler application. The cluster autoscaler scales the number of nodes in the Kubernetes cluster up or down when resources are too constrained or underutilized.

We have created a Helm chart to manage and distribute versioned deployments of our platform. This chart depends on three sub-charts:

1. The `zero-to-jupyterhub` chart, a standard and customizable installation of JupyterHub on Kubernetes. The `zero-to-jupyterhub` chart uses Docker images from the Jupyter Docker Stacks by default and uses the `KubeSpawner` for creating Jupyter notebook servers using the Kubernetes API directly.[18]

2. The `nfs-ganesha-server-and-external-provisioner` chart, which provides a network filesystem server and Kubernetes-compliant storage provisioner.[19]

3. A `mariadb` chart, which provides a MariaDB server and is used as an Apache Hive

[17]Managed nodes are EC2 virtual machines with a tighter coupling to an EKS cluster. Unmanaged nodes allow for more configuration by an administrator.

[18]See https://jupyter-docker-stacks.readthedocs.io/ and https://jupyterhub-kubespawner.readthedocs.io/

[19]See https://github.com/kubernetes-sigs/nfs-ganesha-server-and-external-provisioner

metadata store for AXS.[20]

The Helm chart contains configuration of the three sub-charts. For example, the chart is configured to use a Docker image with installations of Spark/AXS, the OpenSSH client/server, and Jupyter notebook server extensions like the Jupyter Remote Desktop Proxy when the JupyterHub starts a notebook server. Additional configuration in the chart provides instructions for mounting the network filesystem, instructions for setting up the Hive metastore, defines reasonable defaults for using Spark on Kubernetes, and defines notebook server startup-scripts that start the SSH server and set up the user's space on the filesystem (such as copying example notebooks to the user's home directory).

2.2.4 Providing a shared filesystem with granular access control

We found it to be critically important to provide a way for users to easily share files with one another. The default Helm chart and `KubeSpawner` configuration creates a Persistent Volume Claim backed by the default storage device configured for the Kubernetes cluster for each single user server, allowing a user's files to persist beyond the lifetime of their server. For AWS, the default storage device is an EBS volume, roughly equivalent to a network-connected SSD with guaranteed input/output capabilities. By default, this volume is mounted at the filesystem location `/home/jovyan` in the single user container. This setup makes it difficult for the users' results to be shared with others because: a) they are isolated to their own disk, and b) by default all users share the same username and IDs, making granular access control extremely difficult.

To resolve these issues, we provisioned a network filesystem (NFSv4) server using the `nfs-ganesha-server-and-external-provisioner` Helm chart, creating a centralized location for user files and enabling file sharing between users. To solve the problem of access control, each notebook container is started with two environment variables: `NB_USER` set equal to the user's GitHub username, and `NB_UID` set equal to the user's GitHub user id. The start-up scripts included in the default Jupyter notebook Docker image use the values of these environment variables to create a new Linux user, move the home directory loca-

[20] See https://mariadb.org/ and https://github.com/bitnami/charts/tree/master/bitnami/mariadb

tion, update home directory ownership, and update home directory permissions from their default values. Figure 2.3 shows how the NFS server is mounted into single user pods to enable file sharing. The NFS server is mounted at the `/home` directory on the single user server, and a directory is created for the user at the location `/home/<username>`. Each user's directory is protected using UNIX-level file permissions that prevent other users from making unauthorized edits to their files. System administrators can elevate their own permissions (and access the back-end infrastructure arbitrarily) to edit user files at will. The UNIX user ids (UIDs) are globally unique, since they are equal to a unique GitHub ID.

In initial experiments, we used the managed AWS Elastic filesystem (EFS) service to enable file sharing. Using the managed service provides significant benefits, including unlimited storage, scalable access, and automatic back-ups. However, EFS had a noticeable latency increase per Input/Ouput operation compared to the EBS-backed storage of the Kubernetes-managed NFS server. In addition, EFS storage is 3× more expensive than EBS storage.[21]

In addition to storing home directories on the NFS server, we have an option to store all of the science analysis code (typically managed as `conda` environments) on the NFS server. This has several advantages relative to the common practice of keeping the code in Jupyter notebook Docker images. The primary advantage is that this allows for updating of installed software in real-time, and without the need to re-start user servers. A secondary advantage is that the Docker images become smaller and faster to download and start up (thus improving the user experience). The downside is decreased scalability: the NFS server includes a central point, shared by all users of the system. Analysis codes are often made up of thousands of small files, and a request for each file when starting a notebook can lead to large loads on the NFS server. This load increases when serving more than one client, and may not be a scalable beyond serving a few hundred users.

For systems requiring significant scalability, a hybrid approach of providing a base `conda` environment in the Docker image itself in addition to mounting user-created and user-

[21]The cost of EFS is $0.30/GB-Month vs $0.10/GB-Month for EBS. Lifecycle management policies for EFS that move infrequently used data to a higher-latency access tier can reduce costs to approximately the EBS level.

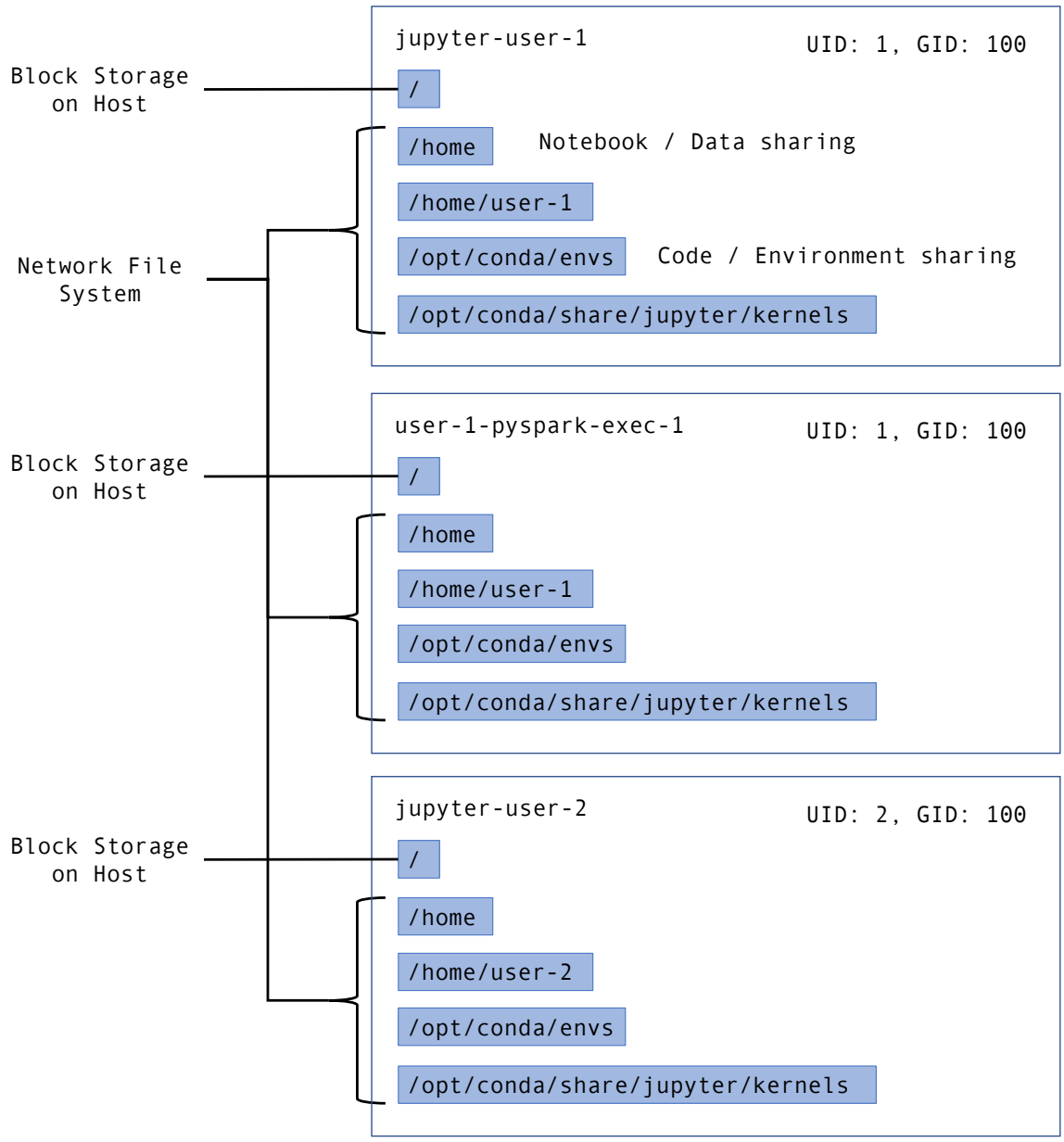


Figure 2.3: An illustration of the filesystem within each container spawned by the JupyterHub (`jupyter-user-1` and `jupyter-user-2`) and by the user in the creation of a distributed Spark cluster. Most of the filesystem (the root directory `/`) exists on an ephemeral storage device tied to the host machine. The home directories, `conda` environment directories, and Jupyter kernel directories within each container are mounted from an external NFS server. This file structure allows for sharing of Jupyter Notebook files and code environments with other users and with a user's individual Spark Cluster. UNIX user ids (`UID`) and group ids (`GID`) are set to prevent unauthorized data access and edits.

managed `conda` environments and Jupyter kernels from the NFS server is warranted. This allows for fast and scalable access to the base environment while also providing the benefit of shared code bases that can be updated in-place by individual users.

2.2.5 Providing Optimal and Specialized Resources

Some users require additional flexibility in the hardware available to match their computing needs. To accommodate this, we have made deployments of this system that allow users to run their notebooks on machines with more CPU or RAM or with specialty hardware like Graphics Processing Units (GPUs) as they require. This functionality is restricted to deployments where we trust the discretion of the users and is not included in the demonstration deployment accompanying this manuscript.

Flexibility in hardware is provided through a custom JupyterHub options form that is shown to the user when they try to start their server. An example form is shown in Fig. 2.4. Several categories of AWS EC2 instances are enumerated with their hardware and costs listed. Hardware is provisioned in terms of vCPU, or "virtual CPU," roughly equivalent to one thread on a hyperthreaded CPU. In this example, users can pick an instance that has as few resources as 2 vCPU and 1 GiB of memory at the lowest cost of $0.01/hour (the `t3.micro` EC2 instance), to a large-memory machine with 96 vCPU and 768 GiB of memory at a much larger cost of $6.05/hour (the `r5.24xlarge` EC2 instance). In addition, nodes with GPU hardware are provided as an option at moderate cost (4 vCPU, 16 GiB memory, 1 NVIDIA Tesla P4 GPU at $0.53/hour; the `g4dn.xlarge` EC2 instance). These GPU nodes can be used to accelerate code in certain applications such as image processing and machine learning. For this deployment, the form is configured to default to a modest choice with 4 vCPU and 16 GiB of memory at a cost of $0.17/hour (the `t3.xlarge` EC2 instance). This range of hardware options and prices will change over time; the list provided is simply an example of the on-demand heterogeneity provided via AWS.

Server Options

Customize...

Compute optimized

C5

	Size	CPU	Memory	Price	Network	Extra Hardware
○	large	2	4 GiB	$0.09/hour	Up to 10 Gigabit	
○	xlarge	4	8 GiB	$0.17/hour	Up to 10 Gigabit	
○	2xlarge	8	16 GiB	$0.34/hour	Up to 10 Gigabit	
○	4xlarge	16	32 GiB	$0.68/hour	Up to 10 Gigabit	
○	12xlarge	48	96 GiB	$2.04/hour	12 Gigabit	
○	24xlarge	96	192 GiB	$4.08/hour	25 Gigabit	

GPU instance

G4DN P3

	Size	CPU	Memory	Price	Network	Extra Hardware
○	xlarge	4	16 GiB	$0.53/hour	Up to 25 Gigabit	1 GPUs and 125 GB NVMe SSD
○	2xlarge	8	32 GiB	$0.75/hour	Up to 25 Gigabit	1 GPUs and 225 GB NVMe SSD
○	4xlarge	16	64 GiB	$1.20/hour	Up to 25 Gigabit	1 GPUs and 225 GB NVMe SSD

Figure 2.4: A screenshot of the JupyterHub server spawn page. Several options for computing hardware are presented to the user with their hardware and costs enumerated. Of note is the ability to spawn GPU instances on demand. When a user selects one of these options, their spawned Kubernetes Pod is tagged so that it can only be scheduled on a node with the desired hardware. If a node with the required hardware does not exist in the Kubernetes cluster, the cluster autoscaler will provision it from the cloud provider (introducing a $\sim$5 minute spawn time).

2.2.6 Multi-cloud support

It is unlikely, and perhaps undesirable, that all scientists and organization will agree to use a single cloud provider when storing data, accessing computing resources, or deploying our system. There are many clouds outside of Amazon Web Services that scientists may have already chosen for computing and data storage based on factors such as the availability of compute credits, academic institutional tie-ins, convenience, familiarity, or differences in product offerings.[22] Therefore, it is necessary to think about and accommodate multi-cloud support in our architecture. The system architecture outlined in 2.2.3 is sensitive in two places to the choice of cloud provider: the deployment solution and the storage solution.

The deployment solution we use is tied to the choice of cloud provider only during creation of the Kubernetes cluster with the deployment scripts. Helm interacts with the Kubernetes cluster directly and not the cloud provider, so it remains cloud-agnostic. For cloud providers that offer a managed Kubernetes service, they typically offer command line interface (CLI) tools for creating and managing a Kubernetes cluster. In other cases, tools like `kops` and `Kubespray` enable cluster creation on a wide variety of public clouds as well on private computing clusters. Our deployment scripts can be extended in the future to accommodate other clouds using these tools.

The storage solution has a tighter coupling to the cloud provider that leads to potential lock-in with the cloud provider as well as issues with sharing data across clouds. While we have chosen Amazon S3 as our storage solution, similar object storage products from other cloud providers can be used. Apache Spark can access data stored in any object store that uses the S3 API. Some cloud providers offer object storage products that expose APIs that are compliant or complementary to the S3 API, which makes both accessing data between clouds feasible. For example, both Google Cloud Platform (GCP) and DigitalOcean provide storage services that have some or full interoperability with S3. This means a user can expect to deploy our system on a cloud with a compatible object store and use that object store with few or no changes in how the data are accessed. However, transferring large amounts

[22]Google Cloud Platform (GCP), Microsoft Azure, IBM Cloud, DigitalOcean, and the National Science Foundation funded Jetstream Cloud is a short list of cloud providers.

of data between clouds through the internet (referred to as "egress") remains very costly, making multi-cloud data access infeasible in practice. AWS quotes data transfer fees of \$0.05-\$0.09 per GB depending on the total volume transferred in a month. This means that a user who has deployed our system in a cloud other than AWS cannot expect to access large amounts of data stored within Amazon S3. This sets the expectation that our system will be deployed in the cloud where a user's data and potentially other relevant and desirable data sets are located. Notably, this challenge persists, at a smaller scale, within an individual cloud due to data transfer costs between geographically distinct data centers (often called regions). AWS quotes data transfer costs between regions at \$0.01-\$0.02 per GB. However, unless very low latency for data access from many countries/continents is required, a user or organization can likely choose and stick to a single region when storing data and acquire computing resources.

Slightly different system architectures allow for easier multi-cloud data access. For example, the Jupyter Kernel Gateway and Jupyter Enterprise Gateway projects can be used to access computing resources that are distributed across multiple clusters.[23] Both of these projects provide a method to create and access a running process in a remote cluster. This allows one to create several Kubernetes clusters in different clouds where desirable datasets are located and use a single JupyterHub as an entrypoint to access data stored in multiple clouds. While this proposed solution does not bring the data closer together, which would be desirable for applications that require jointly analyzing datasets from multiple sources in different clouds, it does allow for baseline multi-cloud data access. True multi-cloud data access is likely to remain infeasible without significant decreases in egress costs or prior agreement on where to store data from dataset stakeholders. New services, such as Cloudflare R2, that provide cloud storage with zero or near-zero egress cost brightens the prospects for cheap, multi-cloud data transfer and would lift the requirement for consensus among stakeholders.

[23] See https://github.com/jupyter-server/kernel_gateway and https://github.com/jupyter-server/enterprise_gateway

Name	Data Size (GB)	# Objects (10^9)
SDSS	65	0.77
AllWISE	349	0.81
Pan-STARRS 1	402	2.2
Gaia DR2	421	1.8
ZTF	4100	1.2
Total	5337	8.9

Table 2.1: The sizes of each of the catalogs available on the ZTF science platform along with the total data volume.

2.3 A Deployment for ZTF Analyses

To demonstrate the capabilities of our system and verify its utility to a science user, we deployed to enable the analysis of data from the Zwicky Transient Facility (ZTF). Section 2.3.1 describes the catalogs available through this deployment, Section 2.3.2 demonstrates the typical access pattern to the data using the AXS API, and Section 2.3.3 showcases a science project executed on this platform.

2.3.1 Catalogs available

Table 2.1 enumerates the catalogs available to the user in this example deployment. We provide a catalog of light curves from ZTF, created from de-duplicated match files. The most recent version of these match files have a data volume of $\sim$ 4 TB describing light curves of $\sim$ 1 billion+ objects in the "g", "r", and "i" bands. In addition, we provide access to catalogs from the data releases of the SDSS, Gaia, AllWISE, and Pan-STARRS surveys for convenient cross matching. The system allows users to upload, cross match, and share custom catalogs in addition to the ones provided, using the method described in 2.2.3.

2.3.2 Typical workflow

Users can query the available catalogs through the AXS/Spark Python API. For example, a user loads a reference to the ZTF catalog like so:

```
import axs
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
catalog = axs.AxsCatalog(spark)
ztf = catalog.load('ztf')
```

The `spark` object represents a Spark SQL Session and a connection to a Hive metastore database, which stores metadata for accessing the catalogs. This object is used as a SQL backend when creating the `AxsCatalog`, which acts as an interface to the available catalogs. Catalogs from the metastore database are loaded by name using the AXS API. Data subsets can be created by selecting one or more columns:

```
ztf_subset = ztf.select('ra', 'dec', 'mag_r')
```

`AxsCatalog` Python objects can be crossmatched with one another to produce a new catalog with the crossmatch result:

```
gaia = catalog.load('gaia')
xmatch = ztf.crossmatch(gaia)
```

The `xmatch` object can be queried like any other `AxsCatalog` object. Spark allows for the creation of User-Defined Functions (UDFs) that can be mapped onto rows of a Spark DataFrame. The following example shows how a Python function that converts an AB magnitude to its corresponding flux in janskys can be mapped onto all ∼63 billion r-band magnitude measurements from ∼1 billion light curves in the ZTF catalog (in parallel):

```
from pyspark.sql.functions import udf
from pyspark.sql.types import ArrayType
from pyspark.sql.types import FloatType
import numpy as np
```

```
@udf(returnType=ArrayType(FloatType()))
def abMagToFlux(m):
    flux = ((8.90 - np.array(m))/2.5)**10
    return flux.tolist()
ztf_flux_r = ztf.select(
    abMagToFlux(ztf['mag_r']).alias("flux_r")
)
```

2.3.3 Science case: Searching for Boyajian star Analogues

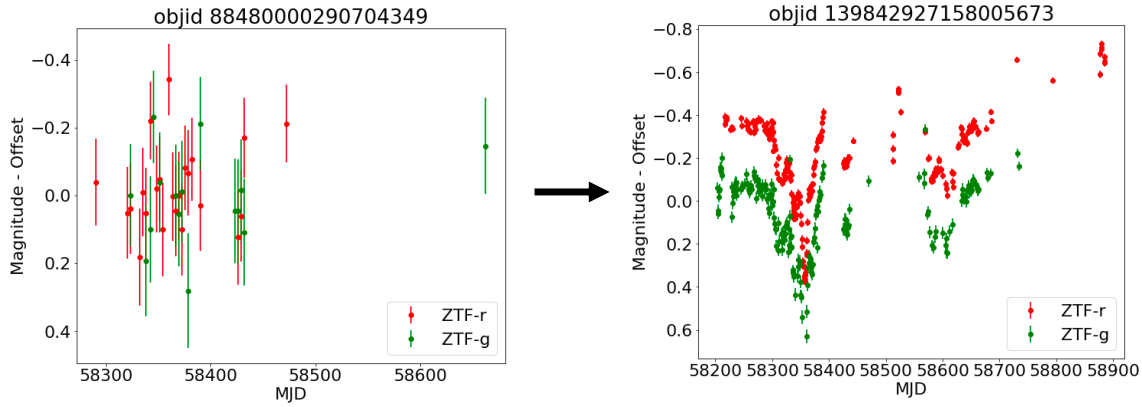
We test the ability of this platform to enable large-scale analysis by using it to search for Boyajian star (Boyajian et al., 2016) analogs in the ZTF catalog. The Boyajian star, discovered with the Kepler telescope, dips in its brightness in an unusual way. We intend to search the ZTF catalog for Boyajian-analogs, other stars that have anomalous dimming events, which will be fully described in Boone et al. (in prep.); here we limit ourselves to aspects necessary for the validation of the analysis system. The main method for our Boyajian-analog searches relies on querying and filtering large volumes of ZTF light curves using AXS and Apache Spark in search of the dimming events. Objects of interest are then spatially cross matched against the other catalogs available, for example to the Gaia catalog to create a color-magnitude diagram and the AllWISE catalog to identify if there is excess flux in the infrared. This presents an ideal science-case for our platform: the *entire ZTF catalog* must be queried, filtered, analyzed, and compared to other catalogs repeatedly in order to complete the science goals.

We wrote custom Spark queries that search the ZTF catalog for dimming events. After filtering the light curves, we created a set of UDFs for model fitting that wrap the optimization library from the `scipy` package. These UDFs are applied to the filtered light curves to parallelize least-squared fitting routines of various models to the dipping events. Figure 2.5 shows an outline of this science process using AXS.

The use of Apache Spark speeds up queries, filtering, and fitting of the data tremendously when deployed in a distributed environment. We used a Jupyter notebook on our platform

```
[1] catalog = axs.AxsCatalog(spark)
    df = catalog.load("ztf")
```

```
[2] dippers = find_and_filter_dips(df)
```




```
[3] fits = fit_light_curves(dippers)
```

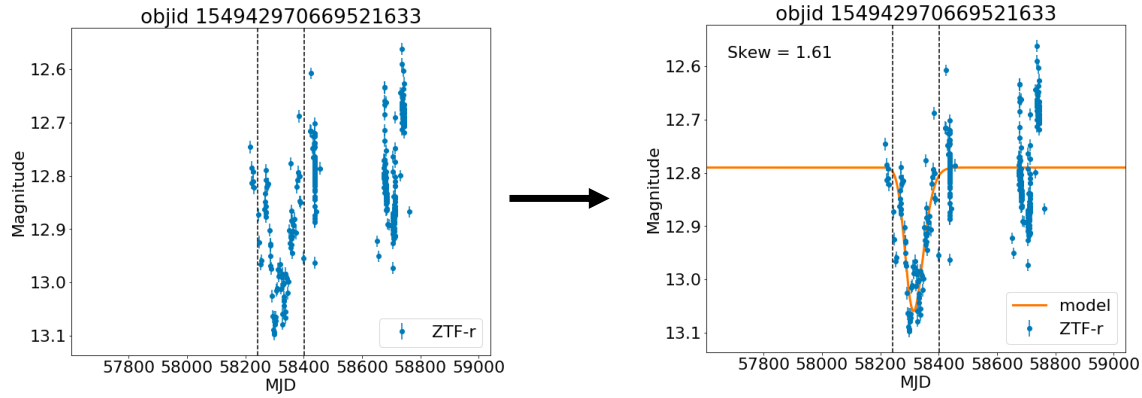


Figure 2.5: An example analysis (boiled down to two lines) that finds light curves in the ZTF light curve catalog with a dimming event. (1) shows how the ZTF catalog is loaded as a Spark DataFrame (`df`), (2) shows the product of filtering light curves for dimming events, and (3) shows the result of fitting a model to the remaining light curves. This process exemplifies that analyses can often be represented as a filtering and transformation of a larger dataset, a process that Spark can easily execute in parallel.

to allocate a Spark cluster of consisting of 96 `t3.2xlarge` EC2 instances. Each instance had access to 8 threads running on an Intel Xeon Platinum 8000 series processor with 32 GiB of RAM, creating a cluster with 768 threads and 3,072 GiB of RAM. We used the Spark cluster to complete a complex filtering task on the full 4 TB ZTF data volume in ∼three hours. The underlying system was able to scale to full capacity within minutes, and scale down once the demanding query was completed just as fast, providing extreme levels of parallelism at minimal cost. The total cost over the time of the query was ∼$100.

This same complex query was previously performed on a large shared-memory machine at the University of Washington with two AMD EPYC 7401 processors and 1,024 GiB of RAM. The query utilized 40 threads and accessed the catalog from directly connected SSDs. This query previously took a full two days to execute on this hardware in comparison to the ∼three hours on the cloud based science platform.[24] Performing an analysis of this scale would not be feasible if performed on a user's laptop using data queried over the internet from the ZTF archive.

In addition, the group was able to gain the extreme parallelism afforded by Spark without investing a significant amount of time writing Spark-specific code. The majority of coding time was spent developing science-motivated code/logic to detect, describe, and model dipping events within familiar Python UDFs and using familiar Python libraries. In alternative systems that provide similar levels of parallelism, such as HPC systems based on batch scheduling, a user would typically have to spend significant time altering their science code to conform with the underlying software and hardware that enables their code to scale. For example, they may spend significant time re-writing their code in a way that can be submitted to a batch scheduler like PBS/Slurm, or spend time developing a leader/-follower execution model using a distributed computing/communication framework such as OpenMPI. Traditional batch scheduling systems running on shared HPC resources typically have a queue that a user's program must wait in before execution. In contrast, our platform scales on-demand to the needs of each individual user.

This example demonstrates the utility of using cloud computing environments for science:

[24]The speedup realized of 48 hours/3 hours = 16× is roughly consistent with the increase in threads available to execute the query 768 threads/40 threads = 19.2×.

when science is performed on a platform that provides on-demand scaling using tools that can distribute science workloads in a user-friendly manner, time to science is minimized.

2.4 Scalability, Reliability, Costs, and User Experience

Our system is expected to scale both in the number of simultaneous users and to the demands of a single user's analysis. In the former case, JupyterHub and its built in proxy can scale to access by hundreds of users as its workload is limited to routing simple HTTP requests. In the latter case, data queries by individual users are expected to scale to very many machines, allowing for fast querying and transforming of very large datasets. Section 2.4.1 summarizes tests to verify this claim.

2.4.1 Scaling Performance

We performed scaling tests to understand and quantify the performance of our system. We tested both the "strong scaling" and "weak scaling" aspects of a simple query. Strong scaling indicates how well a query with a fixed data size can be sped up by increasing the number of cores allocated to it. On the other hand, weak scaling indicates how well the query can scale to larger data sizes; it answers the question "can I process twice as much data in the same amount of time if I have twice as many cores?"

Figure 4.10 shows the strong and weak scaling of a simple query, the sum of the "RA" column of a ZTF light curve catalog, which contains $\sim 3 \times 10^9$ rows, stored in Amazon S3. This catalog is described in more detail in section 2.3.1. In these experiments, speedup is computed as

$$\text{speedup} = t_{\text{ref}}/t_N \tag{2.1}$$

where t_{ref} is the time taken to execute the query with a reference number of cores while t_N is the time taken with N cores. For the weak scaling tests, scaled speedup is computed as

$$\text{scaled speedup} = t_{\text{ref}}/t_N \times P_N/P_{\text{ref}} \tag{2.2}$$

which is scaled by the problem size P_N with respect to the reference problem size P_{ref}. We chose to scale the problem size directly with the number of cores allocated; the 96-core

query had to scan the entire catalog, while the 1-core query had to scan only 1/96 of the catalog. Typically, the reference number of cores is 1 (sequential computing), however we noticed anomalous scaling behavior at low numbers of cores, and so we set the reference to 16 in Fig. 4.10.

In our experiments, we used `m5.large` EC2 instances to host the Spark executor processes, which have 2 vCPU and 8 GiB of RAM allocated to them. The underlying CPU is an Intel Xeon Platinum 8000 series processor. The Spark driver process was started from a Jupyter notebook server running on a `t3.xlarge` EC2 instance with 4 vCPU and 16 GiB of RAM allocated to it. The underlying CPU is an Intel Xeon Platinum 8000 series processor. Single `m5.large` EC2 instances have a network bandwidth of 10 Gbit/s while the `t3.xlarge` instance has a network bandwidth of 1 Gbit/s. Amazon S3 can sustain a bandwidth of up to 25 Gbit/s to individual Amazon EC2 instances. Both the data in S3 and all EC2 instances lie within the same AWS region, us-west-2. The `m5.large` EC2 instances were spread across three "availability zones" (separate AWS data centers): us-west-2a, us-west-2b, and us-west-2c. This configuration of heterogeneous instance types, network speeds, and even separate instance locations represent a typical use-case of cloud computing and offers illuminating insight into performance of this system with these "worst-case" optimization steps.

The weak scaling test showed that scaled speedup scales linearly with the number of cores provisioned for the query; twice the data can be processed in the same amount of time if using twice the number of cores. In other words, for this query, the problem of "big data" is solved simply by using more cores. The strong scaling test showed expected behavior up to vCPU/16 = 5. Speedup increased monotonically with diminishing returns as more cores were added. Speedup dropped from 2.50 with vCPU/16 = 5 to 2.05 with vCPU/16 = 6, indicating no speedup can be gained beyond vCPU/16 = 5. Drops in speedup in a strong scaling test are usually due to real world limitations of the network connecting the distributed computers. As the number of cores increases, the number of simultaneous communications and the amount of data shuffled between the single Spark driver process and the many Spark executor processes increases, potentially reaching the latency and bandwidth limits of the network connecting these computers.

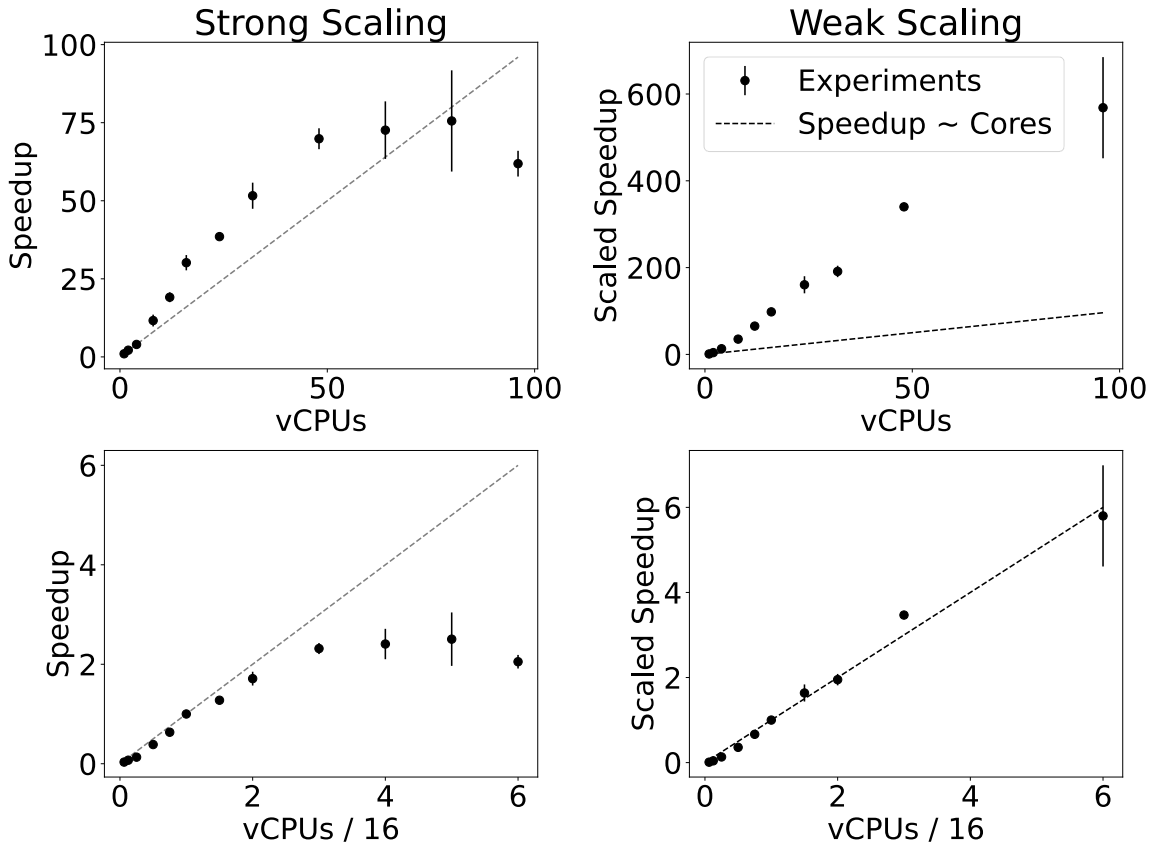


Figure 2.6: Speedup computed in strong scaling (left) and weak scaling (right) experiments of a simple Spark query that summed a single column of the ZTF catalog, $\sim 3 \times 10^9$ rows. Speedup is computed using Eq. 2.1 and scaled speedup is computed using Eq. 2.2. For each value of vCPU, the query was executed several (3+) times. For each trial, the runtime was measured and speedup calculated. Each point represents the mean value of speedup and error bars indicate the standard deviation. The first row shows speedup computed using sequential computing (vCPU = 1) to set the reference time and reference problem size. The second row shows speedup computed using 16 vCPU to set the reference. With sequential computing as the reference, we observe speedup that is abnormally high in both the strong and weak scaling case. By adjusting the reference point to vCPU = 16, we find that we can recover reasonable weak scaling results and expected strong scaling results for a small to medium number of cores. Using the adjusted reference, we observe in the strong scaling case diminishing returns in the speedup as the number of cores allocated to the query increases, as expected. The weak scaling shows optimistic results; the speedup scales linearly with the catalog size as expected.

2.4.2 Caveats to Scalability

As mentioned in section 2.2.4, the use of a shared NFS can limit scalability with respect to the number of simultaneous users. We recommend the administrators of new deployments of our platform consider the access pattern of user data and code on NFS to guarantee scalability to their desired number of users. Carefully designed hybrid models of code and data storage that utilize NFS, EFS, and the Docker image itself (stored on EBS) can be developed that will likely allow the system scale to access from hundreds of users.

2.4.3 Reliability

In general, the system is reliable if individual components (i.e. virtual machines or software applications) fail. Data stored in S3 are in practice 100% durable; at the time of writing, AWS quotes "99.999999999% durability of objects over a given year." Data stored in the EBS volume backing the NFS server are similarly durable – 99.999% at the time of writing. We choose to back up these data using Amazon EBS snapshots on a daily basis so we can recover the volume in the event of volume deletion or undesirable changes.

Kubernetes as a scheduling tool is resilient to failures of individual applications. Application failures are resolved by rescheduling the application on the cluster, perhaps on another node, until a success state is reached. When the Kubernetes cluster autoscaler is used, then the cluster becomes resilient to the failure of individual nodes. Pods that are terminated from a node failure will become unschedulable, which will trigger the cluster autoscaler to scale the cluster up to restore the original size of the cluster. For example, if the user's Jupyter notebook server is unexpectedly killed due to the loss of an EC2 instance, it will re-launch on another instance on the cluster, with loss of only the memory contents of the notebook server and the running state of kernels. The same is true of each of the individual JupyterHub and Spark components. Apache Spark is fault-tolerant in its design, meaning a query can continue executing if one or all of the Spark executors are lost and restarted due to loss of the underlying nodes. Similar loss of the driver process (on the Jupyter notebook server) results in the complete loss of the query.

We have run different instances of this platform for approximately three years in support

of science workloads at UW, the ZTF collaboration, a number of hackathons, and for the LSST science collaborations. Over that period, we have experienced no loss of data or nodes.

2.4.4 Costs

This section enumerates the costs associated with running this specific science platform. Since cloud computing costs can be variable over time, the costs associated with this science platform are not fixed. In this section, we report costs at the time of manuscript submission as well as general information about resource usage so costs can be recomputed by the reader at a later date.

We describe resource usage along two axes: interactive usage and core hours for data queries. Interactive usage encompasses using a Jupyter notebook server for making plots, running scripts and small simulations, and collaborating with others. Data queries encompass launching a distributed Spark cluster to access and analyze data provided on S3, similarly to the methods described in Sec. 2.3.3. Equation 2.3 provides a formula for computing expected monthly costs given the number of users N_u, the cost of each user node C_u, the cost of the Spark cluster nodes C_s, the estimated time spent per week on the system t_u, and the number of node hours used by each user for Spark queries in a month t_s:

$$\begin{aligned} \text{Cost}_{\text{storage}} &= N_u \times 200 \times 0.08 \times (t_u \times (30/7) + t_s), \\ \text{Cost}_{\text{machines}} &= N_u \times (C_u \times t_u \times (30/7) + C_s \times t_s), \\ \text{Cost} &= \text{Cost}_{\text{storage}} + \text{Cost}_{\text{machines}}. \end{aligned} \tag{2.3}$$

Fixed in the equation are constants describing the amount (200 GB) and cost of ($0.08/GB/month) of EBS-backed storage allocated for each virtual machines. Additionally, the term (30/7) converts weekly costs to monthly costs. Node hours can be converted to core hours by multiplying t_s by the number of cores per node.

Table 2.2 enumerates the fixed costs of the system as well as the variable costs, calculated using Eq. 2.3, assuming different utilization scenarios, varying the number of users (N_u), the amount interactive usage per week (t_u), and amount of Spark query core hours each month (t_s). The fixed costs of the system total to $328.51/month, paying for:

1. a small virtual machine, `t3.medium`, for the JupyterHub web application, proxy application, and NFS server ($29.95/month) with 200 GB EBS-backed storage ($16.00/month);

2. two reserved nodes for incoming users at the default virtual machine size of `t3.xlarge` ($119.81/month) with 200 GB EBS-backed storage each ($32.00/month);

3. EBS-backed storage for the NFS server for user files ($8.00/month);

4. and storage of 5,337 GB of catalog data on Amazon S3 ($122.75/month).

Virtual Machines

Type	Unit Cost	Amount	Total
Services (`t3.medium`[25])	$0.0416/hour/node	1 node	$29.95/month
Users (`t3.xlarge`)	$0.1664/hour/node	2 nodes + variable	$119.81/month + variable
Spark Clusters (`t3.xlarge` Spot[26])	$0.0499/hour/node	variable	variable

Storage

Type	Unit Cost	Amount	Total
Catalogs (S3[27])	$0.023/GB/month	5,337 GB	$122.75/month
NFS (EBS[28])	$0.08/GB/month	100 GB	$8.00/month
Node Storage (EBS)	$0.08/GB/month/node	200 GB/node	$48.00/month + variable

Fixed Costs

Type	Total
Virtual Machines	$149.76/month
Storage	$178.75/month

[25] On-Demand pricing in region `us-west-2`: https://aws.amazon.com/ec2/pricing/on-demand/

[26] Spot pricing in region `us-west-2`: https://aws.amazon.com/ec2/spot/pricing/

[27] For the first 50 TB: https://aws.amazon.com/s3/pricing/

[28] General purpose SSD (gp3): https://aws.amazon.com/ebs/pricing/

All	$328.51/month

Variable Costs

Number of Users	**Interactive Usage** (hours/week/user)	**Spark Query Core Hours** (/user/month)	**Total**
10	12	512	$189.32/month
		2048	$466.27/month
	40	512	$415.67/month
		2048	$692.62/month
100	12	512	$1,893.22/month
		2048	$4,662.71/month
	40	512	$4,156.69/month
		2048	$6,926.18/month

Table 2.2: Fixed and variable costs associated with running this analysis platform on Amazon Web Services. This summary provides cost estimates for renting virtual machine and storing data. Additional costs on the order of ∼$10 due to network communication and data transfer are excluded from these results. Reasonable low and high estimates are chosen for the number of active users and the amount of interactive usage they have with the system. The number of Spark query core hours used by each user per month is a guess, but the high end estimate is similar to the core hours used during the analysis in Sec. 2.3.3.

Variable costs are harder to estimate, but Table 2.2 outlines several scenarios to get a sense for the lower/upper limits to costs. 10 scientists using the platform for 4 hours per day 3 days per 7 day week, each using 512 core hours for Spark queries each month (equivalent to 16 hours with a 32 core cluster) adds a cost of $189.32/month. On the other hand, 100 scientists using the platform for 8 hours per day 5 days per 7 day week, each using 2048 core hours for Spark queries each month (64 hours with a 32 core cluster) adds a cost of $6,926.18/month. There are additional costs on the order of ∼$10 that we don't factor into this analysis. Specifically:

1. network communication between virtual machines in different availability zones, introduced when scaling a Spark cluster across availability zones;

2. data transfer costs in the form of S3 GET API requests (data transfer to EC2 virtual machines in the same region is free), introduced in each query executed against the data;

3. and network communication between virtual machines and users over the internet, introduced with each interaction in the Jupyter notebook through the user's web browser.

Each of these costs are minimal, and so we don't include them in our analysis. However, they are worth mentioning because they can scale to become significant. Spark queries requiring GB/TB data shuffling between driver and executors should restrict themselves to a single availability zone to avoid the costs of (1). Costs from (2) are unavoidable, but care should be taken so no S3 requests occur between different AWS regions and between AWS and the internet. Finally, (3) can balloon in size if one allows arbitrary file transfers between Jupyter servers and the user or allows large data outputs to the browser.

The number of core hours for queries is a parameter that will need to be calibrated using information about usage of this type of platform in the real-world. The upper limit guess of 2048 core hours per user per month is roughly equivalent to each user running an analysis similar to that described in Sec. 2.3.3 each month. By monitoring interactive usage of our

own platform and other computation tools, we estimate that realistic usage falls closer to the lower limits we provide; few users will use the platform continuously in an interactive manner, and even fewer will be frequently executing large queries using Spark.

2.4.5 Dynamic Scaling

Recent versions of Apache Spark provide support for "dynamic allocation" of Spark executors for a Spark cluster on Kubernetes.[29] Dynamic allocation allows for the Spark cluster to scale up its size to accommodate long-running queries as well as scale down its size when no queries are running. Figure 2.7 shows pictorially this scaling process for a long-running query started by a user. This feature is expected to reduce costs associated with running Spark queries since Spark executors are added and removed based on query status, not cluster creation. This means the virtual machines hosting the Spark executor processes will be free more often either to host the Spark executors for another user's query or be removed from the Kubernetes cluster completely.

2.4.6 User Experience

While the experience of using the science platform is largely identical to using a local or remotely-hosted Jupyter notebook server, the use of containerized Jupyter notebook servers on a scalable compute resource introduces a few notable points of difference. First, similarly to using a remotely-hosted Jupyter notebook, the filesystem exposed to the user has no direct connection to their personal computer, an experience that can be unintuitive to the user. File uploads and download can be facilitated through the Jupyter interface, but the process remains clunky. For streamlined file transfer, the user must fall back to using an SSH client and utilities like `scp` or `rsync`. In future deployments of this system, it is likely that new user interfaces will need to be produced to maximize usability of the filesystem.

Additionally, in order to allow for scale-down of the cluster, notebook servers are typically shut down after a configurable period of inactivity using the `jupyterhub-idle-culler`

[29]Since Spark version 3.0.0 by utilizing shuffle file tracking on executors as an alternative to an external shuffle file service, which is awaiting support in Kubernetes. See: https://spark.apache.org/docs/3.0.0/configuration.html#dynamic-allocation

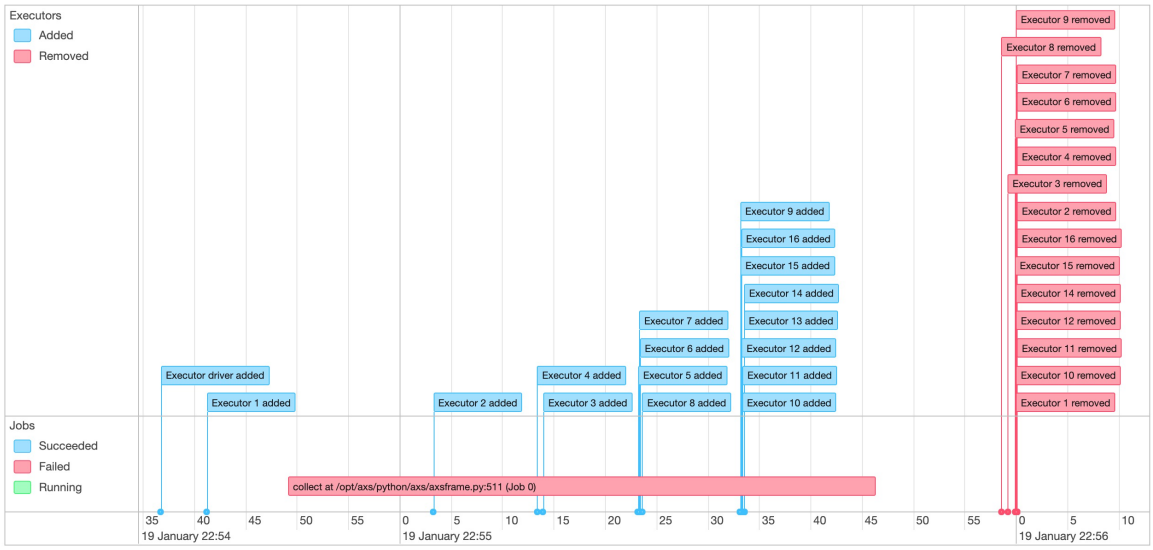


Figure 2.7: A screenshot of the job timeline from the Spark UI when dynamic allocation is enabled. A long-running query is started, executing with a small number of executors. As the query continues, Spark adds exponentially more executors to the cluster at a user-specified interval until the query completes or the max number of executors is reached. Once the query completes (or is terminated, as shown here), the Spark executors are removed from the cluster.

service. A period of $\sim 1-8$ hours is typical for deployments of this science platform. This has the positive effect of reducing costs but at a detriment to the user experience. At the time of writing, inactivity is determined in terms of browser connectivity, so a user cannot expect to leave code running longer than the cull period e.g. overnight. Juric et al. (2021) have implemented functionality to checkpoint the memory contents of the notebook server to disk before stopping with the ability to restore the server to a running state at will. Such checkpoint/restore functionality solves the issue of interrupting running code when culling servers; however, this still does not allow for codes to run longer than the cull period. At the time of writing, additional functionality is being added to the `jupyterhub-idle-culler` service to allow for fine-grained control over which servers are culled and when.

Finally, the underlying scalable architecture introduces server start-up latencies that are noticeable to the user. Virtual machines that host notebook servers and Spark cluster executors are requested from AWS on-demand by the user. The process of requesting new virtual machines from AWS, downloading relevant Docker images to that machine, and starting the notebook/Spark Docker container can take up to $\sim$5 minutes.[30] The user can encounter this latency when logging onto the platform and requesting a server. They also encounter this latency when creating a distributed Spark cluster, as many machines are provisioned on-demand to run Spark executors. The log-in latency can be mitigated by keeping a small number of virtual machines in reserve so that an incoming user can instantly be assigned to a node. The `zero-to-jupyterhub` Helm chart implements this functionality through its `user-placeholder` option. This functionality schedules placeholder servers on the Kubernetes cluster that will be immediately evicted and replaced when a real user requests a server. Additionally, to avoid Docker image download times, relevant Docker images can be cached inside a custom-built virtual machine image (in AWS lingo, the Amazon Machine Image or AMI) that the virtual machine is started from. An alternative solution to this would be to place all incoming users on a shared machine, an equivalent to a "log-in node", before moving them to a larger machine at the user's request or automatically once a new server is provisioned from the cloud provider – a process known as live migration.

[30]This time is dependent on the individual cloud provider. DigitalOcean, another cloud provider, can provision virtual machines in $\sim$1.5 minutes based on the experience of the authors.

Juric et al. (2021) provides a path towards live migration of containerized Jupyter notebook servers, but this advanced functionality remains to be implemented with a JupyterHub deployment on Kubernetes.

2.5 Conclusions

We've described an architecture of a Cloud-based science platform as well as an implementation on Amazon Web Services that has been tested with data from the Zwicky Transient Facility. The system is shown to scale to and allow parallel analysis with O(10TB) sized tabular, time-series heavy, datasets. It has enabled a science project that utilizes a 1 billion+ ZTF light curve catalog in full, while requiring minimal effort from domain scientists to scale their analysis from a single light curve to the full catalog. The system demonstrates the utility of elastic computing, the I/O capacity of the cloud, and distributed computing tools like Spark and AXS.

This work should be viewed in the context of exploring the feasibility of making more astronomical datasets available on cloud platforms, and providing services and platforms – such as the one described here – to combine and analyze them. Using this platform, it is both feasible and practical to perform large-scale cross-catalog analyses using any catalog uploaded to AWS S3 in the AXS-compatible format.[31] This enables any catalog provider – whether large or small – to make their data available to the broad community via a simple upload. Additionally, other organizations can stand up their own services on the Cloud – either use-case specific services or broad platforms such as this-one – to access the data using the same S3 storage API.

In this regime, the roles of data archive and data user can be further differentiated, to the benefit of the user and perhaps at reduced cost. Data archivers upload their data to the cloud and bear the cost of storage. These costs are manageable, even by small organizations; storing 1 TB of data in S3 costs ∼$25 per month. Cost scales dramatically when considering datasets at the PB level and timescales extending over years: a 1 PB

[31] For additional practicality, catalogs must also be uploaded to the same AWS region due to constraints on data transfer costs between AWS data centers. However, given the advent of cloud storage solutions such as Cloudflare R2 with zero data transfer costs, it seems plausible that this practicality constraint may soon be lifted.

dataset will cost $\sim\$3,000,000$ over 10 years assuming storage costs do not decrease. At these scales special pricing contracts may have to be negotiated between the cloud provider and archive. Additional cost scales with the number of requests for this data and the amount of data transferred. So-called "requester-pays" pricing models, supported by some cloud providers, can offload access and data transfer costs to the user. A user – or perhaps an organization of users – can deploy a system like ours at reasonable cost to access the data in a given cloud. In this case, the cost of analysis decouples from the cost of storage: it is the user who controls the number of cores utilized for the analysis, and any additional ephemeral storage used for the analysis. It is easy to imagine the user – as a part of their grant – being awarded cloud credits for their research, which could be applied towards these costs (Norman et al., 2021). Finally, an intermediary role may appear: the science platform provider, which has an incentive towards continuous improvements of science platforms and associated tools, which are now best viewed as systems utilized by astronomers to enable the exploration of a multitude of datasets available. The incentive of a science platform provider is to maximize science capability while minimizing the cost to the user, who now has the ability to "shop around" with their cloud credits for a system most responsive to their needs.

Chapter 3

A LARGE IMAGE PROCESSING CAMPAIGN FOR DISCOVERY OF TRANS-NEPTUNIAN OBJECTS

3.1 Introduction

The history of the solar system, its formation and evolution, is encoded in the dynamical structure and composition of its population of major and minor planets (DeMeo & Carry, 2014; Bottke et al., 2005). The major planets are hypothesized to have undergone significant migration during early times in the solar system's history, sculpting the existing structure of the minor planets (Morbidelli et al., 2010; Morbidelli, 2010). The population of Trans-Neptunian Objects (TNOs) has remained relatively pristine over the age of the solar system, meaning their present-day structure has likely been in place since the formation of the solar system. (Holman & Wisdom, 1993). The dynamics and compositions of TNOs therefore hold clues about the history of solar system formation (Fernández & Ip, 1984; Malhotra, 1993; Duncan et al., 1995; Hahn & Malhotra, 2005; Tsiganis et al., 2005; Levison et al., 2008; Nesvorný & Vokrouhlický, 2016).

These studies are currently limited by the number of known TNOs and the ability to detect new ones. At visible wavelength, TNOs are detected in reflected light from the Sun, resulting in an inverse fourth-power scaling of their flux with distance. This makes them extremely challenging to detect. Studies of this population have been limited to its largest bodies, and efforts to find smaller, fainter TNOs are often limited to "pencil-beam" surveys which utilize large aperture, small field-of-view telescopes such as the Hubble Space Telescope (Bernstein et al., 2004a). In the past two decades, the availability of large-aperture, wide-field of view telescopes paired with high quality CCD imagers have enabled surveys that are currently pushing the boundaries of TNO discovery by inventorying the outer solar system (Trujillo et al., 2001; Millis et al., 2002; Jones et al., 2006; Schwamb et al., 2010; Bannister et al., 2016). The Legacy Survey of Space and Time (LSST; Ivezić

et al. (2019)) promises to significantly extend this inventory of the outer solar system with its 10-year survey of the entire southern sky. The single visit depth achieved (magnitude 23.5 in the r-band) combined with the wide field observed promises to observe and constrain the orbits of $\approx 40,000$ TNOs, increasing the number of known TNOs by a factor of $\approx 10\times$ (LSST Science Collaboration et al., 2009; Kurlander et al., 2024).

The DECam Ecliptic Exploration Project (DEEP; Trilling et al. (2024)) is one of these surveys, utilizing the 3 square degree DECam CCD imager on the 4-meter Blanco telescope at the Cerro Tololo Inter-American Observatory (CTIO). It is an ambitious project with an aim to provide high quality multi-year orbits of 5000+ TNOs as faint as magnitude 27 in the VR-band. We provide a brief summary of this survey in 3.2; however, a series of papers from this collaboration provides a more detailed summary of its goals (Trilling et al., 2024), observation strategy (Trujillo et al., 2024), survey characterization (Bernardinelli et al., 2024), and initial results (Strauss et al., 2024a; Napier et al., 2024; Smotherman et al., 2024; Strauss et al., 2024b).

This survey utilizes synthetic tracking in post-processing to discover moving objects fainter than the single exposure limiting magnitude (Bernstein et al., 2004a; Shao et al., 2014a; Heinze et al., 2015a; Whidden et al., 2019). The ideal input to synthetic tracking algorithm are difference images, the result of subtracting a template of the sky from calibrated survey images (Alard & Lupton, 1998; Becker, 2015). The template itself is typically constructed through coaddition of calibrated images (Zackay & Ofek, 2017). Tools and algorithms are then required for coaddition and image subtraction, in addition to a system to apply these tools to the set of survey images and a data management system to handle their outputs. The LSST Science Pipelines (LSP; Bosch et al. (2018, 2019)) provides these algorithms and the ability to apply them to DECam images through robust and scalable data management and task execution systems (Jenness et al., 2022). Effectively applying the LSP in a data processing campaign requires an additional set of "campaign management" tools which automate and track the execution of steps of one or more data processing pipelines. We use the LSP for the processing of the DEEP survey images. In section 3.3, we provide an overview of the LSST Science Pipelines and introduce a custom-made and light-weight tool used for campaign management.

Constructing difference images requires a template of the sky, the construction of which is non-trivial. The quality of the template impacts the quality of resulting difference images. A poor quality template is an ineffective model of the sky, leading to difference image artifacts from poorly subtracted sources. A shallow template constructed from a small number of input exposures can also introduce noise in the difference image that overwhelms signal from faint objects. These two effects can dramatically impact the effectiveness of moving object discovery algorithms applied to the difference images, especially when recovering faint objects. An additional effect is introduced when constructing templates from survey images: the self-subtraction effect, where flux from a moving object enters the template and is thus subtracted in difference images. The template strategy for DEEP involves constructing templates from the survey images, thus requiring effort either to mitigate the self-subtraction effect or model it in the shift-and-stack procedure or analysis of shift-and-stack candidate discoveries, as performed in Smotherman et al. (2024). In section 3.4, we describe a set of experiments performed to construct an optimal template from the survey images available.

A large-scale data processing campaign was executed on the DEEP survey images using our optimized template strategy. We summarize the results of this campaign, including properties of the templates constructed and measurements of the computational resources utilized in 3.5.

3.2 Survey Overview

The DEEP survey is a 64-night survey of 4 regions of the sky around the invariable plane. The survey is split by time of year into two semesters A and B. For each semester, two regions of the sky are observed: the A0, A1, B0, and B1 fields. These regions are further sub-divided into individual pointings that tile the observed fields. Pointings are observed in a fan-out pattern that tracks the expected motion of TNOs, with the intention of linking TNO discoveries night to night and year to year of the survey. The sky coordinates of the individual pointings are varied year to year to track the mean motion of the population of TNOs expected to be observed. Figure 3.1 visualizes the observing pattern of the A0, A1, B0, and B1 fields, while Figure 3.2 visualizes the individual pointings from the survey for

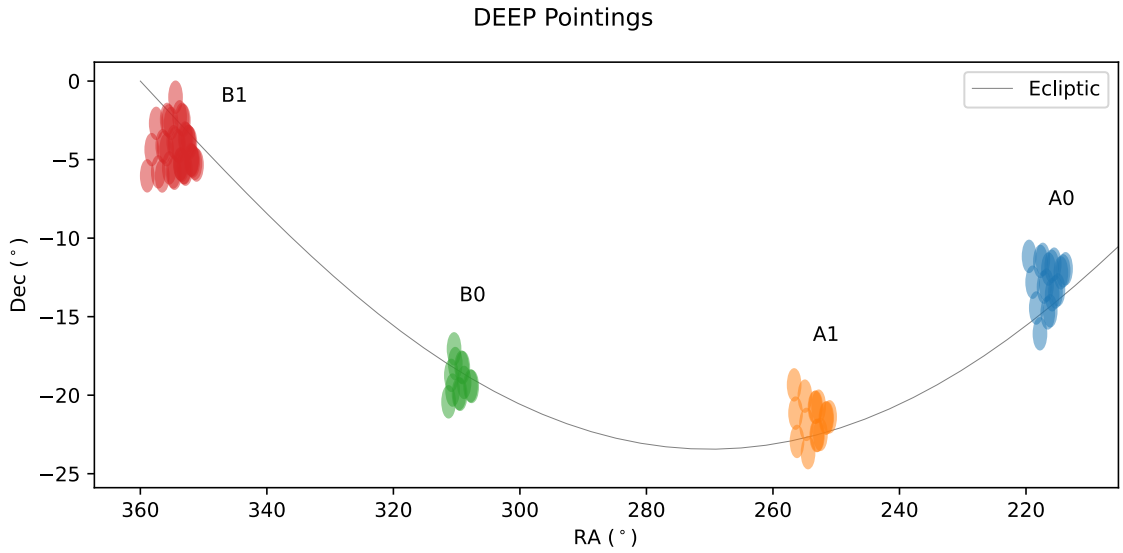


Figure 3.1: The sky-locations of the A0 (blue), A1 (orange), B0 (green), and B1 (red) DEEP fields. Circles represent an approximation of the DECam 3 deg^2 field-of-view. The gray line represents the location of the ecliptic plane on the sky.

each of these fields.

Over the duration of the survey, each of the pointings are observed at least once in a 2-4 hour long-stare pattern; taking 50-100 120-second VR-band images with DECam. The long-stare pointings are used to discover TNOs via digital tracking. Pointings are periodically re-observed in sequences of 5-10 120-second VR-band exposures. A few pointings are observed in bands other than VR to allow for color determination of discovered objects.

3.3 Image Processing with the LSST Science Pipelines

Herein we describe the tools and methods used to construct difference images from the DEEP survey images. Section 3.3.1 provides an overview of the LSST Science Pipelines, which were the main tool used for image reduction and difference image construction. Section 3.3.2 provides details on a campaign management system used to drive the data processing campaign with the LSP.

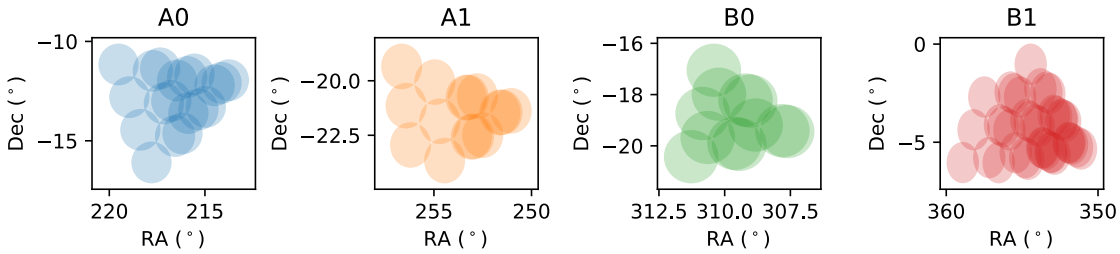


Figure 3.2: The individual pointings of the A0 (blue), A1 (orange), B0 (green), and B1 (red) DEEP fields. Circles represent an approximation of the DECam 3 deg^2 field-of-view.

3.3.1 Overview of the LSST Science Pipelines

The LSST Science Pipelines (LSP) are a sophisticated suite of tools for processing astronomical image data in a scalable, robust, and reproducible manner. The pipelines will be used to process the survey images from the LSST into survey data products produced by the nightly Alert Production (AP) pipelines, Prompt Processing pipelines, and the yearly Data Release Production (DRP) pipelines. Survey data volumes will easily exceed a petabyte, dwarfing the output from prior surveys. The LSP have been used successfully to process DECam images in Smotherman et al. (2021), Smotherman et al. (2024), Danieli et al. (2024), and Fraser et al. (2024) among others.

The LSP consists of core algorithms, written in Python and C++, that perform traditional data reduction routines, as well as a rich set of middleware that enables the operation of these algorithms on input images and catalogs. The algorithms are wrapped in configurable tasks implemented as Python classes. Tasks declare their input and output datasets using dataset types, which is a name paired with a set of dataset dimensions. Dimensions are properties of datasets that can be used to uniquely identify them. Example dimensions are the instrument from which a image was produced, for example DECam, the detector number in the instrument that the image comes from, or the photometric band that the image was taken in. Sequences of tasks can be chained together into a task graph by their input/output dependencies, matching declared input/output dataset types. In LSP par-

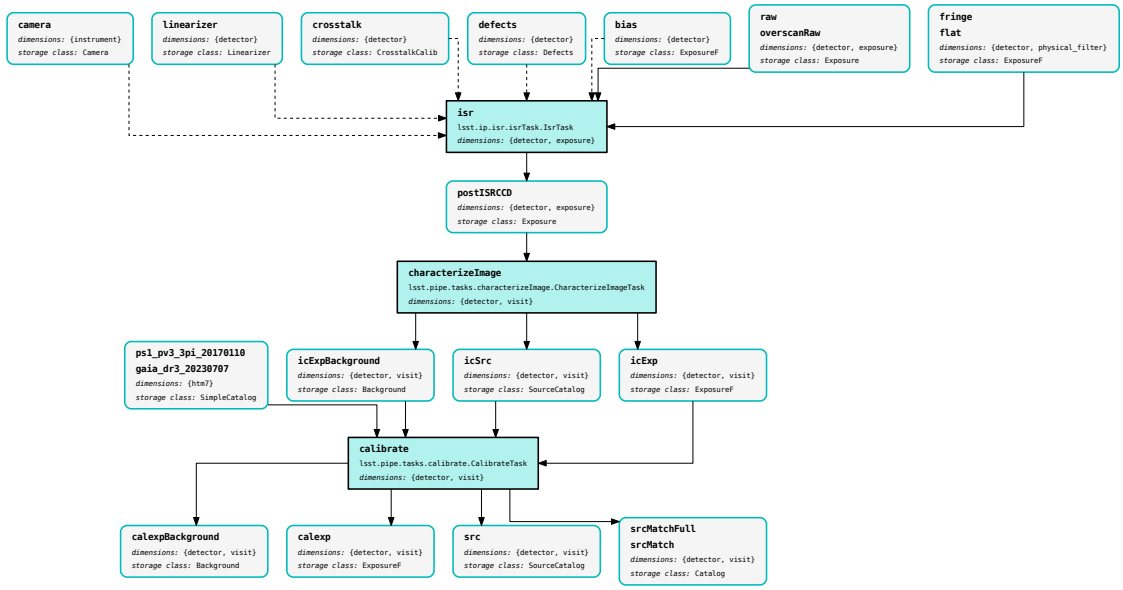


Figure 3.3: A visualization of an example pipeline. Datasets are represented as gray boxes while tasks are represented as cyan boxes. Arrows indicate the input/output relationship between datasets and tasks. Dashed lines represent "prerequisite" dataset inputs which must exist prior to pipeline execution. This pipeline represents the execution of tasks that will instrument correct, characterize, and calibrate a raw image.

lance, this task graph is called the quantum graph where individual nodes in the graph, called quanta, contain information about a task, its configuration, and its inputs/outputs. Pipelines are formed by declaring tasks, the names of their inputs and outputs, and their configurations in YAML files. An example pipeline is visualized in Fig. 3.3. The LSP distributes pipelines for operation of AP, DRP, and Calibration Products (CP) production on images from DECam as well as other instruments such as Hyper-Suprime Camera (HSC), and the LSST Camera and Commissioning Camera. The CP pipeline provides tasks for constructing calibration data products while the DRP pipeline provides tasks for single-frame calibration, coaddition, and image subtraction, providing a full pipeline for producing DECam difference images.

To build a realization of a quantum graph, the quantum graph builder queries the Butler to determine which data inputs exist and which data outputs need to be produced. The Butler is the main mechanism through which data is accessed by the LSP and the end user. The Butler stores datasets in the Datastore, which is an abstraction layer over a traditional file system or object store, and stores metadata about the datasets in the Registry, a set of tables in a database. The Registry contains information about which datasets exist in the Datastore, their dataset types, and their dimensions. Datasets are also associated with dimensions that enable temporal and spatial queries of datasets. The Datastore and Registry are joined together in the so-called Butler "repository".

Once a quantum graph is built, it can be submitted for execution to an execution provider. The execution provider executes the tasks contained in the quantum graph and produces their outputs. The execution provider is abstracted through the LSP Batch Processing System (BPS), which translates the quantum graph execution to one of several supported workflow management systems such as HTCondor, Pegasus, Panda, and Parsl (Thain et al., 2005; Deelman et al., 2019; Maeno et al., 2024; Babuji et al., 2019).

Individual datasets are contained in "collections" in the Registry, which comes in several types. RUN collections contain the outputs from any executed quantum graph or ingestion of raw images into the Butler. Datasets are required to exist in a RUN collection since the collection identifier provides an extra tag to a dataset that disambiguates it from others of the same dataset type. TAGGED collections contain datasets from one or more RUN collection that were manually associated into the TAGGED collection. CALIBRATION collections contain specially tagged calibration datasets. CHAINED collections are linked lists of RUN, TAGGED, CALIBRATION, and other CHAINED collections.

3.3.2 Automated end-to-end processing of DECam images

We developed a light-weight campaign management tool to support automated processing of DECam images using the LSST Science Pipelines with minimal intervention. This campaign management tool intends to bring a set of survey images from their raw form into a processed form, be they calibrated exposures or difference images. The campaign management tool

solves several problems faced when executing a large data processing campaign on DECam images. The first of these issues is the mirroring of the metadata and image data available in the NOIRLab archive for survey images. The second of these issues is long quantum graph generation time when operating the pipelines with a large number of inputs. The quantum graph generation time typically scales linearly with the number of inputs due to its reliance on individual database queries for resolving the data dependencies in the task graph. The third is the desire to recover from transient failures during pipeline execution. These failures may not be due to a failure in the task itself, but rather may be from transient environment factors, say due to a machine running out of memory, disk space, or is preempted/interrupted during task execution. The fourth is a method to handle provenance when executing multiple steps of a processing pipeline, tracking and making readily apparent the status and historical record of processing. Finally, a method is required to execute pipeline processing in a semi-autonomous way.

NOIRLab archive mirroring is achieved through a small script that queries the archive for the available raw exposures under a provided proposal ID. From this information, the observing nights are collated and used to query for available nightly bias and flat images. Nights for which no calibrations are available are collected and provided to the pipeline operator. The metadata for the raw and calibration images are stored in a local exposure database, represented as an Enhanced Character Separated Values (ECSV) file.[1][2] A utility is provided to query from the local exposure database and download subsets of the exposure database to a chosen location, represented as a flat directory of compressed FITS files. The utility uses the NOIRLab REST API to download images over HTTP. The utility checks the md5 hash of the downloaded file and compares it to the md5 hash reported from NOIRLab. If the hashes do not match, a query is made to the NOIRLab REST API to perform an integrity check on the file in the archive. The integrity of the file on disk, the integrity of the file in the archive, and whether the file has been downloaded are stored in a separate database, also represented as an ECSV file. This utility provides a method to stage DECam

[1]For processing at larger scale, a table in a database operated by a database management system such as SQLite or Postgres would be more appropriate.

[2]The ECSV format was introduced and specified in Astropy Proposal for Enhancement (APE) 6.

images and their metadata locally and filter out any images with integrity issues before ingestion into a Butler repository.[3]

Long quantum graph generation time is solved by processing survey data in smaller subsets. Processing the data via subsets also enables parallel execution of quantum graph generation and workflow submissions, resulting in a higher throughput of workflows and better utilization of the computing resources available for a data processing campaign. Subsets are derived by partitioning or sharding on a dimension of the input data. Telescope surveys, and their respective data reduction efforts, are already naturally partitioned temporally by observing night, where survey images are taken one night at a time and are often processed using calibrations that are constructed on a nightly basis. In LSP parlance, this corresponds to selecting on the `day_obs` dimension and its associated groups of exposures. Unfortunately, before version `w_2024_10` of the LSP, the `day_obs` dimension did not correspond to the observing night for DECam, so the observing night information collated in the previously described local exposure database is used as the partitioning value to group raw calibration and science images by their exposure number. When performing co-addition, or when operating other tasks that aggregate over on-sky regions, a spatial partitioning scheme is required. Spatial dimensions are provided through the patch dimension associated with a "skymap". Patches can be grouped arbitrarily into subsets of a desired size; however for co-addition, it is convenient to group patches with similar numbers of warp inputs together so that they can be executed under the same workflow as they are likely to share the same memory requirements during execution.[4]

Our campaign management system recovers from transient failures by operating in a loop 1) building and submitting the quantum graph for execution, and 2) marking failed tasks that meet certain criteria for retry at the end of execution. The loop terminates when all tasks have either executed successfully, or have failed in a way that the pipeline operator

[3]Data integrity issues in the NOIRLab archive have made it typical instead of rare for a downloaded survey image to be partially corrupted. While these data are sometimes still usable, we exclude them from consideration.

[4]This same grouping effect based on resource usage can likely be achieved through clustering of the quantum graph, an operation that occurs during workflow submission; however, the authors remained ignorant of how to effectively use this feature during design and operation of our processing campaign. Additionally, not all execution providers support resource requests assigned through LSP supported methods.

does not wish to recover from. When building the quantum graph, we use the behavior of the quantum graph builder to skip quanta for which a dataset is already available, noted by the presence of task metadata for that quantum in the Butler registry. We additionally have modified the default quantum graph builder to skip quanta that have previously failed, noted by the presence of a task log in the Butler registry while the task metadata is missing. Tasks are marked for retry by searching through the available task log file for matches to a set of pre-defined regular expressions that indicate a failure is transient.[5] Logs that match one of the predefined patterns are moved from their original {task_name}_log dataset type name into a new dataset type called {task_name}_log_retry. This method is not ideal, as the set of regular expressions is not known beforehand, and must be discovered by encountering them, introducing human labor into the operations of the pipelines. Pattern matching against log files is also in-flexible to changes in the logging messages of the tasks themselves. However, the list of patterns can be kept small and be made generic enough to be effective at catching most instances where a task should be retried.

Processing provenance is handled by using the RUN, CHAINED, and TAGGED collections available in the Butler registry. RUN collections contain datasets that are the direct result of pipeline processing. TAGGED collections contain datasets that are from one or more RUN collection, joined via manual association. CHAINED collections represent groups of RUN, CHAINED, and TAGGED collections and provide an entry-point to all datasets relevant to a single processing effort. We developed a method of naming the collections that provides a convenient hierarchy for automatic organization of RUN collections into a parent CHAINED collection. RUN collections are named according to the template {campaign}/{subset}/{proc_type}/{pipeline_name}/{pipeline_step}/{datetime} where campaign is a name for the data processing campaign, subset is the processing subset, proc_type is the processing type (one of bias, flat, drp, coadd, diff_drp), pipeline_name is the name of the YAML file that contains the pipeline configurations, pipeline_step is a selector of a single pipeline task or step in the pipeline YAML, and datetime is the ISO 8601 formatted datetime string representing when pipeline processing was initiated. For ex-

[5]Examples include MemoryError and std::bad_alloc which both indicate issues with the amount of memory available during task execution.

ample, DEEP/20190401/bias/cpBias/cpBiasIsr/20240102T030405Z would be the name assigned to the RUN collection that contains datasets from execution of the cpBiasIsr step of the cpBias pipeline to create bias calibrations for night 20190401 for the DEEP data processing campaign. RUN collections are grouped under parent CHAINED collections with name {campaign}/{subset}/{proc_type}, and the proc_type is used to determine pre-defined input collections that are also grouped under the parent. For example, the parent collection DEEP/20190401/bias is constructed by appending all of the RUN collections with names that match the glob pattern DEEP/20190401/bias/*, sorted by their datetime timestamp as well as appending the pre-defined input collection DECam/calib which contains pre-curated calibration datasets for DECam image processing. Processed nightly calibrations are certified and placed into CALIBRATION collections that follow the naming pattern {campaign}/{subset}/calib/{proc_type} e.g. DEEP/20190401/calib/bias and DEEP/20190401/calib/flat. The {subset} value can be further subdivided into finer grained subsets or to distinguish subsets processed with different configuration values of the pipelines. For example, the subset may be 20190401 when constructing a template for a single night using the default pipeline configuration, 20190401/median to indicate that the pipeline configuration was altered to use the median statistic, or 20190401/group_0 to operate processing on a subdivision of the 20190401 subset. Figure 3.4 provides a visual summary of the collection structure at the end of an example processing campaign. This simple structure for collection naming provides a summary of the status of pipeline processing just by listing the available collections in the Butler registry. The hierarchical structure also provides a convenient method to select RUN or CHAINED collections that contain datasets of interest. This is a crude but effective method for storage of metadata about collection contents and processing provenance.[6]

Semi-autonomous processing is enabled by encoding the previously outlined processing logic in discrete executable programs which can be chained together to process the full survey. When chained together, the programs can be expressed as a task graph that can be

[6]A more effective and scalable solution would involve creating a separate database that tags arbitrary collection names with user-defined metadata. Queries against this database would provide the information about the collections that contain datasets of interest.

Name	Type
DEEP/20190401/bias	CHAINED
DEEP/20190401/bias/cpBias/step{i}/{datetime}	RUN
DECam/calib	CALIBRATION
DEEP/20190401/bias/raw	TAGGED

Name	Type
DEEP/20190401/flat	CHAINED
DEEP/20190401/flat/cpFlat/step{i}/{datetime}	RUN
DEEP/20190401/calib/bias	CALIBRATION
DECam/calib	CALIBRATION
DEEP/20190401/flat/raw	TAGGED

Name	Type
DEEP/20190401/drp	CHAINED
DEEP/20190401/drp/DRP/step{i}/{datetime}	RUN
DEEP/fakes	RUN
refcats	CHAINED
refcats/ps1_pv3_3pi_20170110	RUN
refcats/gaia_dr3_20230707	RUN
skymaps	RUN
DEEP/20190401/calib/flat	CALIBRATION
DEEP/20190401/calib/bias	CALIBRATION
DEEP/calib	CALIBRATION
DECam/calib	CALIBRATION
DEEP/20190401/drp/raw	TAGGED

Name	Type
DEEP/allSky/{group}/coadd	CHAINED
DEEP/allSky/{group}/coadd/DRP/assembleCoadd/{datetime}	RUN
DEEP/allSky/{group}/coadd/warps	TAGGED
skymaps	RUN

Name	Type
DEEP/20190401/allSky/diff_drp	CHAINED
DEEP/20190401/allSky/diff_drp/DRP/step{i}/{datetime}	RUN
DEEP/allSky/coadd	TAGGED
DEEP/20190401/drp	CHAINED

Figure 3.4: Collections for bias, flat, drp, coadd, and diff_drp as printed from Butler query-collections. For brevity, the values {`i`}, {`datetime`}, and {`group`} are placeholders in collection names that follow the same pattern.

executed by a workflow management system. The discrete programs are:

1. `Chain`: Constructs a parent collection for a `subset` and `proc_type` from a pre-defined set of input collections and RUN collections that share the parent collection root.

2. `Associate`: Associates datasets from one or more input collection into an output TAGGED collection.

3. `Ingest`: Search the local image cache for images which match search constraints e.g. on image type (zero, dome flat, or object), on observing night, or band. Ingest images which are downloaded and have passed integrity checks into the Butler repository using the `IngestRawsTask` Python API.

4. `Retry`: Take as input a RUN collection. Search for all datasets matching the pattern `*_log` to find logs for executed tasks. Identify logs that match pre-defined patterns indicating a retry. If a task is marked for retry, move the `*_log` dataset into one with the name `*_log_retry`.

5. `Execute`: Executes a single task or step in a pipeline on a parent chained collection to completion. Constructs a name for a new run collection in the specified format. Generates a quantum graph using the parent chained collection as input and outputting datasets into the new run collection. Skips quanta for which a metadata entry exists, indicating that the dataset was previously produced. Skips datasets for which a log exists, indicating a previous failure. Submits the resulting quantum graph to a workflow management system using the LSP Batch Processing System. Call `Retry` on the resulting RUN collection. Append the RUN to its parent by calling `Chain`. Repeats operation on the parent chained collection until the quantum graph has no tasks to be executed.

6. `Pipeline`: Given a `proc_type`, one or more `subset`, and a sequence of tasks or steps, `Execute` the sequence of steps on the parent collection defined by the `subset` and `proc_type`.

7. `Night`: Enumerates over nights in the survey taken from the local exposure database and matching a user provided pattern. For each night, enumerate over `proc_type` bias, flat, and drp. For each `proc_type` (and in that order), `Ingest` raw image frames from the local cache into the Butler repository, `Associate` raw images into a TAGGED collection, call `Chain` to construct a parent collection for processing, define visits from the input raw files, execute `Pipeline` using a pre-defined sequence of steps for the `proc_type`, automatically certify calibrations if needed.

8. `Coadd Subsets`: Defines subsets for co-addition. Find warps in a set of input collection. Reduce the (tract, patch, band, visit) dimension values of the input warps on the visit dimension to count the number of input warps associated with each (tract, patch, band) tuple. Group (tract, patch, band) tuples into clusters with similar values of the number of input warps and up to a maximum threshold in the total number of input warps contained in the group. `Associate` each group of warps into its own TAGGED collection, constructing a group subset identifier and appending it to the provided coadd subset. Define a parent collection for each group in the coadd subset.

9. `Coadd`: Enumerate over groups defined for a coadd subset. Call `Pipeline` to execute coaddition tasks on the group. `Associate` coadd outputs from coadd subset groups into a TAGGED collection using the coadd subset name.

10. `Diff`: Take as input a coadd subset and enumerate over nights in the survey. For each night, construct a parent collection with the `proc_type` set to diff_drp and specifying the coadd subset to append the correct TAGGED collection containing coadd outputs to use as a template. Call `Pipeline` to execute difference imaging tasks on the parent collection.

In our implementation, these discrete programs are implemented as Python scripts, and we use Parsl to represent and execute a task graph constructed through invocations of the Python programs in a Bash shell. Example command-line invocations of these programs might look like

```
$ night 20190401
```

to ingest data, construct calibrations, and perform single-night processing on science images from observing night 2019-04-01 of the survey. And processing the whole survey in this way looks like

```
$ night .*
```

where `.*` is a regular expression that selects all observing nights available in the local exposure database.

3.4 Optimizing Sky Templates for Slow Moving Object Recovery

Templates of the regions of sky observed are constructed through coaddition of the survey science images. These templates are then subtracted from the same science images to construct difference images, which should only contain signal from transient sources such as moving objects. Templates are constructed at the pixel level by attempting to model a distribution of flux values from the input images. A summary statistic such as the mean is typically used to model this distribution. A transient source in the science image will be an outlier in this distribution and should ideally deviate from the chosen summary statistic value. Moving objects are only transient if they move enough between the epochs of the

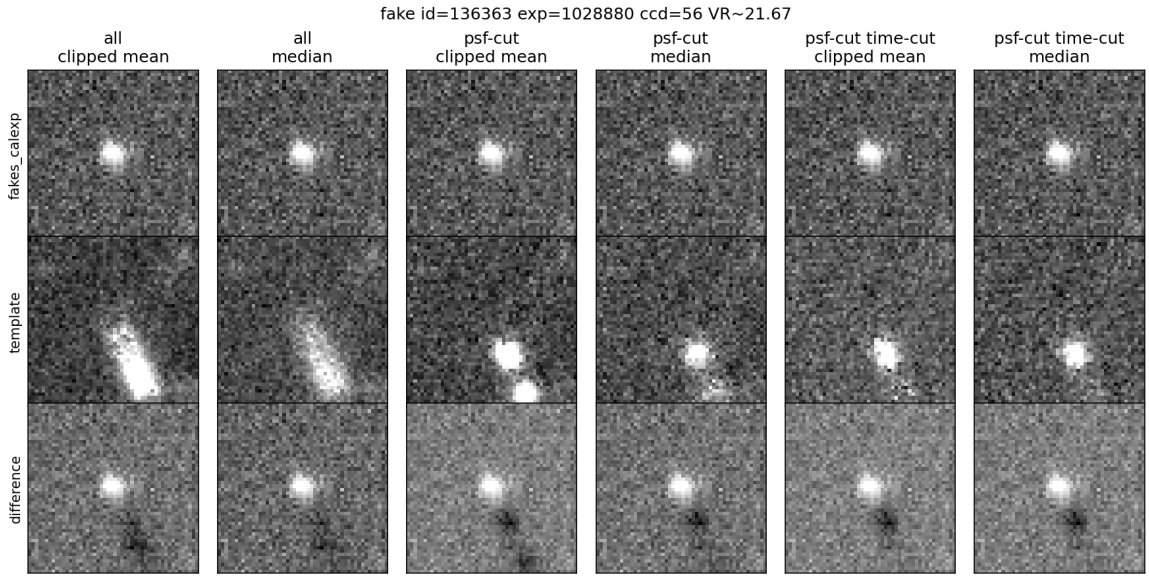


Figure 3.5: An example bright and faint synthetic TNO as it appears in a science image, a co-added template, and a difference image.

science images so as not to bias the input flux distribution for the template. Objects that move slow enough will instead heavily contaminate the input flux distribution and will be fully or partially modeled by the template. In the difference image, flux from the moving object will be lost as it will have been subtracted out–producing the so-called self-subtraction effect. This presents a challenge when attempting to discover slow moving objects, such as Trans-Neptunian Objects, in difference images. Figure 3.5 illustrates this effect for a bright synthetic TNO in the DEEP survey.

A more optimal template can be constructing either by modeling the flux distribution differently or modifying the distribution so that modeling it is easier. The flux distribution can be modeled differently by changing the summary statistic used. In practice, the mean is a poor summary statistic that is easily biased in the presence of outliers. Typical remedies are to use the median or the sigma-clipped mean as an estimate of the center of the input flux distribution. The input distribution can be modified by including observations that do not include the moving object at all to boost the contribution of the static sky to the

flux distribution. This can be challenging or impractical to achieve as it requires modifying the survey observing strategy and sacrificing observing time to re-visit previously observed fields on a different night. A typical pattern of template construction will also sub-sample the input images to select those with the best seeing. This constructs a template which has a narrower PSF than the majority of the science images, which has the effect of improving the quality of the resulting difference images. Sub-sampling the input images also has the result of making the template shallower, which can increase the noise present in the difference images and increase the chance and number of false positive detections when performing a shift-and-stack search on the images.

We performed a set of injection-recovery tests to observe the effect of template construction strategy on the discovery of TNOs using the DEEP data. Our experiment consists of injecting a dense set of synthetic TNOs into the DEEP survey images, constructing a variety of templates using different summary statistics, constructing difference images using those templates, and then applying a shift-and-stack code to recover the synthetic objects in the difference images. A fiducial optimal template is also constructed by using input images without synthetic objects injected into them. The effectiveness of the template strategy is evaluated by modeling the completeness of the shift-and-stack search. We use the three parameter completeness model from Bernardinelli et al. (2024)

$$p(m|c,k,m_{50}) = \frac{c}{1+\exp\left(k(m-m_{50})\right)} \tag{3.1}$$

where m_{50} is the magnitude at 50% completeness, c is the fraction of objects detected when $m \ll m_{50}$, and k is a steepness parameter that encodes how rapid the drop-off in completeness is at m_{50}. The parameter values are estimated by maximizing the likelihood

$$\mathcal{L} = \prod_{i\in\mathcal{D}} p(m_i|c,k,m_{50}) \prod_{i\in\mathcal{D}^c} 1 - p(m_i|c,k,m_{50})$$

where $\mathcal{D}$ are the set of synthetic objects recovered, $\mathcal{D}^c$ is the set of synthetic objects not recovered, and m_i is the magnitude associated with a single synthetic object.

Image processing, template construction, and difference imaging requires substantial computational resources. To keep the computational cost down, we performed our experiment on a small subset of the DEEP survey. We used the tools described in 3.3.2 to process

5 detectors from the 2021-09-04 pointing of the B1e DEEP field. We chose detectors 1, 11, 27, 49, and 56 arbitrarily but with a spread across the DECam focal plane. This pointing contains 98 VR-band exposures which each have exposure times of 120 seconds. A population of synthetic objects was generated using the tools described in Bernardinelli et al. (2024). The density of synthetic objects was increased so that there were approximately 200 objects per detector. Detectors from all observation epochs went through instrument signature removal, nightly calibrations were applied, PSF model fitting, WCS fitting using the Gaia DR3 catalog, and photometric calibration using the PS1 catalog, converting raw images into calibrated exposures. Synthetic objects were injected into the calibrated exposures. The calibrated exposures both with and without the fakes were warped onto a common pixel grid using an identical skymap.

Two modes of epoch selection were applied: one which retained one third of the epochs where the seeing was the best and another which included all epochs. Three summary statistics were used for coaddition: the mean, the sigma-clipped mean, and the median. The sigma clipped mean performs iterative clipping to remove signal that is 5 or more standard deviations from the estimated mean of the input flux distribution. In all, 12 templates were constructed by pairing the three statistics with the two epoch selection strategies and using warped exposures that did and did not contain synthetic objects. Templates constructed with warped exposures that do not contain synthetic objects represent an ideal template, that which does not contain signal from objects we wish to recover. Difference images were produced by subtracting the templates from the calibrated exposures with the synthetic objects.

We used the KBMOD (Whidden et al., 2019) software to apply shift-and-stack to the difference images. For each of the template strategies, we ran KBMOD on stacks of the difference images produced from the chosen template one detector at a time. The searches performed covered velocity rates from 50 to 450 pixels per day and angles with respect to the ecliptic of -45° to 45°. All candidates with a likelihood level of 7 or more were output from KBMOD. Candidate trajectories include a starting location. A search candidate is matched to a synthetic object if its trajectory starting location is within 2 arcseconds of the location of a synthetic object. This partitions the synthetic objects into disjoint sets of

those that were found and not found. Using this information, we fit Eq. 4.10 to produce a completeness curve corresponding to each template type.

Figure 3.6 visualizes the completeness fit for each experiment while Tab. 3.1 enumerates the completeness fit parameters from these experiments. Each of the completeness curves achieves similar m_{50} depth and completeness c, but there is large variation in the steepness parameter k when comparing the completeness derived from the templates that contain synthetic objects and those that do not. We multiply each of the completeness curves with a proposed model of the luminosity function of TNOs, which translates completeness into a model of detection probability of a real object as a function of magnitude. The luminosity function we choose is a rolling power law, modeled and published by Napier et al. (2024)

$$p(m) = 10^{0.89\times(m-23)-0.07\times(m-23)^2} \ .$$

Figure 3.7 visualizes the detection probabilities $p(s|m)p(m)$ for the different template types. This transformation of the completeness curve magnifies the difference in completeness at the faint end, where the number of objects is most numerous. What is observed is a strict ordering in the effectiveness of the template: the clipped mean is better than the median which is better than the mean, and it is better to use all of the epochs than to perform a selection on seeing. This strict ordering is observed again when considering the template constructed without synthetic objects, which is the optimal template possible for reducing the self-subtraction effect.

We can compare template choices numerically by integrating the detection probability to produce an un-normalized count of the number of objects detected:

$$N \propto \int dm \ p(m|s)p(s) \ .$$

The normalization factor would account for factors like the area of sky searched and the size of the total population. However, we can ignore the normalization factor by comparing the un-normalized count for each template experiment to a reference

$$f_{rel} = N/N_{rel} \ .$$

We choose as reference the processing choice in Smotherman et al. (2024), which was to

include all epochs and use the mean statistic. Relative numbers of detections for each of the template choices are enumerated in Table 3.1.

We conclude from these experiments that when constructing templates from survey images, and especially in the case that a single night of data is used to construct the templates that are used to produce difference images that drive discovery, it is more optimal to use all of the epochs available than to perform any epoch selection. Additionally one should rely on the clipped mean as the summary statistic used for the template as opposed to the median or mean. Using the clipped mean is expected to boost the number of detections by 1.31 relative to prior processing in Smotherman et al. (2024). It is more optimal still to construct a template that does not contain the moving objects of interest, accomplished by using out-of-survey images to construct the template or constructing a template for a single night using survey images from different nights. These experiments suggest that this could boost the number of moving objects found by a factor of 2.03.

3.5 Processing Campaign

We utilized the LSST Science Pipelines as well as the tools described in Sec. 3.3.2 to process the DEEP survey images described in Sec. 3.2. The version of the pipelines used was `w_2024_30`.

3.5.1 Pipeline Processing

For each night of the survey, we constructed master calibrations using the available zero and dome flat calibrations from the NOIRLab archive for the observing nights of the survey. The `cpBias.yaml` and `cpFlat.yaml` pipelines were used from the LST `cp_pipe` (calibration pipeline) package. Nights 2019-05-05, 2020-10-19, and 2022-05-25 of the survey were missing raw flat calibrations in the NOIRLab archive, while night 2020-10-19 was missing raw bias calibrations. For these nights, the prior or next night's calibrations were used to calibrate their respective science images. In these pipelines, raw bias frames are combined to form a master bias calibration. Raw flat frames are then bias corrected, normalized, and combined to construct a master flat calibration.

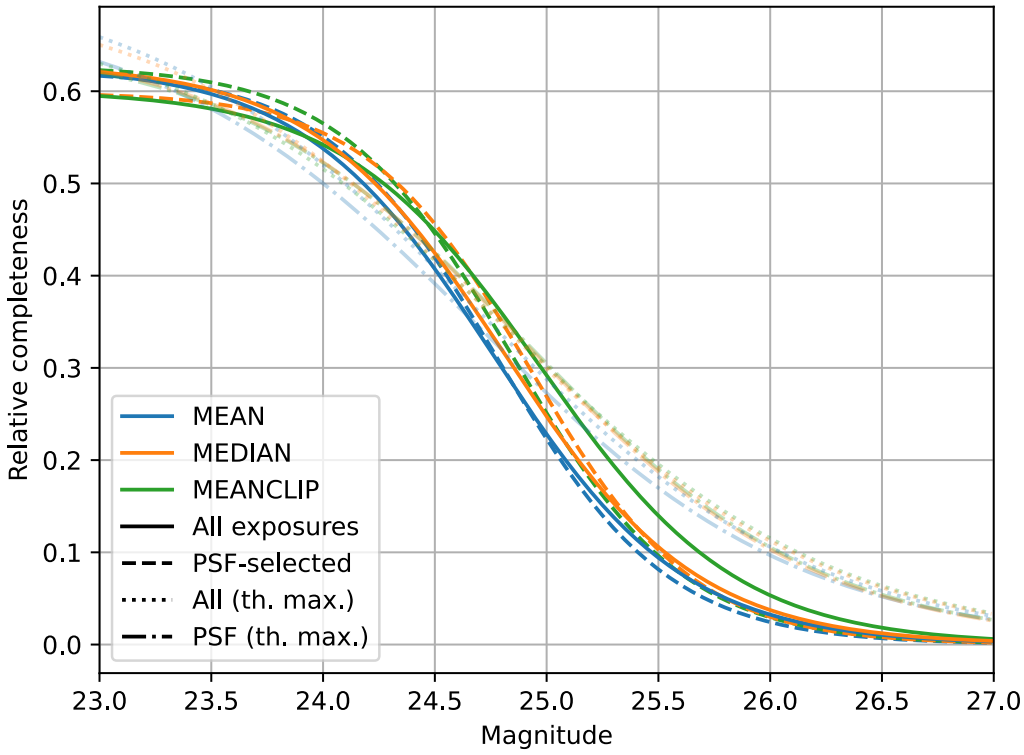


Figure 3.6: The completeness of a shift-and-stack search applied to difference images made using a variety of template construction methods. Colors represent a choice in the coaddition statistic, while a difference in the line styling represents a choice in which exposures were included in the template. Fainter lines indicate that signal from the synthetic objects were not included in the template, representing a "theoretical maximum" recovery efficiency achievable when the self-subtraction effect is removed.

Selection	Statistic	Has Fakes	c	m_{50}	k	f_{rel}
psf	MEAN	withFakes	0.62 ± 0.05	24.78 ± 0.11	2.62 ± 0.37	0.92
all	MEAN	withFakes	0.63 ± 0.05	24.76 ± 0.13	2.35 ± 0.34	1.00
psf	MEANCLIP	withFakes	0.63 ± 0.05	24.84 ± 0.12	2.60 ± 0.38	1.03
psf	MEDIAN	withFakes	0.60 ± 0.04	24.92 ± 0.11	2.74 ± 0.40	1.06
all	MEDIAN	withFakes	0.63 ± 0.05	24.81 ± 0.14	2.33 ± 0.34	1.09
all	MEANCLIP	withFakes	0.60 ± 0.05	24.97 ± 0.14	2.27 ± 0.35	1.31
psf	MEAN	noFakes	0.69 ± 0.11	24.69 ± 0.32	1.39 ± 0.25	1.77
psf	MEDIAN	noFakes	0.66 ± 0.08	24.90 ± 0.25	1.52 ± 0.26	1.88
psf	MEANCLIP	noFakes	0.65 ± 0.08	24.91 ± 0.25	1.51 ± 0.26	1.90
all	MEAN	noFakes	0.73 ± 0.11	24.69 ± 0.31	1.35 ± 0.24	1.91
all	MEDIAN	noFakes	0.71 ± 0.10	24.76 ± 0.29	1.35 ± 0.23	2.00
all	MEANCLIP	noFakes	0.68 ± 0.10	24.83 ± 0.32	1.36 ± 0.26	2.03

Table 3.1: Completeness parameters for Eq. 4.10 for KBMOD searches on difference images constructed using different template types as well as the derived relative number of TNO detections f_{rel}.

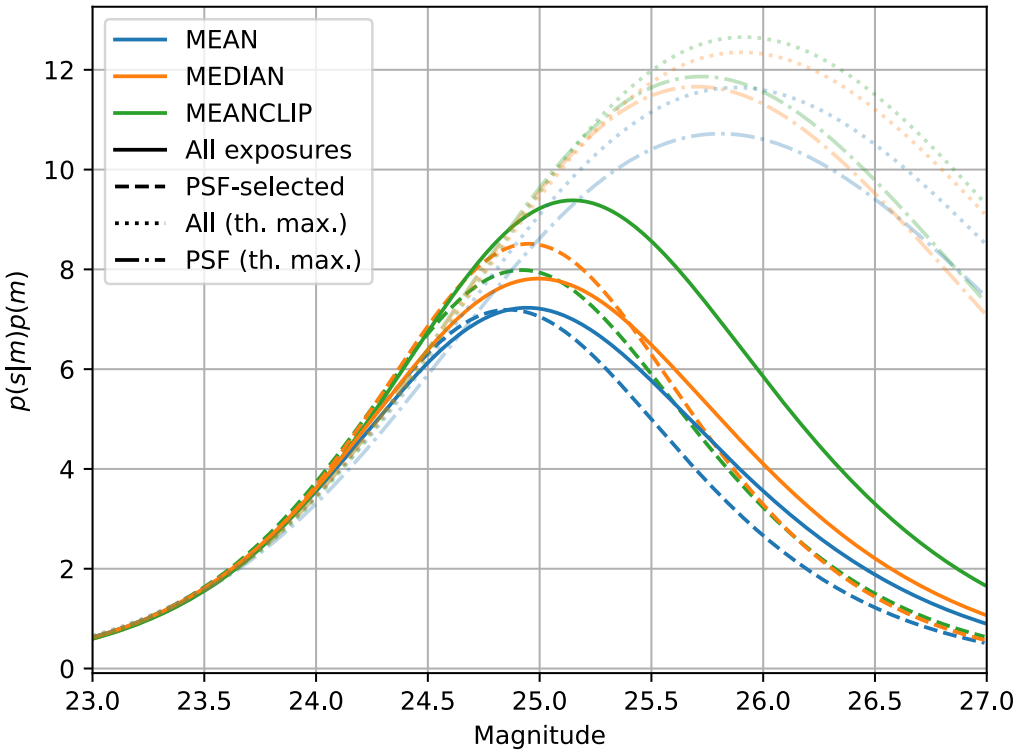


Figure 3.7: The probability of detecting a TNO at a given magnitude, found by multiplying completeness with a luminosity function for TNOs. Colors represent a choice in the coaddition statistic, while a difference in the line styling represents a choice in which exposures were included in the template. Fainter lines indicate that signal from the synthetic objects were not included in the template, representing a "theoretical maximum" recovery efficiency achievable when the self-subtraction effect is removed.

Science images were processed using a modified version of `DRP.yaml` for DECam from the LSP `drp_pipe` (data release pipeline; link) package. At this stage, only minor modifications are made to DRP tasks to accommodate the processing of VR-band images and to split tasks into steps that share common data dimensions. Splitting the tasks this way ensures that a single task failure at one stage does not propagate into cascading downstream failures. A major modification is made in the form of the bad pixel masks applied to the DECam images. It was discovered through this processing effort that the default bad pixel masks applied by the LSP are incorrectly interpreted versions of privately distributed bad pixel masks produced from the Dark Energy Survey (DES; Morganson et al. (2018)) data processing efforts. The existing pipelines incorrectly interpret the DES mask for suspicious pixels (SUSPECT; not to be used in precision analysis) as bad, masking and interpolating over scientifically valuable pixels, mostly at the edge of the detector. The mistake is most apparent when applied to detector 57 (N26), in which a large region of the detector is masked and interpolated over, shown in Figure 3.8. The bad pixel masks were replaced by a new set of masks, ones which combine the published DECam community pipeline bad pixel masks and the DES produced bad pixel masks that are distributed with the LSST Science Pipelines. The SUSPECT mask from the DES bad pixel masks are extracted and applied to input images in a new task called `applySuspect` that has been included in the modified DRP. Figure 3.9 visualizes the result of merging the DECam community pipeline and DES bad pixel masks over a small region of detector 57 (N26), which masks the pixel row the produces the large SUSPECT region that was previously masked out.

For each night of the survey, the modified DRP pipeline was applied to the science images, performing bad pixel masking, instrument signature removal, cosmic ray detection and repair, PSF fitting, background modeling and subtraction, WCS fitting via cross-matching detected sources with Gaia DR3 (Gaia Collaboration et al., 2016b, 2023), photometric calibration via cross-matching with Pan-STARRS1 (Chambers et al., 2019), and source detection. PSF and WCS models are used to inject a catalog of synthetic sources into survey images after instrument signature removal. The process of background modeling and subtraction, PSF fitting, WCS fitting, and source detection are applied again to these images to produce calibrated images and source detection catalogs that include

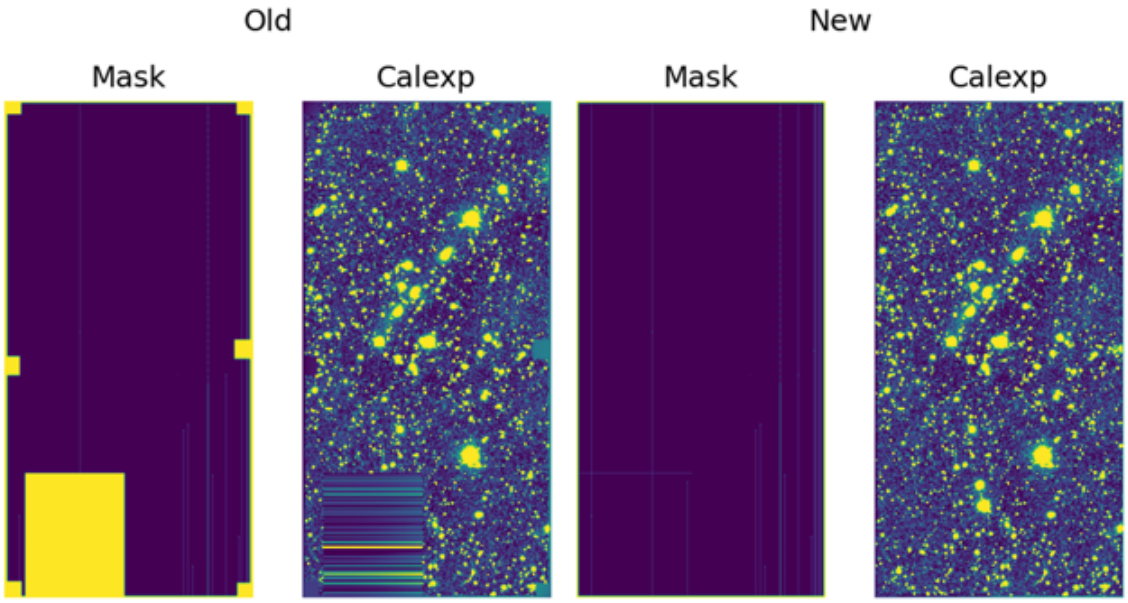


Figure 3.8: The default and updated bad pixel mask for detector 57 (N26) as well as the before/after calibrated exposure that results after application of these mask. When using the updated bad pixel mask, a large region of the detector—in addition to a region at the edge–is no longer masked and interpolated over. In the calibrated exposures, the contrast is heightened to highlight the masked regions.

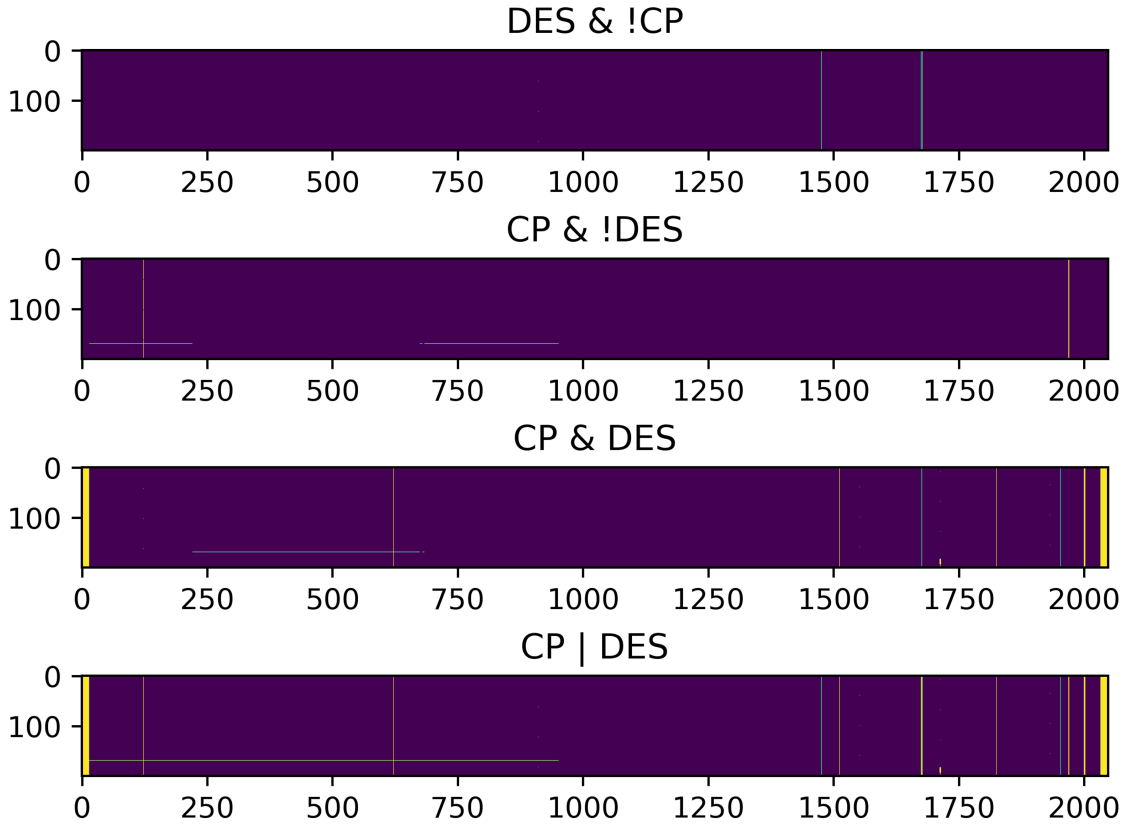


Figure 3.9: The differences between and the result of merging the bad pixel masks produced by the DECam community pipeline (CP) and Dark Energy Survey (DES). Binary operators "!", "&", and "|" stand for "not", "and", and "or". Merging the masks produces a conservative bad pixel mask that can be applied to the DECam images.

synthetic objects. Calibrated exposures with synthetic objects injected are aligned using their WCS and warped onto a common pixel scale. Warped exposures are co-added using the `CompareWarpAssembleCoaddTask`, which compares both direct and PSF-matched warps to exclude transient artifacts in the template, to construct a nightly template for image subtraction. The template is subtracted from the processed science images using the `SubtractImageTask` based on the method of Alard-Lupton (Alard & Lupton, 1998). After all survey processing is done, all warped exposures are co-added using the `CompareWarpAssembleCoaddTask` to construct a single survey template. The survey template is subtracted from the science images using the `SubtractImageTask`. We terminate the DRP processing at this point, excluding downstream tasks that perform detection on the coadd, associate difference image sources into difference image objects, and perform forced photometry on calibrated and difference exposures for the coadd sources and difference image objects. These tasks were excluded as they were not critical for the science goals of the DEEP collaboration and substantially increases the computational resources necessary for pipeline execution.

3.5.2 Processing Errors

Errors were encountered when executing the pipeline tasks during this processing. Transient errors due to node failures or memory errors were retried; however, failures determined to be due to data quality issues or failures with unknown causes were not tracked down and fixed. Detectors 2 (S30), 31 (S7), and 61 (N30) produce persistent errors in processing. Detector 2 (S30) is masked out during data reduction leading to a predicable failure, detector 31 (S7) has one amplifier masked out, and detector 61 (N30) has unmasked data quality issues. Example images for these three detectors are visualized in Fig. 3.10. These detectors can be considered unreliable and should likely be excluded outright.

Among the other detectors, the most major failures occurred during PSF modeling, WCS fitting, and photometric calibration with the most common failure mode being a paucity of quality sources detected on the science images; these are images for which the weather was bad. A shortlist of "bad weather" nights are enumerated in Table 3.2. These nights had

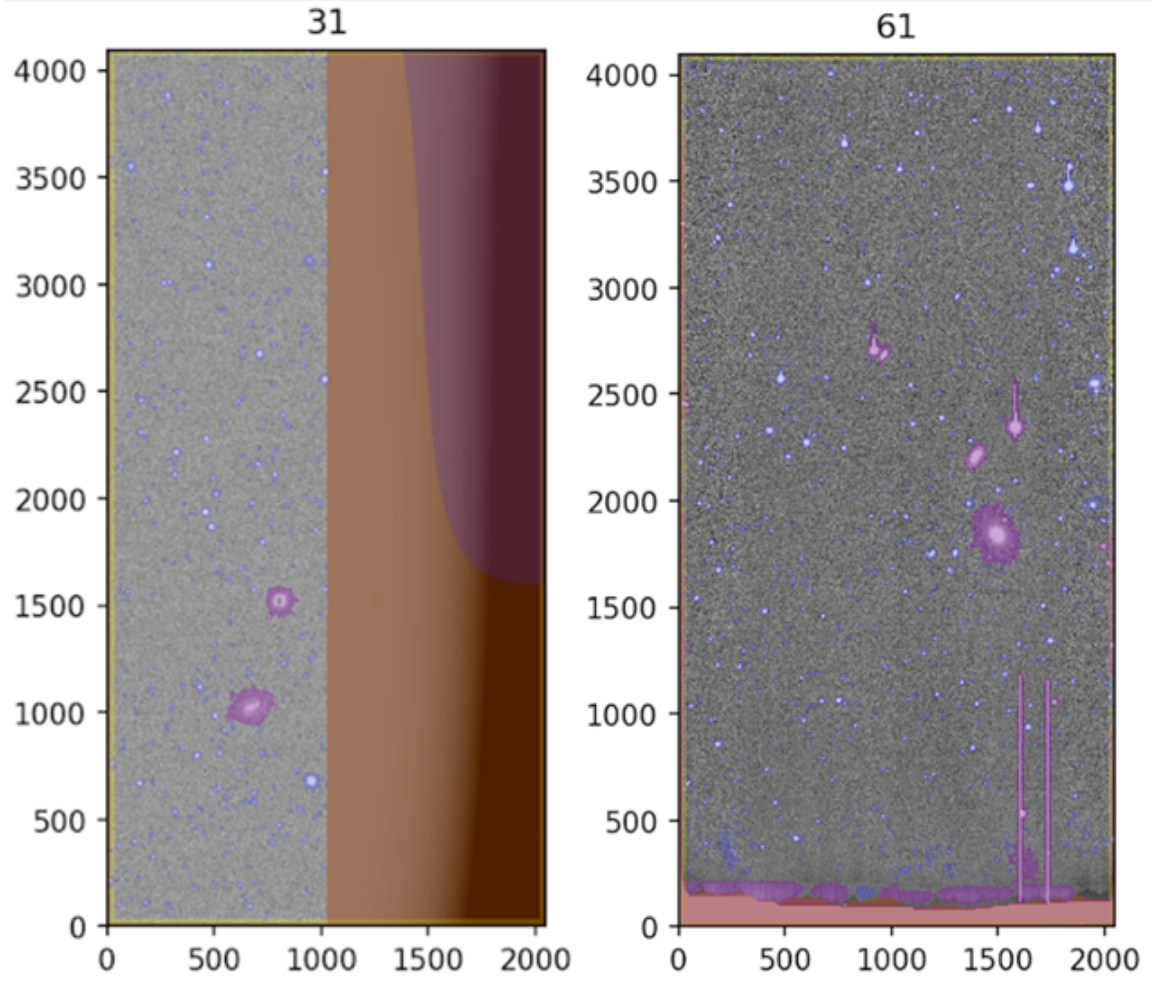


Figure 3.10: Images of calibrated exposures for detector 31 and 61 which have pervasive issues during processing. Pixels are visualized in gray while colors indicate layers of values in the exposure mask.

lost more than 3 of the 62 focal plane detectors on average during at least one stage of the single-night processing through image characterization (the `characterizeImage` task; PSF fitting) and image calibration (the `calibrate` task; background subtraction, WCS fitting, and photometric calibration).

There were two failure modes observed when performing coaddition. In the first, a mismatched number of non-empty direct and PSF-matched warps were provided as input to the co-addition task. It is unclear how this could happen, but it occurred for a small number of the coaddition tasks performed. This points to a likely bug in the LSST Science Pipelines logic that should be fixed. This error was fixed by modifying the inputs to the coaddition task to exclude all visits for which there was an empty warp written to disk. This information is non-trivial to retrieve and involved parsing the log files for the failed coaddition tasks to determine which warps were empty when loaded. A second failure mode was observed where a convex polygon could not be constructed that bounds the input warps.[7] We hypothesized that this was due to errors in the WCS models of the input detectors used for the warps. It was found that these tasks had input warps that included detector 31, one of the detectors with processing issues and which would sometimes produce WCS models that were inaccurate beyond the single unmasked amplifier. These errors were fixed by modifying the inputs to this task to remove visits for which detector 31 was included in the warp. In future processing, detectors 31 and 61 should be excluded from the warping process to avoid these errors and to also avoid unmasked bad data from entering the template.

In difference imaging, the only failure mode observed was again related to matching of the geometries of the difference image and the template arising as a `SinglePolygonException`. These errors were likely again due to bad WCS models, however we did not investigate these errors further.

[7]The error reported was `RuntimeError: Polygon is not convex`

night	icExp	calexp
20190505	0.05	0.06
20190603	0.02	0.07
20201015	0.02	0.05
20210506	**0.19**	**0.48**
20210510	**0.08**	**0.37**
20210512	0.05	0.05
20210513	0.02	0.06
20210515	**0.10**	**0.39**
20210518	**0.19**	**0.47**
20210903	0.02	0.05
20210907	**0.30**	**0.76**
20210910	0.06	0.06
20220528	**1.00**	**1.00**
20220821	0.03	0.06
20220822	0.02	0.05
20220823	0.02	0.08

Table 3.2: The fraction of datasets produced of type postISRCCD, icExp, and calexp for nights that have a relatively large number of processing errors, resulting in at least 3 detectors per focal-plane being lost on average. Nights with more than 5 detectors per focal-plane lost on average are highlighted with boldface text.

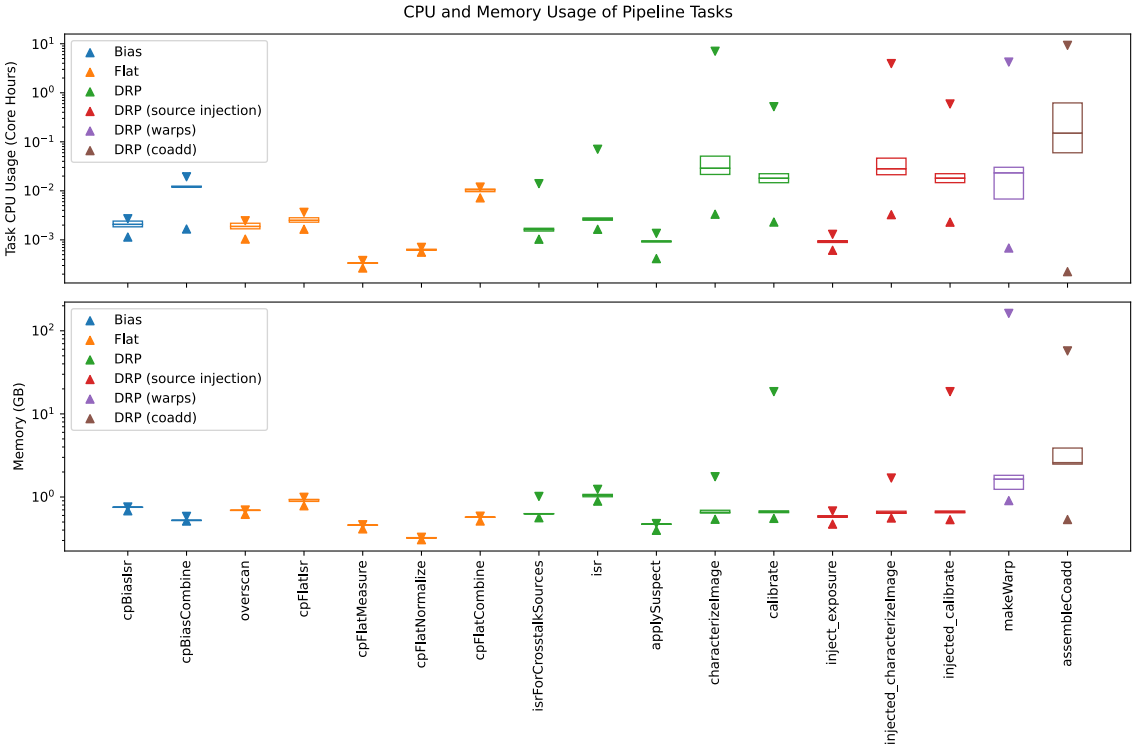


Figure 3.11: The runtime and memory usage grouped by task. Arrows indicate the minimum and maximum value across tasks, while the bottom, middle, and top of the boxes indicate the 25th, 50th, and 75th percentile in values. Colors indicate pipeline or pipeline step(s) that these tasks belong to.

3.5.3 Summary of Computational Cost

The computational cost of this processing campaign was vast, requiring dedicated computing resources and a vast storage array to achieve. The total data volume for processing exceeded 500TB and the runtime exceeded 100,000 core hours. Figure 3.11 visualizes the CPU runtime and memory usage of the tasks executed while Fig. 3.12 visualizes the amount of storage used by the datasets produced by these tasks.

Memory requirements were an issue when constructing the survey template, as it was unknown what the memory requirements would be prior to execution. The memory re-

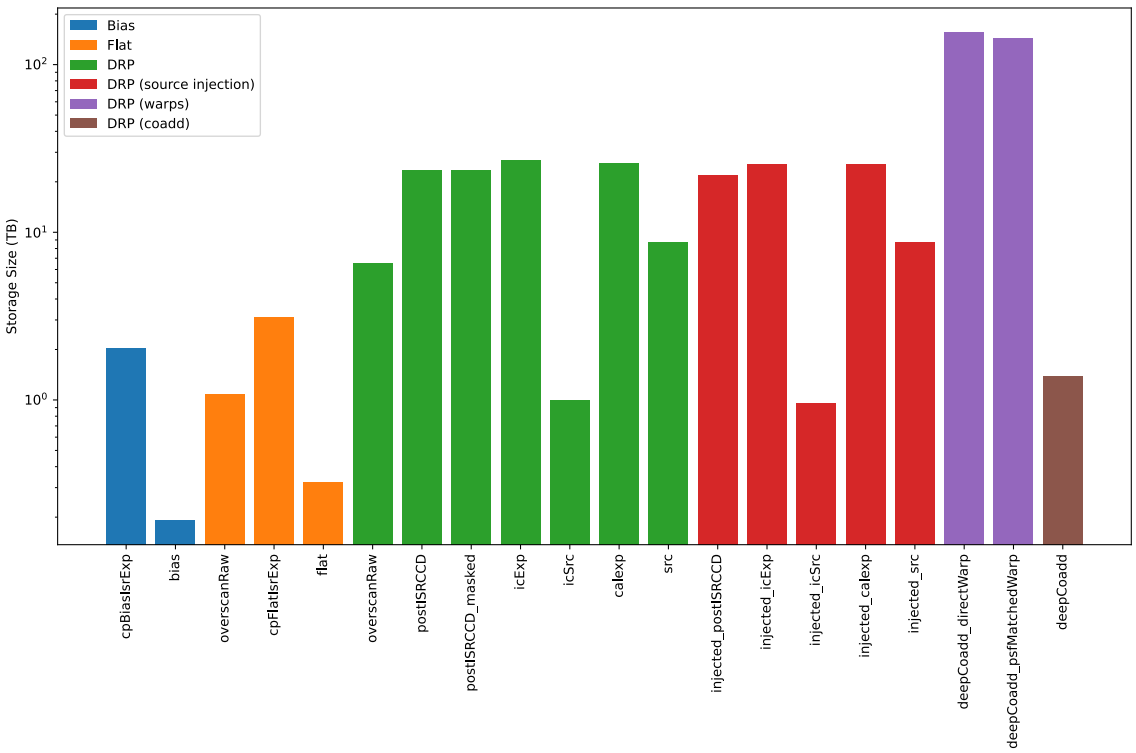


Figure 3.12: The total storage required for the tasks in the pipeline executed. Tasks are colored by the pipeline or pipeline steps they belong to.

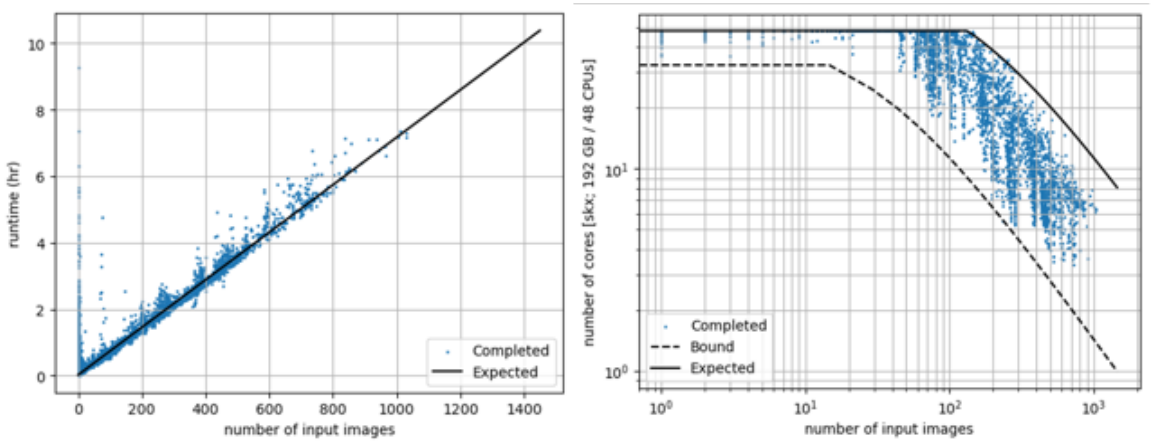


Figure 3.13: The runtime and memory usage of coaddition tasks for constructing the survey template.

quirements can be predicted based on the number of input warps to a coaddition task and the configurable portion of the input images that are loaded into memory to performing the coaddition.[8] However, it was found that memory requirements typically exceeded the predicted memory requirements. When constructing the survey template, we used a progressive prediction strategy where we guessed the amount of memory tasks with few inputs needed and then checked after execution how much they used, providing a more accurate prediction for future runs of tasks with larger numbers of inputs. The large memory requirements limit the single-node parallelism available when executing the tasks, resulting in a ballooning of the node-hour runtime relative to core-hour runtime, which is important to consider when trying to predict the estimated runtime of the workflow or when requesting fixed-size allocations of computing resources from a supercomputing center. Figure 3.13 visualizes the memory usage, runtime, and available parallelism when executing survey template coaddition tasks on a machine with 48 cores on an Intel Xeon Platinum 8160 CPU and 192 GB of RAM as a function of the number of inputs warps.

[8]This is tuned through the `subregionSize` configuration of the `AssembleCoaddTask`.

3.5.4 Effectiveness of All-Sky Coadds

In our processing campaign, we constructed a single all-sky template using all processed survey images, in order to mitigate the self-subtraction effect for slow moving objects and test the predictions of Sec. 3.4. To investigate whether this was an effective mitigation strategy, we measure the fluxes of a subset of synthetic objects injected into the DEEP survey images. The subset we analyze are injected synthetic objects in detectors 1, 3, and 5 from survey night April 1, 2019. For each synthetic object and each observing epoch in this set, we perform forced photometry using the position of the synthetic object and a PSF-photometry model, obtaining an estimate of its flux and magnitude. The PSF-photometry model convolves pixel-level flux and inverse variance values around pixel x_0, y_0:

$$c = \sum_{x,y} \frac{f(x,y)}{\sigma^2(x,y)} p(x,y|x_0,y_0)$$
$$a = \sum_{x,y} \frac{p(x,y)^2}{\sigma^2(x,y)}$$

where the sum is over pixels in a cutout around x_0, y_0 indexed by position x, y, f are flux values σ^2 are variance values, and p is a realization of the PSF model on the image at x_0, y_0. This convolution produces a PSF flux estimate $\hat{f}$, uncertainty σ_f, and SNR:

$$\hat{f} = \frac{c}{a}$$
$$\sigma_f = \frac{1}{\sqrt{a}}$$
$$\text{SNR} = \frac{c}{\sqrt{a}}.$$

Fluxes are zero-point corrected using the equation

$$f^{\text{ref}} = 10^{-2/5\left(z_p - z_p^{\text{ref}}\right)} \times f$$

where z_p is the photometric zero-point from a single-detector photometry model and $z_p^{\text{ref}} = 31$ is a reference zero-point. Magnitudes are then derived from these fluxes using the equation

$$m^{\text{ref}} = -\frac{5}{2} \log_{10}\left(f^{\text{ref}}\right) + z_p^{\text{ref}} \ .$$

Figure 3.15 visualizes the distribution of the difference between the PSF-fluxes found through forced photometry at the synthetic object positions and their expected flux given their simulated injection magnitude. The figure visualizes this distribution for forced photometry from the calibrated exposure and from difference images produced using the survey template and a nightly template. Figure 3.14 visualizes the corresponding magnitude distributions. By inspecting the distribution of fluxes on the calibrated exposures, we find that the flux measurements skew larger than expected. This is likely due to the presence of background objects in the calibrated exposures which contaminate the flux measurements. The flux measurements skew lower than expected for the difference image subtracted with the nightly template. This is the signal of the self-subtraction effect. The flux measurements return to their expected distribution when using the survey template: there is not skew towards more or less than is expected in the difference image. This tells us that the survey template works as expected in reducing the self subtraction effect. When observing the distribution of magnitudes in Fig. 3.14, we note a peak in lost flux at ~ 0.25 mag when using the nightly template, which is recovered fully with the survey template, which is a rough measure of the depth—at least for bright objects—recovered by using the survey template.

Figure 3.16 visualizes a bright (VR ~ 20.98) synthetic TNO in the calibrated exposure and two sets of difference images, in addition to cutouts of the template used to construct the difference images at the synthetic object location. We find that while the injected flux from the slow moving TNO is clearly present in the nightly template, it is vastly reduced–but still faintly present–in the survey template. This visualization indicates that the signal of the bright object in the template can be further reduced through extra mitigation methods, such as a modified form of sigma-clipping.

3.6 Conclusion

In this chapter, I discuss how the LSST Science Pipelines can be used to construct difference images for DECam imaging surveys. I outline a set of tools for performing and managing these data processing efforts in a semi-autonomous and reproducible manner using a light weight "campaign management" tool. I additionally introduce a method for constructing an

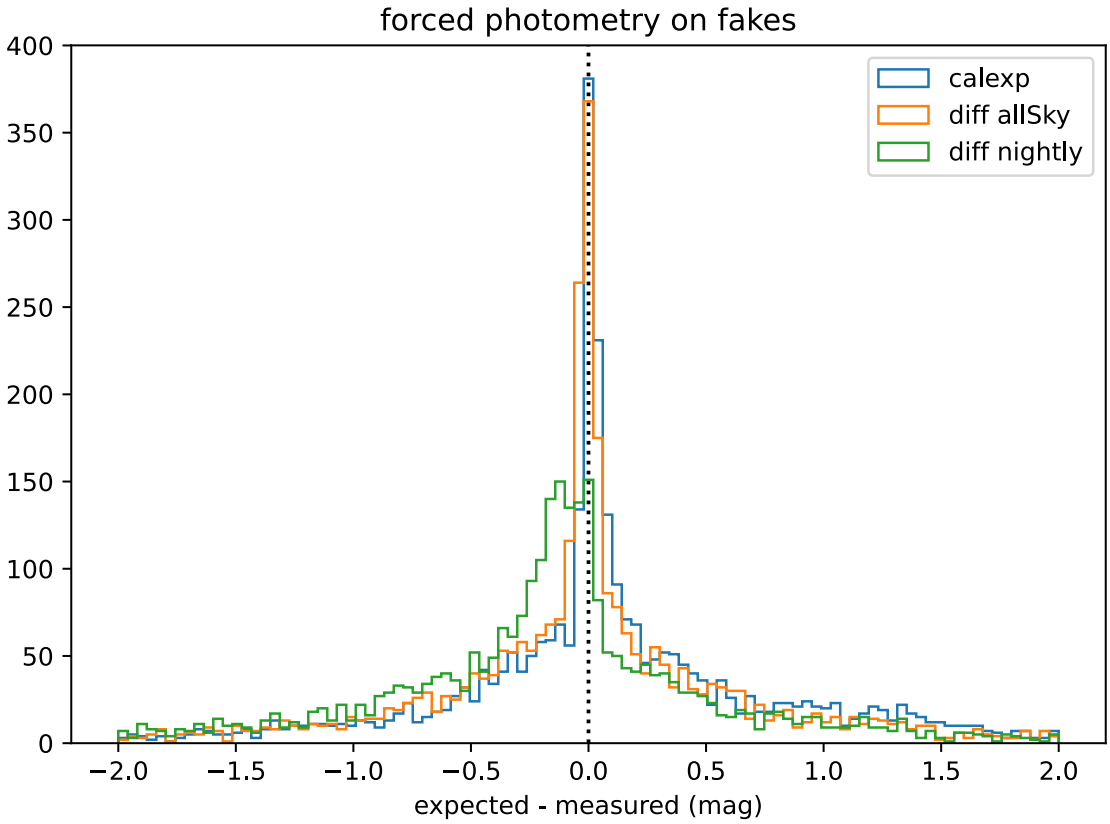


Figure 3.14: The measured magnitudes of a subset of the synthetic moving objects injected into the DEEP survey images. Fluxes are measured on a calibrated exposure (calexp; blue), as well as a difference image (diff) using a survey template (allSky; orange) and nightly (green) template.

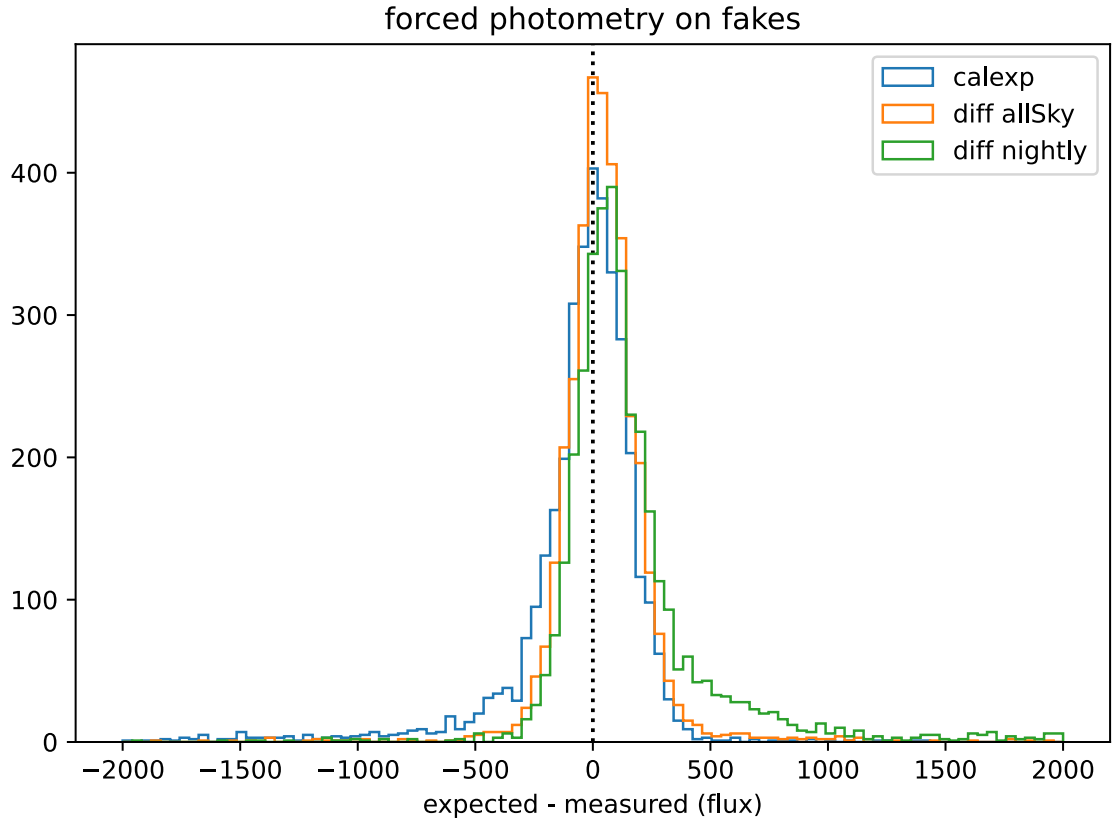


Figure 3.15: The measured fluxes of a subset of the synthetic moving objects injected into the DEEP survey images. Fluxes are measured on a calibrated exposure (calexp; blue), as well as a difference image (diff) using a survey template (allSky; orange) and nightly (gree) template.

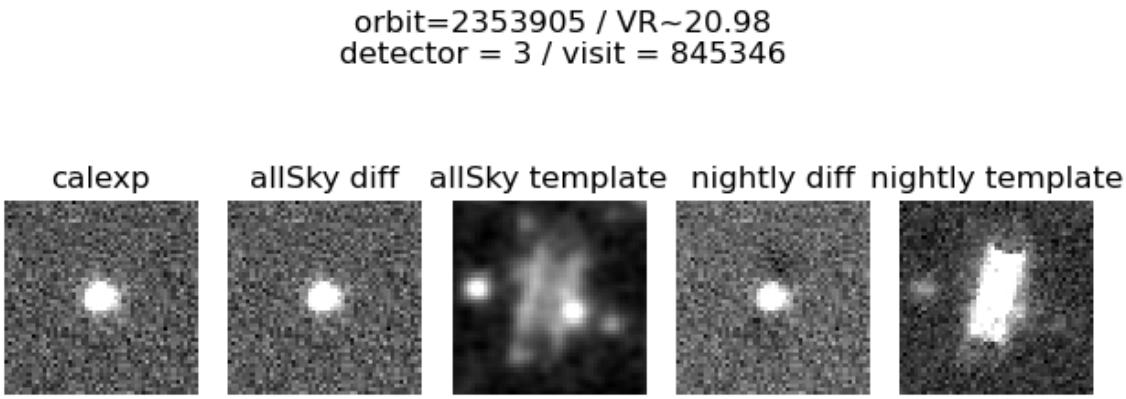


Figure 3.16: A cutout of a single bright (VR $\sim$ 20.98) synthetic TNO injected into the DEEP survey images. Panels visualize from left to right the synthetic TNO in a calibrated exposure (calexp), in a difference image (diff) produced via the survey template (allSky), the cutout of the template at the given location of the object, and the difference and template constructed using a single night's data. In the nightly template difference image, darkened "wings" of lost flux are present in the image while in the nightly template, a large and bright streak appears due to flux from the moving object. In the survey template difference image, the "wings" vanish while in the template the flux from the moving object appears as a faint streak relative to the already faint sources around it.

optimal template for recovering slow moving objects in difference images. My tests indicate that using all survey images and the clipped mean as coaddition statistic is optimal for recovering slow moving objects. Using this strategy is expected to approximately double the number of detected TNOs in the DEEP survey. Finally, I summarize an effort to process all of the images from the DEEP survey using the outlined campaign management tools. In addition to improvement in the survey template construction methodology, I introduce improvements to the masking of bad pixels in the survey images. These efforts will maximize the TNO discovery potential for the DEEP survey.

Chapter 4

AN EFFICIENT SHIFT-AND-STACK ALGORITHM APPLIED TO DETECTION CATALOGS

4.1 Introduction

Discovering new solar system objects helps constrain existing and form new theories about the formation and evolution of our solar system. The boundary of discovery lies in the observation and recovery of the faintest solar system objects, probing into smaller size regimes of the different populations of solar system small bodies: near-earth objects, main belt asteroids, Jupiter and Neptune Trojan populations, Centaurs, Trans-Neptunian objects, and comets. Discovering and characterizing the smallest bodies in these populations helps constrain their size distribution, supporting or refuting different theories of accretion and planetary formation.

All telescope surveys are limited in their ability to find faint objects from single-epoch imaging data, due to the practice of reporting detections that appear at a signal-to-noise ratio (SNR) of 5 or more to avoid contaminating detection catalogs with noise. Objects that may be too faint to detect confidently in single-epoch imaging can be recovered through coaddition of images across epochs (Zackay & Ofek, 2017).

Shift-and-stack, or digital tracking, is an algorithm used to discover faint solar system objects in multi-epoch imaging of the night sky through coaddition. In the shift-and-stack algorithm, pixels from a set of multi-epoch images are first aligned along a candidate trajectory of a solar system object, typically tracking on-sky linear motion, and then coadded together. If the candidate trajectory aligns with a real object, signal from each image will accumulate coherently in coaddition, allowing a high-confidence detection of the object in a coadded "stack" even if the object appears at low signal to noise in any individual image. In stacks of 100 images, this algorithm can discover objects 10× fainter in flux (2.5 magnitudes) than would otherwise be possible, making it a powerful driver for discovery

of faint solar system objects. Shift-and-stack has been broadly applied to drive discovery of Trans-Neptunian objects (Bernstein et al., 2004b), main belt asteroids (Heinze et al., 2015b), and near-Earth objects (Shao et al., 2014b).

The main barrier to applying shift-and-stack is its enormous computational expense: the number of pixel stacks required for a complete search can easily exceed 10^{10} for small scale searches and take hours or days of compute time. Speedup can be achieved through improved parallelism, for example application of shift-and-stack on GPUs (Whidden et al., 2019; Smotherman et al., 2021), or through reducing the complexity of the algorithm, for example by re-using values in the stacking process (Nguyen et al., 2024).

Our approach to speedup is to stack detection catalogs, which are necessarily sparser than the images they are derived from. We utilize the work of Budavári et al. (2017) which showed that it is possible to stack detection catalogs and recover faint objects by analyzing the relative Bayesian evidence that a set of detections are generated from an astrophysical object or a noise process. We combine this approach with and build upon the work of Dalitz et al. (2017) which outlined a method to find lines in three-dimensional point cloud data through a Hough transform algorithm. The Hough transform represents a class of template matching algorithms most often used in the field of computer vision for identifying geometric features in two-dimensional digital images. The transform was first proposed for identifying linear tracks in images of bubble chamber experiments (Hough, 1959) and later extended to identify lines and curves (Duda & Hart, 1972) as well as generalized shapes (Ballard, 1981) in digital images. The central feature of these algorithms is an accumulator array whose entries correspond to the likelihood that a given pattern with a certain parameterization exists in an input signal. In Dalitz et al. (2017), the pattern is a line in three-dimensional space and the input signal are (x, y, z) coordinates of point cloud data. We modify this method to search for two-dimensional lines in Right Ascension (α) and Declination (δ) indexed by time t as an independent variable, operating on a point cloud of $(t, \alpha(t), \delta(t))$ coordinates. This modification behaves similarly to the Hough transform approach of THOR, in which clusters are identified in the residuals between the predicted and observed positions of moving objects using hypothesized orbits (Moeyens et al., 2021). The result is an efficient shift-and-stack algorithm that performs $10^3 - 10^5$ fewer effective stacks and achieves wall-clock speedups of

$10-10^3$ relative to an image based shift-and-stack while retaining the ability to find moving objects that are within the noise of an individual exposure.

Section 4.2 outlines our algorithm and its approach to shift-and-stack. The algorithm is validated by applying it to an imaging dataset from the Dark Energy Camera (DECam; Flaugher et al. (2015)), outlined in section 4.3. We find that the algorithm achieves the promised speedup and is able to recover moving objects fainter than the SNR cutoff used to construct the detection catalog. Additionally, a reference implementation is provided via GitHub.[1]

4.2 Efficient And Exhaustive Stacking of Detection Catalogs

Our algorithm applies the shift-and-stack technique to detection catalogs: lists of the observation times (t) and on-sky positions i.e. right ascensions (R.A. / α) and declinations (Dec. / δ) of observed astronomical sources. The rate of motion and curvature of the on-sky position of a solar system object is dependent on the object's orbit, and typically most strongly dependent on its distance from the Earth: objects near the earth tend to move faster on the sky while those farther away move more slowly. All solar system objects appear to exhibit linear motion on the sky over certain timescales. Heinze et al. (2015b) notes that this time is 1-2 hours for near earth objects, 8 hours for main belt asteroids, and more than 24 hours for trans-Neptunian objects. While shift-and-stack can be applied exhaustively to track a space of orbits, an exhaustive search over on-sky linear motion will track all orbits sufficiently over certain timescales. This algorithm focuses on this regime: searching for linear motion in the on-sky positions of sources in the detection catalog.

In a locally-tangent plane approximation of the celestial sphere, linear on-sky motion is modeled using the equations

$$\alpha(t) = (t - t_0)v_\alpha + \alpha_0$$
$$\delta(t) = (t - t_0)v_\delta + \delta_0$$

where v_α is the on-sky velocity in RA, v_δ is the on-sky velocity in Dec., and α_0, δ_0 are the RA and Dec. at a reference epoch i.e. when $t = t_0$.

[1] https://github.com/dirac-institute/salad

If one makes a guess at the motion using a hypothesis velocity v_α^t, v_δ^t, the residuals of the sky position are

$$\begin{aligned}
\alpha(t) &= (t - t_0)v_\alpha + \alpha_0 \\
\alpha^t(t) &= (t - t_0)v_\alpha^t + \alpha_0^t \\
\Delta\alpha = \alpha(t) - \alpha^t(t) &= (t - t_0)(v_\alpha - v_\alpha^t) + (\alpha_0 - \alpha_0^t) \\
\delta(t) &= (t - t_0)v_\delta + \delta_0 \\
\delta^t(t) &= (t - t_0)v_\delta^t + \delta_0^t \\
\Delta\delta = \delta(t) - \delta^t(t) &= (t - t_0)(v_\delta - v_\delta^t) + (\delta_0 - \delta_0^t) \ .
\end{aligned}$$

From this equation, and considering just right ascension α (the same results will hold for δ), we note that the magnitude of the residuals $\Delta\alpha$ scales linearly with the difference between the observation time t and the reference epoch t_0 and with the difference between the trial velocity v_α^t and the true velocity v_α^*. For a given moving object that has been observed multiple times and with a fixed hypothesis v_α^t, the residuals are maximized at the time farthest from the reference epoch

$$\max\left[\Delta\alpha\right] = \max\left[t - t_0\right](v_\alpha^* - v_\alpha^t) + (\alpha_0^* - \alpha_0^t)$$

and minimized at the time closest to the reference epoch

$$\min\left[\Delta\alpha\right] = \min\left[t - t_0\right](v_\alpha^* - v_\alpha^t) + (\alpha_0^* - \alpha_0^t) \ .$$

This implies that the residuals among all of the observations of a single object are bounded in a range Δx

$$\begin{aligned}
\Delta x &= \max\left[\Delta\alpha\right] - \min\left[\Delta\alpha\right] \\
&= \max\left[t - t_0\right](v_\alpha^* - v_\alpha^t) + (\alpha_0^* - \alpha_0^t) \\
&\quad - \min\left[t - t_0\right](v_\alpha^* - v_\alpha^t) - (\alpha_0^* - \alpha_0^t) \\
&= \left(\max\left[t\right] - \min\left[t\right]\right)\left(v_\alpha^* - v_\alpha^t\right) \\
\Delta x &= \Delta t \Delta v_\alpha
\end{aligned}$$

where Δt is the time difference between the first and last observation of the object and Δv_α is the difference between the true and hypothesized velocity of the object. This result implies that the positional residuals of a given object are bounded within some radius Δx. Δx will be large when the velocity hypothesis is far from the truth, say for stationary objects, and small when the velocity hypothesis is close to the truth, say for moving objects. If instead of fixing the hypothesis v_α^t and observing how Δx changes we choose to fix the value of Δx and vary the hypothesis, we find that an object's residuals will stay in a cluster of size Δx when

$$\Delta v_\alpha \leq \frac{\Delta x}{\Delta t} \ .$$

The same relationship is found when analyzing the residuals in declination

$$\Delta v_\delta \leq \frac{\Delta x}{\Delta t} \ .$$

These results show that a cluster finding algorithm can be applied to the space of projected positions using the clustering radius Δx to recover moving objects with true velocities within $\Delta v_\alpha, \Delta v_\delta$ of a hypothesis v_α^t, v_δ^t. These results also provide a method to construct the set of velocities that must be hypothesized to perform a complete blind search for a target population of moving objects: simply test all on-sky velocities between v_{min} and v_{max} which differ in their α and δ components by no more than Δv_α and Δv_δ. For a given value of Δx, this results in a set of velocities $\mathcal{V}$ of size

$$\begin{aligned} |\mathcal{V}| &= \frac{v_{\mathrm{max},\alpha} - v_{\mathrm{min},\alpha}}{\Delta v_\alpha} \times \frac{v_{\mathrm{max},\delta} - v_{\mathrm{min},\delta}}{\Delta v_\delta} \\ &= \left(\frac{\Delta t \Delta v}{\Delta x} \right)^2 \end{aligned} \tag{4.1}$$

where $\Delta v = v_{min} - v_{max}$ is assumed to be the same for α and δ. The complete blind search can be thought of as exploring the set of velocity hypotheses that bound the residuals of objects from the target population within a threshold Δx.

Once a cluster is found in the projected space, its exact velocity is not known except that it is plausibly within $\Delta x / \Delta t$ of the initial hypothesis velocity. The true velocity of the object can be recovered by repeating the blind search procedure on the clustered data alone and with Δx set to a smaller value, perhaps a value that would be reflective of the

expected positional residuals given the object's expected brightness. However, if the cluster is relatively pure, meaning it mostly contains detections from a single moving object, a data-driven approach will suffice in which case the true velocity of the object can be recovered via multivariate regression. The multivariate regression problem with N observations is cast as

$$Y = B^T X + c + \epsilon$$

where $Y \in \mathbb{R}^{N \times q}$ are the response variables, $X \in \mathbb{R}^{N \times p}$ are the independent variables, $B \in \mathbb{R}^{p \times q}$ is the slope matrix, $c \in \mathbb{R}^q$ is the intercept vector and $\epsilon \sim \mathcal{N}(0, \Sigma_\epsilon)$ is a random variable representing errors in observations and is modeled as a q-dimensional normal distribution with zero-mean. For clarity, we can translate this into the problem at hand of estimating α and δ as a function of time t:

$$\begin{bmatrix} \alpha_1 & \delta_1 \\ \cdots & \cdots \\ \alpha_N & \delta_N \end{bmatrix} = \begin{bmatrix} v_\alpha \\ v_\delta \end{bmatrix}^T \begin{bmatrix} t_1 \\ \cdots \\ t_N \end{bmatrix} + \begin{bmatrix} \alpha_0 & \delta_0 \end{bmatrix} + \epsilon \ .$$

The slope, intercept, and covariance of the errors are estimated from the mean and covariance matrices of the response and independent variables. If we partition the mean μ and covariance Σ according to X and Y

$$\mu = \begin{bmatrix} \mu_Y \\ \mu_X \end{bmatrix} \Sigma = \begin{bmatrix} \Sigma_{XX} & \Sigma_{XY} \\ \Sigma_{YX} & \Sigma_{YY} \end{bmatrix}$$

then the slope matrix is estimated from the sample mean $\hat{\mu}$ and sample covariance matrix $\hat{\Sigma}$ as

$$B = \hat{\Sigma}_{XX}^{-1} \hat{\Sigma}_{XY}$$

while the intercept vector is

$$c = \hat{\mu}_Y - B^T \hat{\mu}_X$$

and the covariance of the errors is

$$\hat{\Sigma}_\epsilon = \hat{\Sigma}_{YY} - B^T \hat{\Sigma}_{XX} B \ .$$

Once the velocity of the moving object is found via regression, the object can be considered recovered.

The sample mean and covariance matrix are sensitive to the inclusion of outliers. If the cluster contains noise observation or contaminating detections from artifacts, then the recovered trajectory can be biased away from the truth. Multivariate regression can still be used if made robust to the inclusion of noise. In the prior formulation, a robust estimator of the mean and covariance of the data will provide a robust estimate of the regression parameters. The Minimum Covariance Determinant (MCD; Rousseeuw (1985); Peter J Rousseeuw & Gulló (2004)) provides such a method for robust determination of the mean and covariance of a dataset. This method enumerates over subsets of the dataset of size $n/2 \leq h < n$ to find a subset whose covariance determinant is minimized. The mean and covariance of this subset are reported as robust estimates of the sample mean and sample covariance. The subset with the minimum covariance determinant achieves the minimal scatter about a central point i.e. this subset excludes outliers that may be present in the full dataset.

When setting $h = n/2$, this method will be robust up to 50% noise contamination in the dataset, achieving the best possible break down point of any method. Rousseeuw & Driessen (1999) introduces the FAST-MCD, which enables fast iteration over subsets of large datasets to find either the exact MCD or an approximation of it. An implementation of FAST-MCD is provided in `scikit-learn`, enabling its use in Python. A Python implementation of the FAST-MCD for robust multivariate regression is provided by us.[2] Figure 4.1 visualize how the technique of robust multivariate regression can be used to recover a moving object's trajectory in the presence of noise.

4.2.1 The Algorithm

Based on these observations, our algorithm follows, which is in essence the iterative Hough transform algorithm presented by Dalitz et al. (2017). Take as input a detection catalog

$$\mathcal{C} = \begin{bmatrix} \alpha_i & \delta_i & t_i \end{bmatrix}_{i=1}^{n}$$

[2] https://github.com/stevenstetzler/robust_linear_regression

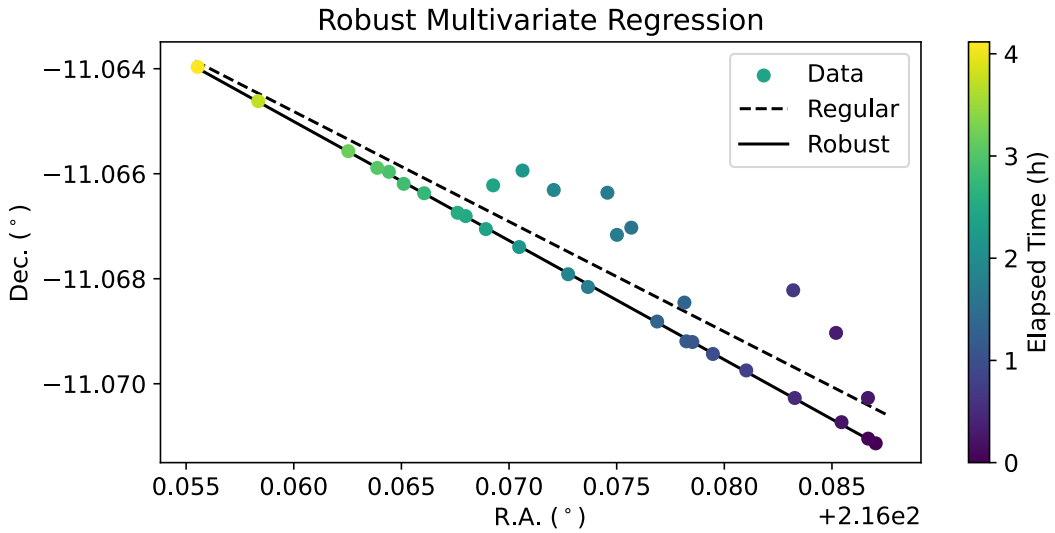


Figure 4.1: The on-sky coordinates of a moving object with respect to time. The dashed black line represents a trajectory estimated using least squares multivariate regression, which does not predict the trajectory accurately due to the presence of outliers in the dataset. The solid black line represents a trajectory estimated using robust multivariate regression utilizing the Minimum Covariance Determinant (MCD), which accurately predicts the moving object trajectory in the presence of outliers.

a set of hypothesis velocities

$$\mathcal{V} = \left[v_\alpha^j \quad v_\delta^j \right]_{j=1}^{m}$$

a clustering radius Δx and a minimum value for the number of detections we expect from each moving object $N_{\min}$.

1. Project: First, project the detection catalog to the reference epoch, which we choose to be the minimum of all of the observation times contained in the detection catalog.

2. Cluster: Next, for each hypothesis velocity, construct a two dimensional histogram of the projected positions using a resolution of Δx. Consider the set of histograms jointly as a three dimensional histogram.

3. Find clusters: In a while loop, identify the location of the maximum of the histogram. If the maximum is $< N_{\rm min}$, break to the next stage. Otherwise, identify the points in the detection catalog that contributed to that maximum. These points form a cluster. Remove those points from the histogram. Continue until the loop is broken.

4. Refine: For each cluster produced, perform multivariate regression with the MCD to fit a linear trajectory to the points recovered.

5. Gather: Gather points from the original detection catalog within Δx of the refined line.

6. Optional: Repeat steps (4) and (5) until convergence i.e. the set of gathered points does not change in each iteration.

4.2.2 Setting the algorithm parameters

It remains to be addressed how one should set the parameters of the algorithm: Δx and $N_{\rm min}$. We will address this point in this section. Both are related to the expected purity of the clusters found by the algorithm. If the input detection catalog is already pure with no false positive detections, the value of Δx should be set based on the sky density of astronomical sources in the catalog. If Δx is set too large, then one will expect significant contamination in the space of projected positions from stationary sources as well as other moving objects. If the input catalog is constructed at low-SNR, meaning some (or even most) of the detections in the catalog are spurious, then the value of Δx should be set to control the purity of the resulting cluster in the algorithm. A larger Δx will accumulate more noise detections while a smaller Δx will accumulate fewer noise detections in a single cluster.

The value of Δx can be determined by reasoning about the number of noise peaks $N_{\rm noise}$ and real object peaks $N_{\rm signal}$ expected to land in a bin of size Δx^2. The best value to choose for Δx maximizes the ratio $\mathbb{E}[N_{\rm signal}/N_{\rm noise}]$, which is akin to a likelihood ratio. If $N_{\rm signal}/N_{\rm noise} > 1$, then the true moving object can be distinguished from the noise peaks

included. If $N_{\rm signal}/N_{\rm noise} > 1$, then the contamination rate $N_{\rm noise}/(N_{\rm signal} + N_{\rm noise}) < 0.5$ and we expect the MCD multivariate regression technique to recover the object.

Assuming the value of Δx is chosen to maximize signal relative to noise, then the value of $N_{\rm min}$ determines the number of false positives produced by the algorithm. It can be chosen so that

$$P(N_{\rm noise} \geq N_{\rm min}) \leq \epsilon$$

where ϵ represents an admissible false positive rate of detected objects. Under the algorithm considered, a reasonable value of ϵ is

$$\epsilon = \frac{N_{\rm false}}{N_{\rm stacks}}$$

where $N_{\rm false}$ is the maximum number of false results produced by the algorithm and $N_{\rm stacks}$ is the number of independent stacks performed. If we assume the trial velocities and stack bins are independent, then the number of stacks is

$$\begin{aligned} N_{\rm stacks} &= |\mathcal{V}| \frac{\Omega}{\Delta x^2} \\ &= \frac{\Delta t^2 \Delta v^2 \Omega}{\Delta x^4} \end{aligned} \tag{4.2}$$

using Eq. 4.1 to substitute in $|\mathcal{V}|$ and where Ω is the solid angle of sky searched. What remains is to specify the distributions of $N_{\rm noise}$ and $N_{\rm signal}$.

The distribution of noise peaks has been considered in detail by Budavári et al. (2017) which specifies the spatial distribution of noise peaks as a function of the detection statistic. Assuming the detection process uses a Gaussian window function (i.e. PSF detection) and the sky noise is white, the surface density of noise peaks produced by the detection process is (using Budavári et al. (2017) Eq. 40 and 46)

$$n_{\rm pk} = \frac{\exp(-z^2/2)}{2\pi^2 a^2} \left(\frac{\sqrt{2}}{4} z + B\left(\frac{\sqrt{2}}{2} z, 1 \right) \left[\frac{1}{2} + \frac{3}{4} z \right] + B\left(\frac{\sqrt{2}}{2} z, 2 \right) \right)$$

where z is the value of the detection statistic, a is the PSF size, and

$$B(s,b) \equiv \frac{\pi}{b} \exp\left(\frac{s^2}{2b} \right) \left[1 + \mathrm{erf}\left(\frac{s}{\sqrt{2b}} \right) \right] .$$

The cumulative number of peaks above threshold ν_{th} is then

$$\lambda(\nu_{th}) = \int_{\nu_{th}}^{\infty} n_{\rm pk}(z)dz \ . \tag{4.3}$$

The spatial distribution of peaks is a Poisson point process, meaning the number of noise detections $N_{\rm noise}$ is Poisson distributed

$$N_{\rm noise} \sim {\rm Pois}\left(\Lambda\right)$$

with a rate Λ equal to

$$\Lambda = \int_A \lambda(\nu_{th})dA$$

where A represents a spatial region of interest. The probability of having $N_{\rm noise} = k$ noise peaks in a square region with area Δx^2 (in units of the PSF size a) is

$$\begin{aligned} P(N_{\rm noise} = k) &= \frac{\Lambda^k e^{-\Lambda}}{k!} \\ &= \frac{(\lambda(\nu_{th})\Delta x^2)^k e^{-\lambda(\nu_{th})\Delta x^2}}{k!} \ . \end{aligned}$$

If we consider multiple epochs i with different values of the PSF size a_i, the number of noise peaks above a threshold ν_{th} within a stacked bin of size $(\Delta x/a_i)^2$ is

$$\begin{aligned} N_{\rm noise} &\sim \sum_i {\rm Pois}(\Lambda_i) \\ &\sim {\rm Pois}\left(\sum_i \Lambda_i\right) \\ &\sim {\rm Pois}\left(\sum_i \lambda(\nu_{th})(\Delta x/a_i)^2\right) \ . \end{aligned}$$

The number of real object detections $N_{\rm signal}$ depends on the probability that an object is observed with an above-threshold flux and the probability that its observed position is within the spatial region of interest. The observed flux of a source is normally distributed

$$\hat{f} \sim \mathcal{N}\left(f, \sigma_f^2\right)$$

where $\hat{f}$ and f are the observed and true fluxes of an object, and σ_f is the associated uncertainty with the flux measurement. The object is detected above a threshold $f_{\rm th}$ with

probability

$$P(\hat{f} \geq f_{\rm th}|f) = \frac{1}{2}{\rm erfc}\left(\frac{f_{\rm th} - f}{\sigma\sqrt{2}}\right) \ .$$

Expressed in term of the signal to noise $\nu = f/\sigma_f$ we find

$$P(\hat{\nu} \geq \nu_{\rm th}|\nu) = \frac{1}{2}{\rm erfc}\left(\frac{\nu_{\rm th} - \nu}{\sqrt{2}}\right) \ .$$

The observed position x of the object at a single epoch is also normally distributed

$$\hat{\alpha} \sim \mathcal{N}\left(\alpha, \sigma_\alpha^2\right)$$
$$\hat{\delta} \sim \mathcal{N}\left(\delta, \sigma_\delta^2\right)$$

where σ_α and σ_δ are the positional uncertainties. The positional uncertainties depend on the PSF size a and the signal to noise of the object ν. Portillo et al. (2020) provides a derivation of the positional uncertainties (Eq. 14):

$$\begin{aligned}\sigma_\alpha^2 = \sigma_\delta^2 &= 2a^2\left(\frac{f^2}{\sigma_f^2}\right)^{-1} \\ &= 2\frac{a^2}{\nu^2}\end{aligned}$$

where ν is the object signal to noise. The probability that an object is observed within Δx of the true position α, δ is then

$$\begin{aligned}P(|\hat{\alpha} - \alpha| \leq \Delta x/2) &= 1 - 2P(\hat{\alpha} - \alpha \leq -\Delta x/2) \\ &= {\rm erf}\left(\frac{\Delta x}{\sigma_\alpha 2\sqrt{2}}\right).\end{aligned}$$

Substituting the relationship between σ_α and signal to noise ν we find

$$P(|\hat{\alpha} - \alpha| \leq \Delta x/2) = {\rm erf}\left(\frac{\Delta x \nu}{4a}\right) \ .$$

The probability that both α and δ are within Δx is then

$$P(|\hat{\alpha} - \alpha| \leq \Delta x/2, |\hat{\delta} - \delta| \leq \Delta x/2) = {\rm erf}\left(\frac{\Delta x \nu}{4a}\right)^2 \ .$$

If we assume the trial velocity v_α^t, v_δ^t exactly matches the true moving object trajectory, such that $\alpha(t) = \alpha^t(t)$ and $\delta(t) = \delta^t(t)$, then the probability of the detection from a single epoch landing in a bin of size Δx^2 is

$$\begin{aligned} &P(\hat{\nu}_i \geq \nu_{\text{th}}, |\alpha^t(t) - \hat{\alpha}_i| \leq \Delta x, |\delta^t(t) - \hat{\delta}_i| \leq \Delta x) \\ &= \frac{1}{2}\text{erfc}\left(\frac{\nu_{\text{th}} - \nu}{\sqrt{2}}\right)\text{erf}\left(\frac{\Delta x \nu}{4a}\right)^2 . \end{aligned}$$

Across many epochs, this probability is repeated with a per-epoch PSF a_i. N_{signal} is the number of times that this outcome occurs and is the result of several Bernoulli trials with per-epoch rates. N_{signal} is then Poisson Binomial distributed. Making the simplifying assumption that the per-epoch PSF is constant and equal to a, then N_{signal} follows a Binomial distribution:

$$P(N_{\text{signal}} = k) = \prod_{i \in \mathcal{D}} p_i \prod_{i \in \mathcal{D}^c} (1 - p_i)$$

where p is the probability of detection in a single epoch and $\mathcal{D}$ ($\mathcal{D}^c$) is the set of epochs the object is (not) detected in.

The distributions of N_{signal} and N_{noise} depend on the SNR of the object, the SNR threshold used to derive the catalog, and the PSF width at each epoch. Figure 4.2 provides a visual guide for choosing reasonable values of Δx and N_{min} based on the expected values of N_{signal} and N_{noise}. Each panel in the figure plots the expected number of counts $\mathbb{E}[N_{\text{signal}}] + \mathbb{E}[N_{\text{noise}}]$ as a function of the aperture size Δx for objects of varying SNR ν found in a detection catalog of varying SNR thresholds ν_{th}. For each value of ν, the plot indicates when $\mathbb{E}[N_{\text{signal}}]/\mathbb{E}[N_{\text{noise}}]$ (used as an approximation of the harder to compute $\mathbb{E}[N_{\text{signal}}/N_{\text{noise}}]$) is greater than or less than 1, indicating that an object is recoverable or not using the MCD regression technique. Additionally, a predicted value for N_{min} that is expected to produce $100/N_{\text{stacks}}$ false detection candidates for an MBA and TNO search is overplotted, representing the "noise floor" for a given search; setting N_{min} lower than these values can recover fainter objects at the expense of increasing numbers of false positives. Notably, this figure illustrates that not all objects of a given SNR ν are recoverable over all aperture sizes (or any in the case of $\nu = 0.5$). This is due to the scaling of the positional uncertainties

with SNR, which is expected to diverge as $\nu \to 0$; at some or all aperture sizes, the rate with which noise detections are accumulated surpasses the real object detection rate.

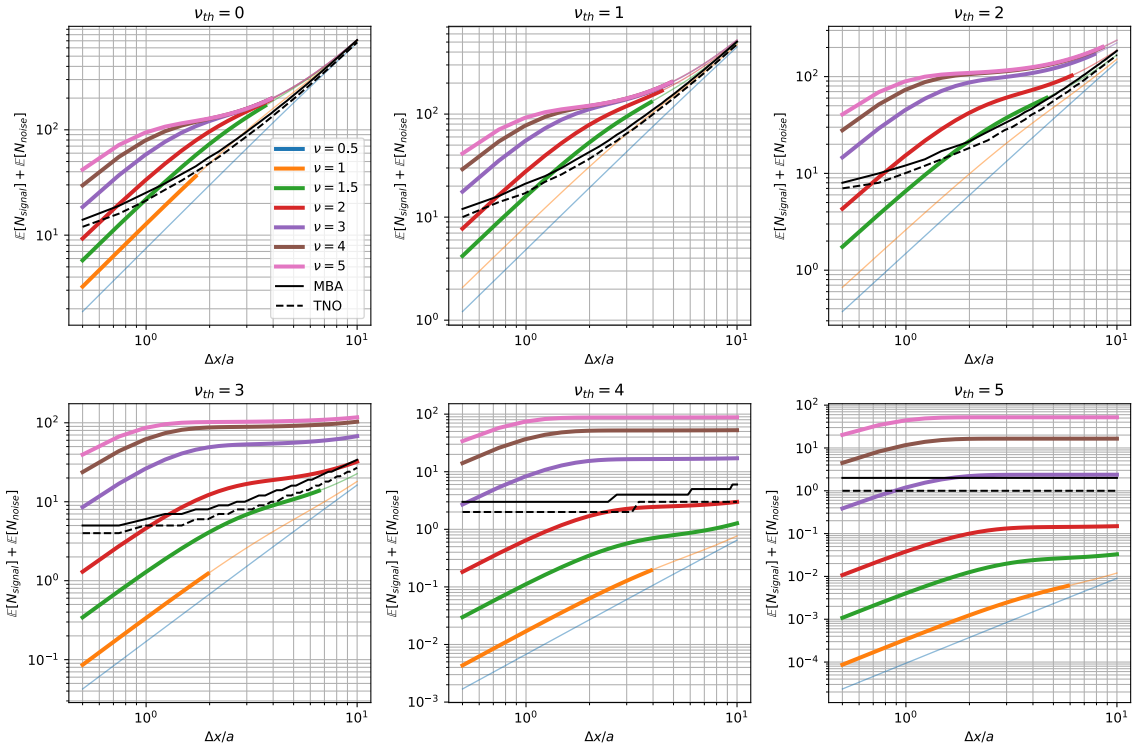


Figure 4.2: The expected number of detection counts in a bin of size Δx for objects of different SNR ν found in catalogs of different SNR thresholds $\nu_{\rm th}$. Thick (thin) colored lines approximately indicate when the expected number of signal detections will be larger (less) than the expected number of noise detections i.e. $\mathbb{E}[N_{\rm signal}]/\mathbb{E}[N_{\rm noise}]$. Black solid (dashed) lines indicate values of $N_{\rm min}$ that produce a false positive rate of $\epsilon = 100/N_{\rm stacks}$ in an example MBA (TNO) search covering $\Omega = 162$ arcmin^2 with $a = 1$ arcsec seeing and a $\Delta t = 4$ hour time baseline.

4.2.3 Algorithmic Scaling

An image-based shift-and-stack algorithm has algorithmic complexity in number of operations that scales as

$$\mathcal{O}(N_{\mathrm{t}} \times N_{\mathrm{pixels}} \times |\mathcal{V}|)$$

or

$$\mathcal{O}\left(N_{\mathrm{t}} \times \frac{\Omega}{p_s^2} \times |\mathcal{V}|\right) \tag{4.4}$$

where N_{t} is the number of epochs/number of images stacked, N_{pixels} is the number of pixels per image, $|\mathcal{V}|$ is the size of the velocity set searched, Ω is the solid angle subtended by a single image, and p_s is the image pixel scale. This is due to the fact that two operations (shift + add) are performed on every pixel in the set of provided input images for each direction searched. The memory usage of this algorithm scales as

$$\mathcal{O}(N_{\mathrm{t}} \times N_{\mathrm{pixels}}) \tag{4.5}$$

since all images need to be stored in memory during the procedure.

In our algorithm, N_{pixels} is replaced with N_{peaks} where N_{peaks} is the number of detection peaks per image. Assuming the detections are dominated by noise, then N_{peaks} is computed relative to N_{pixels} using Eq. 4.3 as

$$\begin{aligned} N_{\mathrm{peaks}} &= \lambda(\nu_{th}) \frac{N_{\mathrm{pixels}}}{a_p^2} \\ &= \lambda(\nu_{th}) \frac{N_{\mathrm{pixels}} p_s^2}{a^2} \\ &= \lambda(\nu_{th}) \frac{\Omega}{a^2} \end{aligned}$$

where Ω is the solid angle subtended by a single image, p_s is the pixel scale, a_p is the PSF-width in pixels and a is the PSF-width in angular units. The process of source detection scales as

$$\mathcal{O}\left(N_{\mathrm{t}} \times N_{\mathrm{pixels}}\right)$$

in operations and

$$\mathcal{O}\left(N_{\text{pixels}}\right)$$

in memory assuming the detection process is applied sequentially to the input images. The complexity, accounting for steps 1 (Project) and 2 (Cluster) of our algorithm, is then

$$\mathcal{O}\left(N_{\text{t}} \times \lambda(\nu_{th})\frac{\Omega}{a^2} \times |\mathcal{V}|\right)$$

in operations, while the memory scaling is

$$\mathcal{O}\left(N_{\text{t}} \times \lambda(\nu_{th})\frac{\Omega}{a^2} \times |\mathcal{V}| + \frac{\Omega}{\Delta x^2} \times |\mathcal{V}|\right)$$

since the projected positions of the detection catalog are stored in memory in addition to the Hough space. Step 3 (Find Clusters) scales in operations as

$$\mathcal{O}\left(N_{\text{results}} \times \frac{\Omega}{\Delta x^2} \times |\mathcal{V}|\right)$$

since it involves finding a maximum entry in the Hough space of this same size. The total complexity of the catalog-based shift-and-stack approach is then

$$\mathcal{O}\left(N_{\text{t}} \times N_{\text{pixels}} + N_{\text{t}} \times \lambda(\nu_{th})\frac{\Omega}{a^2} \times |\mathcal{V}| + N_{\text{results}} \times \frac{\Omega}{\Delta x^2} \times |\mathcal{V}|\right) \tag{4.6}$$

in operations and

$$\mathcal{O}\left(N_{\text{pixels}} + N_{\text{t}} \times \lambda(\nu_{th})\frac{\Omega}{a^2} \times |\mathcal{V}| + \frac{\Omega}{\Delta x^2} \times |\mathcal{V}|\right) \tag{4.7}$$

in memory accounting for all steps.

Taking only the terms in Eqns. 4.4, 4.5, 4.6, and 4.7 which scale with the velocity set $|\mathcal{V}|$ and excluding result finding (i.e. $N_{\text{results}} = 0$), we can calculate the speedup that our algorithm (2) attains relative to an image-based approach (1) in the shift-and-stack procedure itself:

$$\begin{aligned} s &= \frac{N_{\text{t}} \times \frac{\Omega}{p_s^2} \times |\mathcal{V}_1|}{N_{\text{t}} \times \lambda(\nu_{th})\frac{\Omega}{a^2} \times |\mathcal{V}_2|} \\ &= \frac{a^2 \times |\mathcal{V}_1|}{\lambda(\nu_{th}) p_s^2 \times |\mathcal{V}_2|} \\ &= \frac{a_p^2 \times |\mathcal{V}_1|}{\lambda(\nu_{th}) \times |\mathcal{V}_2|} \end{aligned}$$

while the relative memory usage is

$$r = \frac{N_{\rm t} \times \lambda(\nu_{th}) \frac{\Omega}{a^2} \times |\mathcal{V}_2| + \frac{\Omega}{\Delta x_2^2} \times |\mathcal{V}_2|}{N_{\rm t} \times \frac{\Omega}{p_s^2}}$$
$$= |\mathcal{V}_2| \left(\lambda(\nu_{th}) \frac{p_s^2}{a^2} + \frac{p_s^2}{\Delta x_2^2} \right) \ .$$

Assuming the same velocity range and time span is searched, and the two searches differ by Δx, then using Eq. 4.1, the speedup and relative memory usage are

$$s = \frac{a_p^2}{\lambda(\nu_{th})} \left(\frac{\Delta x_2}{\Delta x_1} \right)^2 \tag{4.8}$$

$$r = \left(\frac{\Delta v \Delta t}{\Delta x_2} \right)^2 \left(\lambda(\nu_{th}) \frac{p_s^2}{a^2} + \frac{p_s^2}{\Delta x_2^2} \right) \ . \tag{4.9}$$

Our algorithm is efficient in the sense that it performs fewer effective stacks relative to an image-based search, due to the scaling of speedup with the value of Δx^2. Assuming $a_p = 3$ pixels, $\Delta x_1 = 1a$ (1 PSF width) for an image-based search and $\Delta x_2 = 5a$ (5 PSF widths) for the catalog-based search, then at $\nu_{th} = 0$, the speedup attained is $\approx 4 \times 10^3$. As implemented, the greatest weakness of our algorithm is in its memory usage. Since all search directions are considered jointly and the projected positions of all detections are stored at once, the memory footprint can quickly outgrow the capacity of most machines at low-SNR and low values of Δx.

4.3 Application

In this section, we describe searches for faint moving objects in images from the Dark Energy Camera (DECam). We perform searches for both slow-moving Trans Neptunian Objects (TNOs) and faster-moving Main Belt Asteroids (MBAs). We analyze the performance of the algorithm as a function of the SNR of the catalog used in terms of its ability to recover implanted fake objects and the computation time and memory needed to perform the search.

4.3.1 Data Used

The data used are a single night of data from the DECam Ecliptic Exploration Project (DEEP; Trilling et al. (2024)). The DEEP observing strategy involves tiling the sky around

the invariable plane of the solar system using a long-stare observing strategy across multiple years (2019-2022). Each night of the survey comprises of one or more long-stare pointing—sequences of 120 second VR-band exposures taken with DECam over a period of 2-4 hours. The data we test our algorithm with are selected from the third night of the survey—April 3, 2019–and the long-stare observation of one field of the survey—A0c—which includes 104 120-second VR-band exposures.[3]

Data are processed using the LSST Science Pipelines (LSP; Bosch et al. (2018, 2019)) version `w_2024_09`.[4] Raw science images, VR-band dome flat field images, and bias images are downloaded from the NOIRLab archive.[5] Bias and flat field images are processed through the LSP calibration product pipeline (`cp_pipe`) for DECam. The output of this pipeline are per-detector nightly stacked flat and bias master calibration files which are applied to the science images.

The science images are processed using the LSP data release product pipeline (`drp_pipe`). Each detector is processed individually including defect masking, application of the master calibrations, cosmic ray detection and repair, PSF model fitting to bright stars, background subtraction, source detection, fitting of a world coordinate system (WCS) by crossmatching detected sources to the Gaia DR3 (Gaia Collaboration et al., 2016b, 2023) catalog, and photometric zero point fitting by crossmatching detected sources to the Pan-STARRS1 catalog (Chambers et al., 2019). The photometric zero point is fit using r-band measurements from Pan-STARRS1, neglecting a color term correction for the VR-band images. The VR-band overlaps well with the r-band, meaning the resulting photometry will be approximately correct.

Synthetic moving objects are injected into the images after WCS/PSF/photometric zero point fitting (so that accurate astrometric and photometric models are available) to test the recovery efficiency of this algorithm. The synthetic objects are drawn from two populations that model the existing population of main belt asteroids and trans-Neptunian objects. The

[3] Trilling et al. (2024) outlines the nomenclature used for field naming: A refers to the A semester, A0 to a patch which tracks the year-to-year motion of TNOs, and A0c to a single field observed in that patch.

[4] The LSST Science Pipelines are available freely at `pipelines.lsst.io`.

[5] `https://astroarchive.noirlab.edu/`

distribution of on-sky velocities and their injected magnitudes are visualized in Figure 4.3.

Detectors that were successfully processed through the stage of PSF and WCS fitting are warped/resampled onto an identical pixel grid and coadded using the LSP `CompareWarpAssembleCoaddTask`, which compares the result of direct and PSF-matched warps to construct the coadd. This coadd acts as a template which is subtracted from the processed science images using the LSP `SubtractImageTask` based on the method of Alard-Lupton (Alard & Lupton, 1998). The result are difference images in which ideally only non-static astronomical sources remain: galaxies and non-variable stars are removed whereas variable stars and moving objects remain. The image subtraction is typically not perfect, especially when the WCS solutions are not perfect, leaving behind poorly subtracted stars. The cores of bright stars are typically also not modeled well in the template and remain in the single epoch images after subtraction. These artifacts can be a source of contamination in detection catalogs derived from difference images. The difference images are the inputs to the algorithm as described.

4.3.2 Source Detection

We perform source detection using the LSP `SourceDetectionTask`. This task uses a PSF detection model to determine the location of putative sources. An input image—pixel-level flux values and their estimated variance—are convolved with the PSF. A ratio of the convolved flux and variance is computed to form an image where each pixel represents the SNR of a putative detection at that pixel location. This SNR-image is thresholded at a provided SNR cutoff to form "footprints" or regions of pixels above-threshold. Peaks in SNR are found in the footprints and reported as a putative source detection, including its pixel location and SNR. Properties of the source such as the brightness, shape, and location (at a sub-pixel level) can be further refined using model fitting.

Figure 4.4 visualizes the empirical number of peaks as a function of SNR threshold for a real difference image from DECam. The number of peaks is far fewer than the number of pixels in the image across all values of the SNR threshold. Detection peaks are associated with a sky-location RA α and Dec δ via a world coordinate system (WCS) fit to the exposure

as well as an observation time t. For moving objects, the observation time is set to the midpoint of the exposure, including the time to operate the shutter. Figure 4.5 visualizes the sky coordinates of a SNR ≥ 5 detection catalog derived from a sequence of 104 DECam exposures limited to a single detector. Moving objects appear clearly as lines in this stacked visualization of the input detection catalogs. The goal of our algorithm is to recover these lines.

4.3.3 Search

We used our catalog shift-and-stack algorithm to search for TNOs and MBAs in the processed difference images. To search for TNOs, we used a velocity range of $0.003 - 0.04$ deg/day ($0.45 - 6$ arcsec/hour) and for MBAs a velocity range of $0.1 - 0.5$ deg/day. Test trajectories were generated using these velocity ranges and covering all on-sky angles. These rates cover the span of angular velocities of synthetic sources injected into the images. We search for objects in each DECam detector in the single long-stare pointing. Of the 62 detectors in the DECam focal plane, we exclude detector 2 since it is masked entirely in the data reduction process and detector 61 since it has data quality issues. We searched for moving objects in detection catalogs constructed using different SNR thresholds. The value of Δx was varied among the MBA and TNO searches for each SNR threshold used. Table 4.1 enumerates the values of Δx used for each search. Δx was limited to 10 and the SNR of the detection catalog to 3 for the MBA search due the excessive computational cost involved. For each search performed, the first 1000 clusters with the highest number of detections are output. Each cluster is passed through the MCD regression step, resulting in a refined estimate of the candidate object's trajectory. For each refined trajectory guess, detections are gathered from the per-epoch detection catalogs that are within a positional distance threshold of Δx.

4.3.4 Performance

The performance of this algorithm is validated by its ability to recover the injected synthetic objects i.e. the completeness of the search. We measure completeness as a function of

	Δx (a)	
ν	TNO	MBA
1	1	-
2	1	-
3	3	10
4	10	10
5	10	10

Table 4.1: Values of Δx in units of the PSF width a for each search performed. A value of - indicates the search was not performed for the combination of ν and population.

magnitude using a parametrized model as in Bernardinelli et al. (2024)

$$p(m|c,k,m_{50}) = \frac{c}{1+\exp\left(k(m-m_{50})\right)} \tag{4.10}$$

where m_{50} is the magnitude at 50% completeness, c is the fraction of objects detected when $m \ll m_{50}$, and k is a steepness parameter that encodes how rapid the drop-off in completeness is at m_{50}. The parameter values are estimated by maximizing the likelihood

$$\mathcal{L} = \prod_{i\in\mathcal{D}} p(m_i|c,k,m_{50}) \prod_{i\in\mathcal{D}^c} \left(1-p(m_i|c,k,m_{50})\right)$$

where $\mathcal{D}$ are the set of synthetic objects recovered, $\mathcal{D}^c$ is the set of synthetic objects not recovered, and m_i is the magnitude associated with a single synthetic object.

Search candidates are matched to synthetic objects by comparing their trajectories. Trajectories are matched by comparing the differences in the positions predicted by the trajectories at each epoch. Two trajectories are considered matched if their predicted positions differ by less than 1 arcsec in 50% or more of the epochs. A synthetic object is considered recovered if its trajectory is successfully matched to a candidate trajectory from our algorithm.

A single-epoch SNR $\sim$ 5 limiting magnitude is estimated by matching the catalog of synthetic objects to detected sources in a SNR $\geq$ 5 detection catalog using a 1 arcsec matching radius. For each synthetic object injected, the fraction of times it appears in the

SNR ≥ 5 detection catalog is measured. This fraction follows the same statistics as the survey completeness, allowing us to model it as a function of magnitude using Eq. 4.10. This produces a single-epoch SNR ~ 5 limiting magnitude of $m_{50} = 23.7$. A theoretical upper limit on the m_{50} depth achievable through coaddition can be found by considering the relative flux of a single-epoch detection at SNR ν and a coadded detection at the same SNR ν:

$$\begin{aligned} \nu &= \frac{f}{\sigma} = \frac{f_{\text{coadd}}}{\sigma_{\text{coadd}}} \\ &\rightarrow \frac{f}{f_{\text{coadd}}} = \frac{\sigma}{\sigma_{\text{coadd}}} \ . \end{aligned}$$

If we assume constant background noise and image quality, the uncertainty in the coadd scales as

$$\begin{aligned} \sigma_{\text{coadd}} &= \sqrt{\frac{\sigma^2}{N_t}} \\ \sqrt{N_t}\sigma_{\text{coadd}} &= \sigma \end{aligned}$$

which lets us compute the relative single-epoch and coadded flux as

$$\frac{f}{f_{\text{coadd}}} = \frac{\sqrt{N_t}\sigma_{\text{coadd}}}{\sigma_{\text{coadd}}} = \sqrt{N_t}$$

and is equivalent to a magnitude difference of

$$\begin{aligned} m_{50}^{\text{coadd}} - m_{50} &= \left(-\frac{5}{2}\log_{10}(f_{\text{coadd}}\right) + z_p) - \left(-\frac{5}{2}\log_{10}(f) + z_p\right) \\ &= \frac{5}{2}\log_{10}\left(\frac{f}{f_{\text{coadd}}}\right) \\ &= \frac{5}{2}\log_{10}(\sqrt{N_t}) \end{aligned}$$

where z_p is a shared photometric zero-point. Given $N_t = 104$, we find that $m_{50}^{\text{coadd}} = m_{50} + \frac{5}{2}\log_{10}(\sqrt{N_t}) = 26.2$.

Figure 4.6 visualizes the completeness of the TNO and MBA searches performed after matching the 1000 candidate trajectories per-search to synthetic object trajectories. The solid black link in this figure represents the estimated single-epoch SNR ≥ 5 limiting magnitude of $m_{50} = 23.7$ while the dashed black line represent the theoretical achievable depth of an optimal coadd at $m_{50}^{\text{coadd}} = 26.2$.

SNR	TNO m_{50}	MBA m_{50}
5	23.96 ± 0.06	24.19 ± 0.05
4	24.29 ± 0.05	24.41 ± 0.07
3	24.68 ± 0.04	24.84 ± 0.06
2	25.21 ± 0.05	-
1	25.87 ± 0.21	-

Table 4.2: The m_{50} depth achieved in the TNO and MBA searches of a catalog of the given SNR. A value of - indicates the search was not performed.

The completeness varies as a function of the parameter N_{min}. Greater m_{50} depth can be achieved by decreasing the value of N_{min} at the cost of increasing the number of false results included. This is a choice that the operator of this algorithm must make, and will depend on the operator's desired false positive rate. This trade-off is explored by choosing an N_{min} cutoff and removing candidate trajectories from the set of 1000 candidates per search. For each value of N_{min}, the m_{50} depth, the number of candidate trajectories, and the number of fakes found are evaluated. These values as a function of N_{min} are visualized in Fig. 4.7. We find that decreasing the value of N_{min} increases m_{50} depth and the number of synthetic objects recovered, as expected. However, this comes at the cost of vastly increasing the number of results produced. Most of the numerous results produced at low values of N_{min} are likely false-positive candidates due to stacks of noise detections. The scaling of achievable m_{50} depth with the SNR of the input catalog is visualized in Fig. 4.8 and enumerated in Tab. 4.2 by choosing a value of N_{min} for each search that corresponds to 200 results per CCD.

4.3.5 Computational Cost

The computational cost of the algorithm scales from negligible to vast depending on the range of velocity trial trajectories, the SNR of the detection catalog, and the value of Δx. Figure 4.9 visualizes the median wall-clock runtime and memory usage among the searches

performed for TNOs and MBAs. Wall-clock time includes the time just for the algorithm operations, and excludes the time for source detection on the input images. The search runtime is $\sim$ 1 min (0.017 core-hours) for SNR 3-5 detection catalogs using the values of Δx in Table 4.1 whereas the memory usage is $\sim 1 - 10$ GB for the same runs. The computational cost balloons for the TNO searches at SNR 1 and 2, where $\Delta x = 1$ in which case the memory usage is $\sim 10 - 100$ GB and search wall-clock times can range from $\sim 10 - 40$ minutes ($0.17 - 0.67$ core-hours).

In Fig. 4.10, we explore the compute cost and m_{50} depth trade off of our algorithm by measuring the CPU-time and memory required to perform the TNO searches and comparing that to the m_{50} depth achieved, assuming 200 results per-detector are reported. We report only the amount of CPU time spent in performing the core-search component of our algorithm, excluding image loading, source detection, and trajectory refinement. The memory usage reported is the maximum amount of memory used during this procedure. The scaling of CPU time and memory usage as a function of Δx is also visualized in Fig. 4.11, verifying the results predicted by Eq. 4.8.

We additionally make an explicit comparison in the depth achieved when utilizing an image-based shift-and-stack approach. We utilize the `KBMOD` software (Whidden et al., 2019) to perform a set of GPU-accelerated shift-and-stack searches on the same images used in this study.[6] We used `KBMOD` to search a similar range of on-sky velocity rates but covering 180° in angle around the ecliptic as opposed to the 360° angular range we searched. Results that had a coadded likelihood of 10 or more were reported by the search. A value of $m_{50} = 25.47 \pm 0.01$, visualized as a dotted black line in Fig. 4.10, was computed for the `KBMOD` search using a similar method of matching search results to synthetic injected sources and modeling the completeness using Eq. 4.10. The SNR $\geq 5, 4, 3, 2$ searches we performed produce an m_{50} depth that is smaller than the `KBMOD` search. Our searches at SNR ≥ 1 surpass the m_{50} depth of the `KBMOD` search, achieving a depth of $m_{50} = 25.87 \pm 0.21$. These numbers provide only rough depth comparison between the catalog and image-based shift-and-stack approach, as we make no attempt to compare or achieve parity between the two

[6]KBMOD is available online at https://github.com/dirac-institute/kbmod/

implementations in terms of false positive and true positive rates, which are ultimately the factors that matter when deriving scientific insight from these searches. However, we can conclude that our method of stacking detection catalogs can achieve comparable depths as an image-based approach in a practical sense.

The `KBMOD` software is GPU-accelerated and provides no CPU-only implementation of its search, making it challenging to perform a compute-time comparison between these two methods. In order to make this comparison explicitly, we produced a CPU-implementation of the approach outlined in Whidden et al. (2019). Our implementation convolves input images with the PSF to form a per-epoch likelihood image, the combination of convolved inverse-variance weighted flux values and convolved inverse-variance values. The likelihood images are shifted at the pixel level by amounts that correspond to the on-sky linear trajectories used in our TNO search. For each trajectory and corresponding set of pixel shifts, the convolved weighted flux and convolved inverse variance values are added together across epochs at the pixel level. The ratio of the summed convolved flux and the square-root of the summed convolved variance is taken to form an image of coadded detection SNR. Pixels in this SNR image that are above a provided threshold are found, which correspond to a putative coadded detection for the given trajectory. In this implementation, we perform and measure the CPU-time required for the pixel shifts, additions, divisions, and thresholding, excluding the actual reporting or filtering of results. This allows us to make a direct comparison of the compute-time (i.e. number of operations) required for the core search components of the catalog and image-based shift-and-stack approaches. We performed a single search using the same velocity and angle ranges used in our TNO-search with a trajectory divergence of no more than 1 arcsecond over the timespan of the data. The CPU-time required for the search was 2.54 hours and required 17.5 GB of memory. While the CPU-time measurement is likely faithful to the expected runtime of the image-based shift-and-stack, the memory usage can likely be reduced in a more optimized implementation. These values are lower limits on the total compute time and memory usage of the image-based shift-and-stack, since we do not find and report detection results in our CPU-only implementation. These measurements are used to compute the relative CPU-time (speedup) and relative memory usage between our detection catalog stacking algorithm and an image-based shift-and-stack

approach, producing lower and upper limits on these value respectively. Figure 4.10 visualizes the total compute-time speedup and relative memory usage while Fig. 4.11 visualizes the speedup and relative memory usage when comparing the core-search components of the catalog and image-based shift-and-stack approach. Examining Fig. 4.11, we find that the core-search component of our algorithm attains speedups of $10 - 10^3$ across values of Δx from $1 - 10$ and SNR $1 - 5$. These results verify the theoretical scaling relations in Eq. 4.8.

Values of the absolute and relative total CPU-time, memory usage, and m_{50} depth achieved by our algorithm and the image-based shift-and-stack approach are reported in Table 4.3. We find that total compute-time speedups range from 3.9 for the SNR ≥ 1 search to 203.3 for the SNR ≥ 5 search, providing speedup over all achievable m_{50} depths. Our algorithm uses $10.9\times$ as much memory for the SNR ≥ 1 search and $50\times$ less memory for the SNR ≥ 5 search, realizing memory savings for most searches except the SNR ≥ 1 search, which has a likely unmanageable memory footprint. Taking the SNR ≥ 2 search as a comparison point, our algorithm achieves a speedup of 30.8 with 88% of the memory footprint of an image-based shift-and-stack approach, while sacrificing 0.25 mag in m_{50} depth.

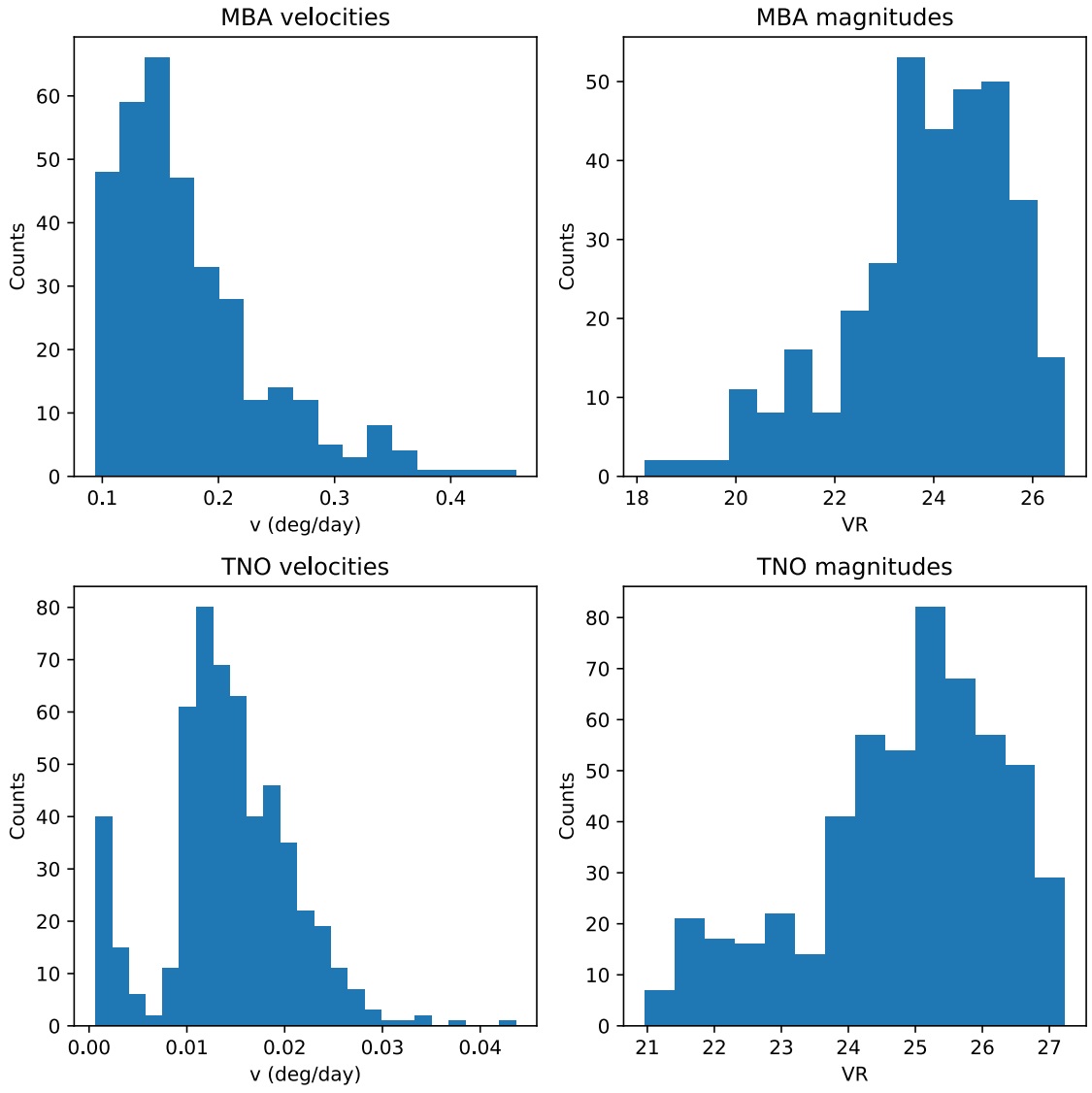


Figure 4.3: The distribution of velocities and magnitudes of the injected synthetic MBAs and TNOs.

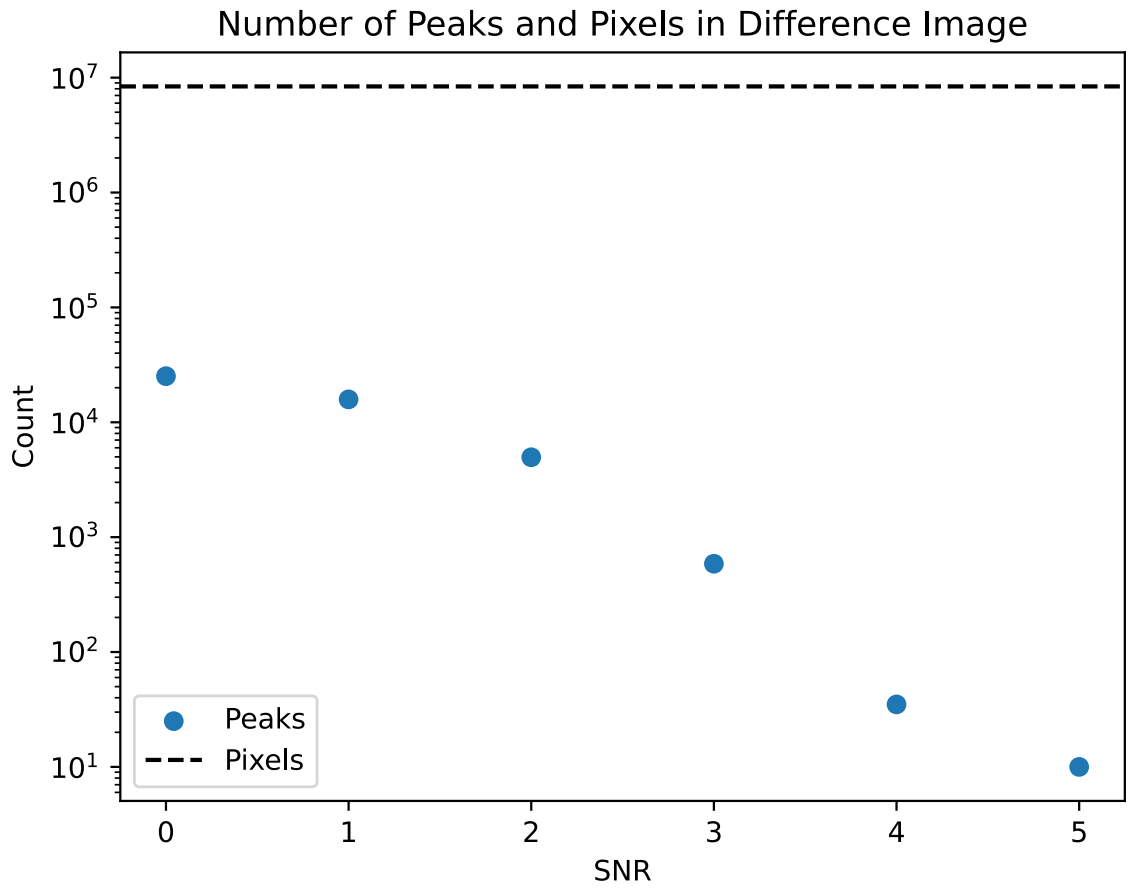


Figure 4.4: The number of peaks and pixels in a likelihood image derived from a 2048×4096 CCD difference image as a function of the SNR threshold. At all values of SNR, the number of detection peaks is smaller by factors of $10^2 - 10^6$.

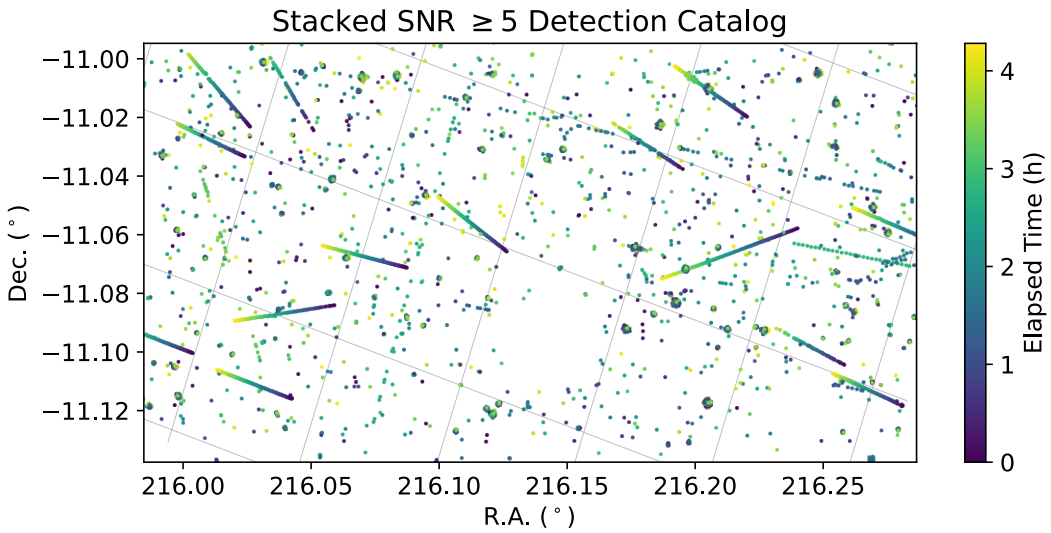


Figure 4.5: A visualization of the sky locations of a stacked detection catalog derived from a 4 hour sequence of DECam images with a grid of constant ecliptic latitude/longitude overlaid. Detections at a single epoch represent the on-sky location of a putative source on a difference image. Detections are accumulated across all 104 epochs and plotted jointly in this visualization. Moving objects appear as lines in this stacked visualization. Stationary variable sources as well as difference imaging artifacts appear as clusters and isolated points.

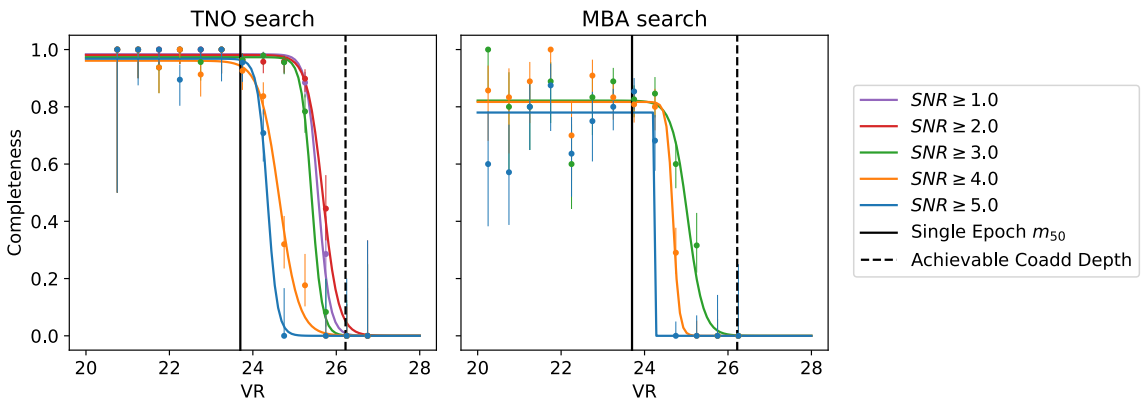


Figure 4.6: The fraction of synthetic objects recovered as a function of their magnitude for the TNO and MBA searches. Dots represent the detection fraction in bins of size 0.5 mag and error bars represent the asymmetric uncertainty in the estimated fraction using the Wilson score interval. Two reference magnitudes are visualized as black solid and dashed vertical lines. The solid black line visualizes the estimated single-epoch $m_{50} = 23.7$. The dashed black line represents the theoretical achievable depth by coadding all of the images and is equal to $m_{50}^{\text{coadd}} = 26.2$.

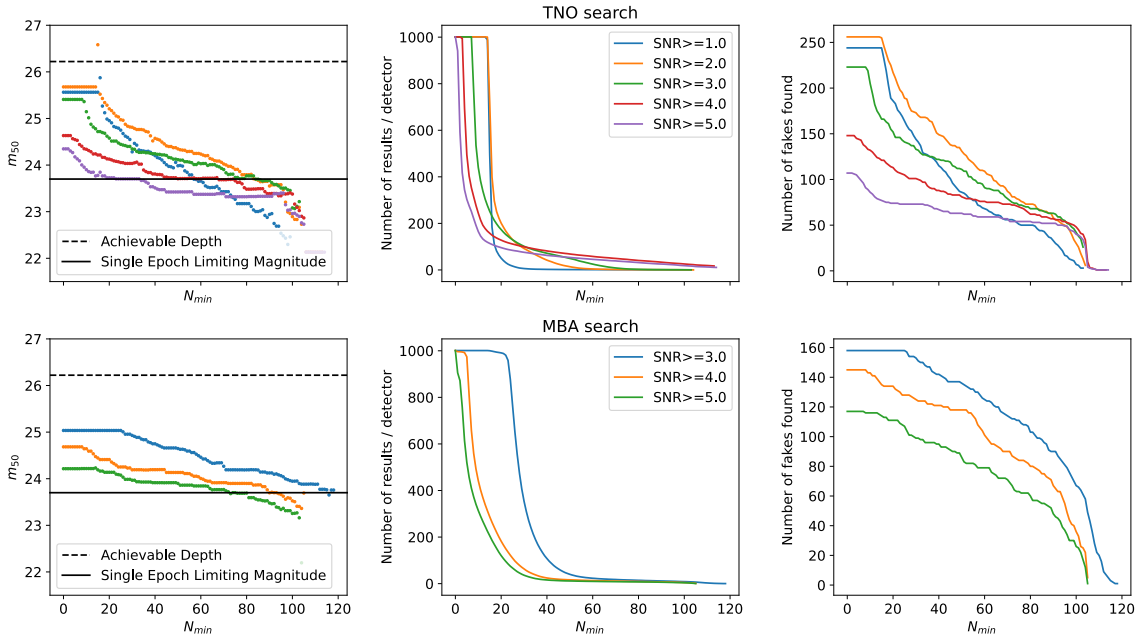


Figure 4.7: The m_{50} depth from fitting of the completeness (left), the number of candidate detections produced per detector (middle), and the number of synthetic objects (fakes) recovered (right) for the TNO and MBA searches as a function of the $N_{\rm min}$ parameter. The solid black line visualizes the estimated single-epoch $m_{50} = 23.7$. The dashed black line represents the theoretical achievable depth by coadding all of the images and is equal to $m_{50}^{\rm coadd} = 26.2$. Decreasing the value of $N_{\rm min}$ increases m_{50} depth and the number of synthetic objects recovered, at the cost of vastly increasing the number of results produced. Most of the results produced at low values of $N_{\rm min}$ are false-positive candidates.

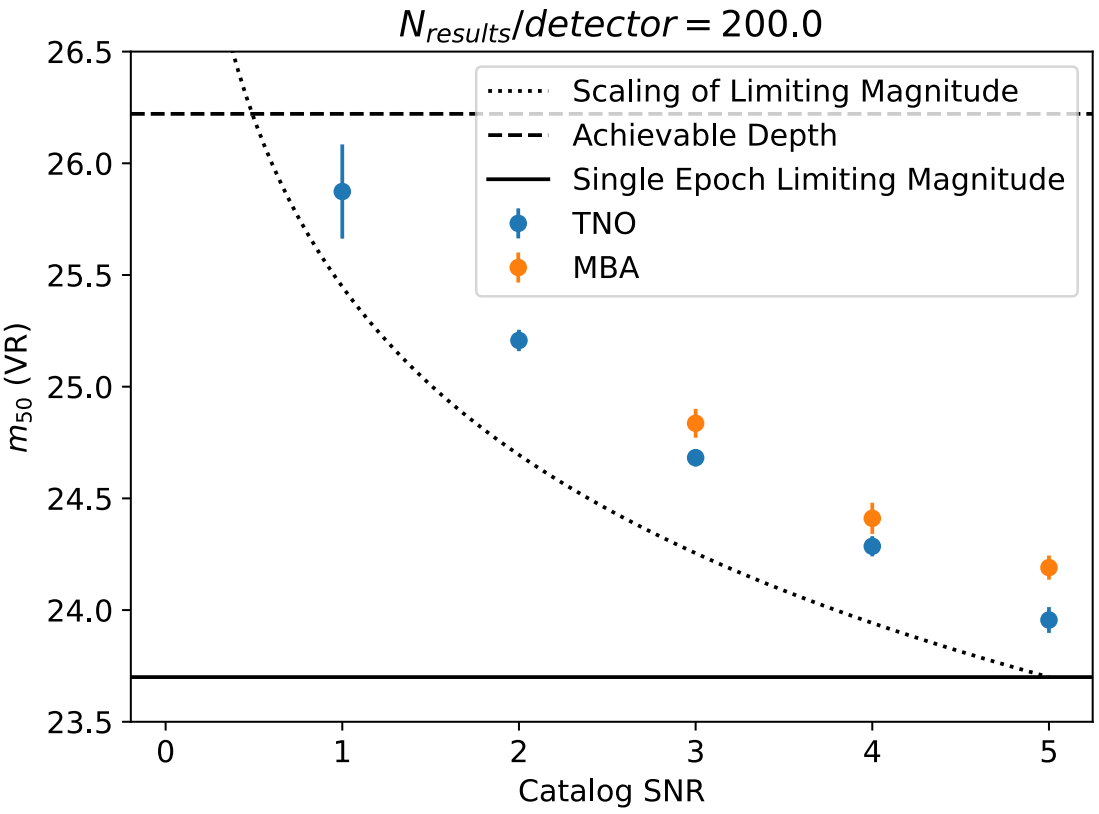


Figure 4.8: The m_{50} depth achieved for the TNO (blue) and MBA (orange) searches as a function of the SNR of the input catalog choosing $N_{\min}$ such that the number of results per detector is fixed at 200. The solid black line represents a single-epoch SNR ~ 5 limiting magnitude of $m_{50} = 23.7$ while the dashed black line represents the theoretical optimal achievable coadded depth of $m_{50}^{\text{coadd}} = 26.2$. The dotted black line visualizes the expected scaling of achieved depth with SNR, extrapolated from the single-epoch limiting magnitude.

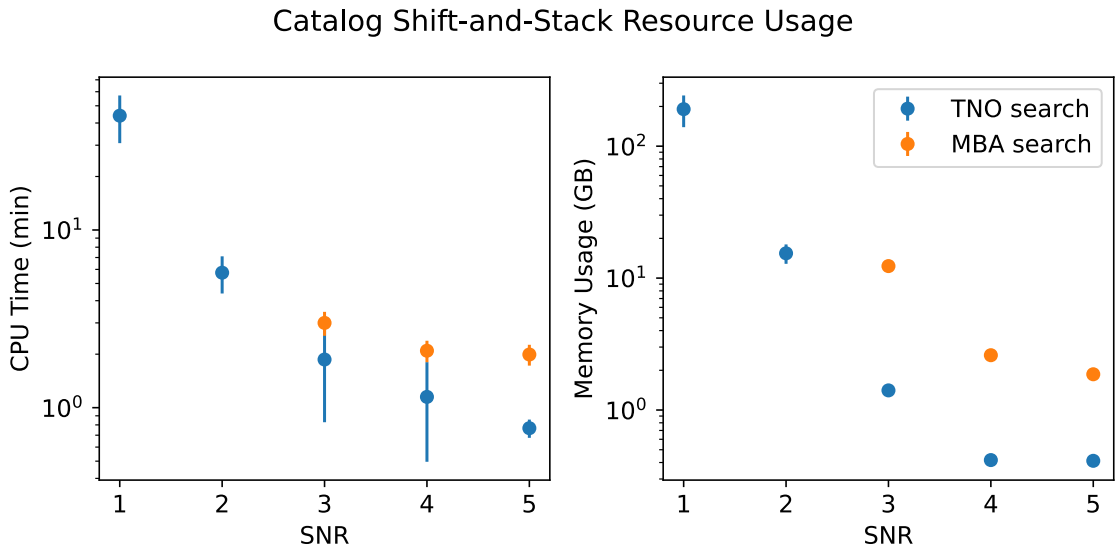


Figure 4.9: The median wall-clock runtime and memory usage of our catalog shift-and-stack algorithm for the TNO (blue) and MBA (orange) searches across the 60 DECam detectors searched. Error bars represent the standard deviation of the runtime and memory usage values measured.

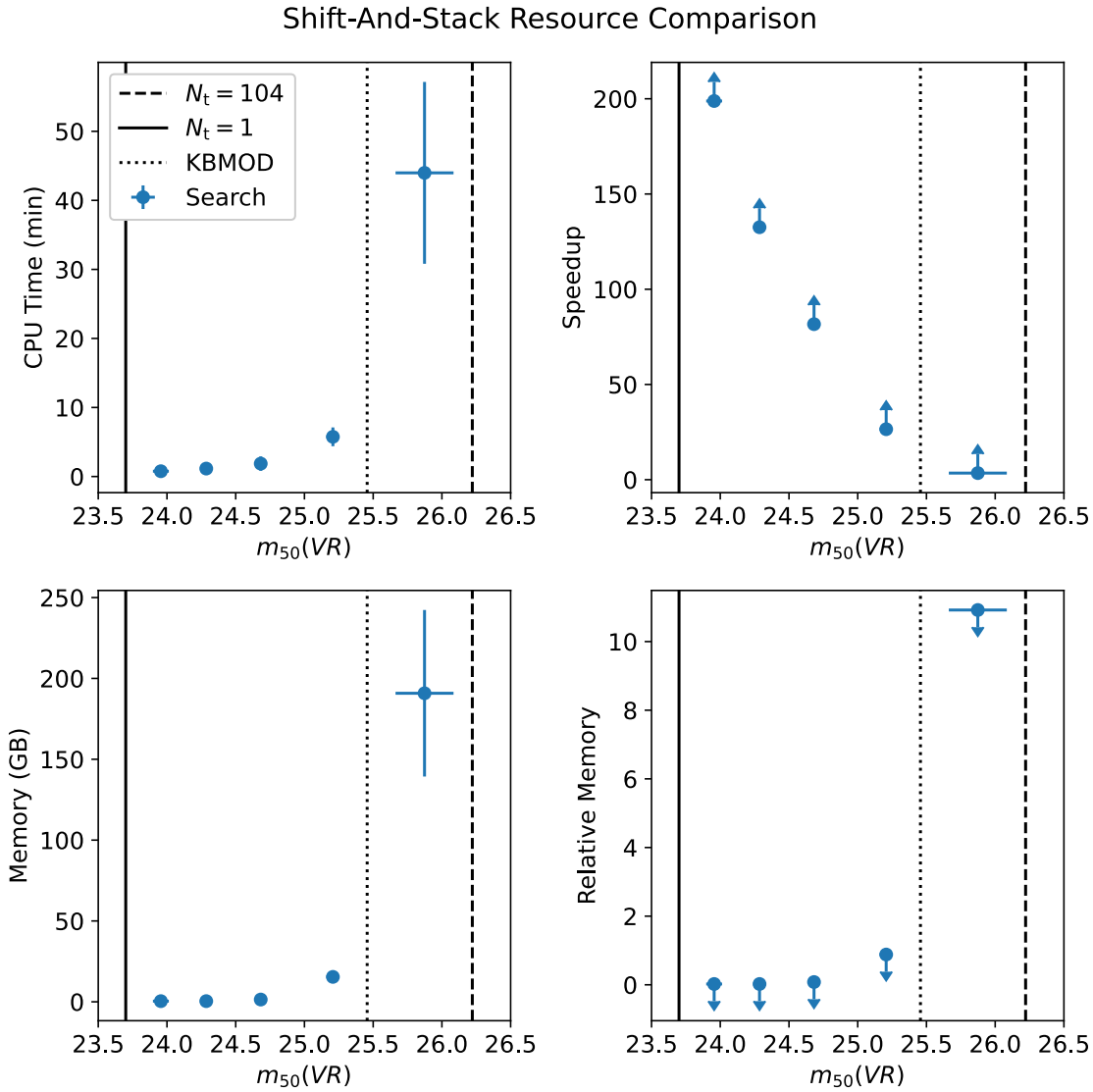


Figure 4.10: The left panels visualize the CPU-time and memory usage of our algorithm as a function of the m_{50} depth achieved. The black solid line visualizes the single-epoch $m_{50} = 23.7$ while the dashed black line visualizes the theoretical maximum $m_{50}^{\text{coadd}} = 26.2$ that could be achieved through optimal coaddition. The dotted black line visualizes the $m_{50}^{\text{KBMOD}} = 25.47$ depth achieved in an image-based shift-and-stack approach, utilizing the `KBMOD` software. The right panels visualize the relative total CPU-time (speedup) and memory usage of our algorithm when compared to an image-based shift-and-stack search.

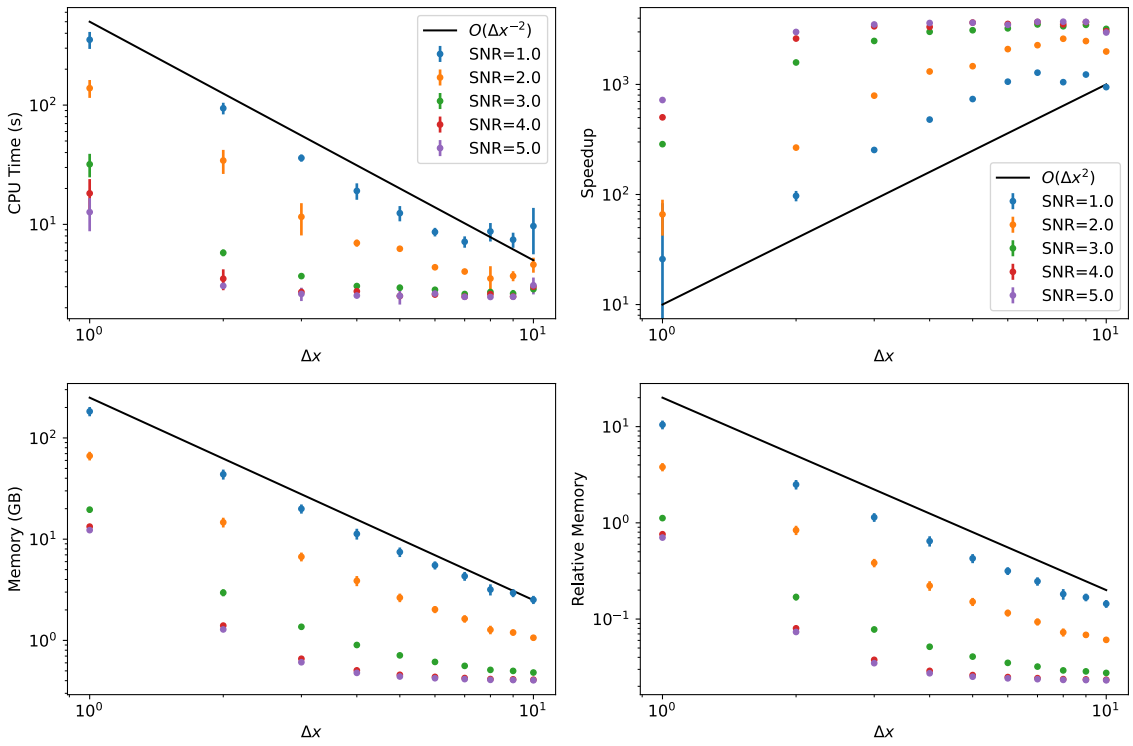


Figure 4.11: The CPU-time and memory usage of the core search component of our algorithm as a function of the Δx value and the SNR of the catalog searched. The black solid lines represent theoretical scaling laws with Δx. The right panels visualize the relative core-search CPU-time (speedup) and memory usage when compared to an implementation of an image-based shift-and-stack search.

SNR	m_{50}	CPU (min)	Memory (GB)	Speedup	Relative Memory (GB)	Δm_{50}
1	25.87 ± 0.21	39.5	190.8	≥ 3.9	≤ 10.92	0.42 ± 0.21
2	25.21 ± 0.05	5.0	15.4	≥ 30.8	≤ 0.88	-0.25 ± 0.05
3	24.68 ± 0.04	1.8	1.4	≥ 85.9	≤ 0.08	-0.77 ± 0.04
4	24.29 ± 0.05	1.1	0.4	≥ 138.6	≤ 0.02	-1.17 ± 0.05
5	23.96 ± 0.06	0.8	0.4	≥ 203.3	≤ 0.02	-1.50 ± 0.06

Table 4.3: Total compute-time, memory usage, and m_{50} depth achieved when searching for TNOs. Speedup, relative memory usage, and difference in m_{50} depth in comparison to the core-search routine of an image-based shift-and-stack approach are provided, using 2.54 core-hours, 17.5 GB of memory, and $m_{50} = 25.47 \pm 0.01$ as fiducial values.

4.4 Conclusions

We have presented an efficient shift-and-stack algorithm capable of finding both bright and faint moving objects in detection catalogs derived from CCD images. The algorithm finds objects in both pure and noisy catalogs derived from high- and low-SNR detections. The algorithm is efficient in the sense that it reduces the number of stacks performed relative to an image-based shift-and-stack algorithm.

We have validated the performance of the algorithm by using it to recover synthetic moving objects implanted in difference images representing the Main Belt Asteroid and Trans-Neptunian Object populations. The algorithm is found to be computationally efficient and effective at recovering faint moving objects below the noise floor of an individual image. We provide a method to choose the parameters of the algorithm that determine the depth it can achieve as well as its false positive rate. We show how depth achieved scales with the number of candidate objects discovered, and the required resource usage to attain a given depth. We provide a comparison of resource usage and depth obtained between our method and an image-based shift-and-stack approach. We find that our method can achieve wall-clock runtime speedups of $\sim 30\times$ with 88% of the memory usage while sacrificing 0.25 mag in m_{50} depth.

The algorithm does not reach the theoretical limit expected from ideal coaddition. One possible reason for this is due to our limited method of results filtering. In this study, we analyze depth based on the first 1000 candidates produced in a search. Applying a more sophisticated filtering method on the produced results may allow fainter objects of lower significance to be found while keeping the number of results manageable. Additionally, it is possible that this is due to the metric used for accumulating signal: the number of detections across epochs in an aperture of variable size. A different metric, perhaps the detection likelihood could be accumulated instead, mimicking the approach of Whidden et al. (2019). The likelihood naturally weights signal detections higher than noise detections, whereas the indicator on detected vs non-detected provides equal weight to these outcomes. In this scheme, a likelihood ratio test could be used to reason about the relative odds that a candidate moving object is real or the result of noise detections, following more closely the

detection catalog coaddition framework outlined in Budavári et al. (2017), which was able to clearly distinguish real faint objects from noise detections at very low detection significance. A benefit of our equal-weighting scheme is that it potentially makes our shift-and-stack method less sensitive to contamination by difference image artifacts and stationary sources. Exploring the cost-benefit trade-off of using the likelihood in the stacking procedure is left for future work.

This algorithm provides a path forward for broad application of shift-and-stack to large imaging surveys such as the Legacy Survey of Space and Time (LSST; Željko Ivezić et al. (2019)). It is very computationally challenging to perform complete image-based shift-and-stack searches on such large surveys, especially for populations of "fast"-moving objects such as main belt asteroids and near-Earth objects. As demonstrated in this work, it is possible to probe 0.5 mag deeper into the population of main belt asteroids at relatively low computational expense by detecting at SNR = 3 and applying this algorithm. This algorithm will be immediately applicable in finding these objects in near-ecliptic deep drilling fields of the LSST as well as in the Rubin Observatory's commissioning survey of these fields.

Chapter 5

CONCLUSIONS

This thesis lays the groundwork for how to scale scientific analyses in the era of the Vera C. Rubin Observatory's Legacy Survey of Space and Time. In Chapter 2, I showed how scientific analyses can be scaled to the petabyte-sized detection catalog produced by the LSST by utilizing tools that partition the dataset into smaller units upon which a user's code is executed. In Chapter 3, I showed how the LSST Science Pipelines can be applied to process the images from the LSST. Additionally, I outlined tools that can be used by users and groups to perform their own processing campaigns with the LSST Science Pipelines. In Chapter 4, I introduced improvements to the shift-and-stack algorithm, enabling its broad application to the LSST imaging dataset and its use in finding faint Main Belt Asteroids in the survey. It is my hope that these tools and algorithms I have built and shared have a broad impact by minimizing the technical burden faced by the astronomy community as we take action toward our shared goal of understanding the universe.

BIBLIOGRAPHY

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

Babuji, Y., Woodard, A., Li, Z., et al. 2019, in 28th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC), doi: 10.1145/3307681.3325400

Ballard, D. H. 1981, Pattern recognition, 13, 111

Bannister, M. T., Kavelaars, J. J., Petit, J.-M., et al. 2016, , 152, 70, doi: 10.3847/0004-6256/152/3/70

Batygin, K., & Brown, M. E. 2016, The Astronomical Journal, 151, 22, doi: 10.3847/0004-6256/151/2/22

Becker, A. 2015, HOTPANTS: High Order Transform of PSF ANd Template Subtraction, Astrophysics Source Code Library, record ascl:1504.004

Bektesevic, D., Chiang, H.-F., Lim, K.-T., et al. 2020, A Gateway to Astronomical Image Processing: Vera C. RubinObservatory LSST Science Pipelines on AWS, arXiv, doi: 10.48550/ARXIV.2011.06044

Bellm, E. C., Kulkarni, S. R., Graham, M. J., et al. 2019, Publications of the Astronomical Society of Pacific, 131, 018002, doi: 10.1088/1538-3873/aaecbe

Bernardinelli, P. H., Smotherman, H., Langford, Z., et al. 2024, , 167, 134, doi: 10.3847/1538-3881/ad1527

Bernardinelli, P. H., Smotherman, H., Langford, Z., et al. 2024, The Astronomical Journal, 167, 134, doi: 10.3847/1538-3881/ad1527

Bernstein, G. M., Trilling, D., Allen, R., et al. 2004, The Astronomical Journal, 128, 1364

Bernstein, G. M., Trilling, D. E., Allen, R. L., et al. 2004a, , 128, 1364, doi: 10.1086/422919

—. 2004b, , 128, 1364, doi: 10.1086/422919

Bosch, J., Armstrong, R., Bickerton, S., et al. 2018, , 70, S5, doi: 10.1093/pasj/psx080

Bosch, J., AlSayyad, Y., Armstrong, R., et al. 2019, in Astronomical Society of the Pacific Conference Series, Vol. 523, Astronomical Data Analysis Software and Systems XXVII, ed. P. J. Teuben, M. W. Pound, B. A. Thomas, & E. M. Warner, 521, doi: 10.48550/arXiv.1812.03248

Bottke, W. F., Durda, D. D., Nesvorný, D., et al. 2005, , 175, 111, doi: 10.1016/j.icarus.2004.10.026

Boyajian, T. S., LaCourse, D. M., Rappaport, S. A., et al. 2016, Monthly Notices of the Royal Astronomical Society, 457, 3988, doi: 10.1093/mnras/stw218

Budavári, T., Szalay, A. S., & Loredo, T. J. 2017, The Astrophysical Journal, 838, 52, doi: 10.3847/1538-4357/aa6335

Chambers, K. C., Magnier, E. A., Metcalfe, N., et al. 2019, The Pan-STARRS1 Surveys. https://arxiv.org/abs/1612.05560

Cheng, Y., Liu, F. C., Jing, S., Xu, W., & Chau, D. H. 2018, in Proceedings of the Practice and Experience on Advanced Research Computing, PEARC '18 (New York, NY, USA: Association for Computing Machinery), doi: 10.1145/3219104.3229288

Dalitz, C., Schramke, T., & Jeltsch, M. 2017, Image Processing On Line, 7, 184

Danieli, S., Kado-Fong, E., Huang, S., et al. 2024, arXiv e-prints, arXiv:2410.01884, doi: 10.48550/arXiv.2410.01884

Dark Energy Survey Collaboration, Abbott, T., Abdalla, F. B., et al. 2016, , 460, 1270, doi: 10.1093/mnras/stw641

Deelman, E., Vahi, K., Rynge, M., et al. 2019, Computing in Science Engineering, 21, 22, doi: 10.1109/MCSE.2019.2919690

Dekany, R., Smith, R. M., Riddle, R., et al. 2020, , 132, 038001, doi: 10.1088/1538-3873/ab4ca2

DeMeo, F. E., & Carry, B. 2014, , 505, 629, doi: 10.1038/nature12908

Denneau, L., Jedicke, R., Grav, T., et al. 2013, Publications of the Astronomical Society of the Pacific, 125, 357, doi: 10.1086/670337

Dubois-Felsmann, G., Lim, K.-T., Wu, X., et al. 2017, LSST Science Platform Design, Rubin Observatory. http://ls.st/ldm-542

Duda, R. O., & Hart, P. E. 1972, Commun. ACM, 15, 11–15, doi: 10.1145/361237.361242

Duncan, M. J., Levison, H. F., & Budd, S. M. 1995, The Astronomical Journal, 110, 3073, doi: 10.1086/117748

Fernández, J., & Ip, W.-H. 1984, Icarus, 58, 109, doi: 10.1016/0019-1035(84)90101-5

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

Fraser, W. C., Porter, S. B., Peltier, L., et al. 2024, , 5, 227, doi: 10.3847/PSJ/ad6f9e

Gaia Collaboration, Prusti, T., de Bruijne, J. H. J., et al. 2016a, Astronomy and Astrophysics, 595, A1, doi: 10.1051/0004-6361/201629272

Gaia Collaboration, Prusti, T., de Bruijne, J. H. J., et al. 2016b, , 595, A1, doi: 10.1051/0004-6361/201629272

Gaia Collaboration, Vallenari, A., Brown, A. G. A., et al. 2023, , 674, A1, doi: 10.1051/0004-6361/202243940

Gladman, B., Kavelaars, J., Nicholson, P. D., Loredo, T. J., & Burns, J. A. 1998, The Astronomical Journal, 116, 2042

Graham, M. J., Kulkarni, S. R., Bellm, E. C., et al. 2019, , 131, 078001, doi: 10.1088/1538-3873/ab006c

Hahn, J. M., & Malhotra, R. 2005, The Astronomical Journal, 130, 2392, doi: 10.1086/452638

Heasley, J. N. 1999, in Astronomical Society of the Pacific Conference Series, Vol. 189, Precision CCD Photometry, ed. E. R. Craine, D. L. Crawford, & R. A. Tucker, 56

Heinze, A. N., Metchev, S., & Trollo, J. 2015, The Astronomical Journal, 150, 125

Heinze, A. N., Metchev, S., & Trollo, J. 2015a, , 150, 125, doi: 10.1088/0004-6256/150/4/125

—. 2015b, , 150, 125, doi: 10.1088/0004-6256/150/4/125

Helmi, A. 2020, , 58, 205, doi: 10.1146/annurev-astro-032620-021917

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

Holman, M. J., & Wisdom, J. 1993, , 105, 1987, doi: 10.1086/116574

Hough, P. V. C. 1959, Conf. Proc. C, 590914, 554

Ivezić, Ž., Kahn, S. M., Tyson, J. A., et al. 2019, , 873, 111, doi: 10.3847/1538-4357/ab042c

Jenness, T., Bosch, J. F., Salnikov, A., et al. 2022, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 12189, Software and Cyberinfrastructure for Astronomy VII, 1218911, doi: 10.1117/12.2629569

Jones, R. L., Gladman, B., Petit, J. M., et al. 2006, , 185, 508, doi: 10.1016/j.icarus.2006.07.024

Jurić, M., Ciardi, D., & Dubois-Felsmann, G. 2017, LSST Science Platform Vision Document, Rubin Observatory. http://ls.st/lse-319

Juric, M., Stetzler, S., & Slater, C. T. 2021, Checkpoint, Restore, and Live Migration for Science Platforms. https://arxiv.org/abs/2101.05782

Kaiser, N., Burgett, W., Chambers, K., et al. 2010, in , Vol. 7733, Ground-based and Airborne Telescopes III, 77330E, doi: 10.1117/12.859188

Kluyver, T., Ragan-Kelley, B., Pérez, F., et al. 2016, in Positioning and Power in Academic Publishing: Players, Agents and Agendas, ed. F. Loizides & B. Scmidt (Netherlands: IOS Press), 87–90. https://eprints.soton.ac.uk/403913/

Kubica, J., Denneau, L., Grav, T., et al. 2007, Icarus, 189, 151, doi: https://doi.org/10.1016/j.icarus.2007.01.008

Kurlander, J., Bernardinelli, P., Schwamb, M., et al. 2024, in AAS/Division for Planetary Sciences Meeting Abstracts, Vol. 56, AAS/Division for Planetary Sciences Meeting Abstracts, 212.07

Lang, D., Hogg, D. W., Mierle, K., Blanton, M., & Roweis, S. 2010, , 137, 1782

Levison, H. F., Morbidelli, A., VanLaerhoven, C., Gomes, R., & Tsiganis, K. 2008, Icarus, 196, 258, doi: 10.1016/J.ICARUS.2007.11.035

LSST Science Collaboration, Abell, P. A., Allison, J., et al. 2009, arXiv e-prints, arXiv:0912.0201, doi: 10.48550/arXiv.0912.0201

Maeno, T., Alekseev, A., Barreiro Megino, F. H., et al. 2024, Computing and Software for Big Science, 8, doi: 10.1007/s41781-024-00114-3

Malhotra, R. 1993, Nature, 365, 819, doi: 10.1038/365819a0

Masci, F. J., Laher, R. R., Rusholme, B., et al. 2019, , 131, 018003, doi: 10.1088/1538-3873/aae8ac

Millis, R. L., Buie, M. W., Wasserman, L. H., et al. 2002, , 123, 2083, doi: 10.1086/339481

Moeyens, J., Jurić, M., Ford, J., et al. 2021, The Astronomical Journal, 162, 143, doi: 10.3847/1538-3881/ac042b

Morbidelli, A. 2010, Comptes Rendus Physique, 11, 651, doi: 10.1016/j.crhy.2010.11.001

Morbidelli, A., Brasser, R., Gomes, R., Levison, H. F., & Tsiganis, K. 2010, , 140, 1391, doi: 10.1088/0004-6256/140/5/1391

Morganson, E., Gruendl, R. A., Menanteau, F., et al. 2018, , 130, 074501, doi: 10.1088/1538-3873/aab4ef

Napier, K. J., Lin, H. W., Gerdes, D. W., et al. 2024, , 5, 50, doi: 10.3847/PSJ/ad1528

Nesvorný, D., & Vokrouhlický, D. 2016, The Astrophysical Journal, 825, 94, doi: 10.3847/0004-637X/825/2/94

Nesvorný, D. 2011, The Astrophysical Journal Letters, 742, L22, doi: 10.1088/2041-8205/742/2/L22

Nguyen, T., Woods, D. F., Ruprecht, J., & Birge, J. 2024, The Astronomical Journal, 167, 113, doi: 10.3847/1538-3881/ad20e0

Norman, M., Kellen, V., Smallen, S., et al. 2021, in Practice and Experience in Advanced Research Computing, PEARC '21 (New York, NY, USA: Association for Computing Machinery), doi: 10.1145/3437359.3465586

O'Mullane, W., Economou, F., Huang, F., et al. 2021, arXiv e-prints, doi: 10.48550/ARXIV.2111.15030

Patterson, M. T., Bellm, E. C., Rusholme, B., et al. 2018, Publications of the Astronomical Society of the Pacific, 131, 018001, doi: 10.1088/1538-3873/aae904

Peloton, J., Arnault, C., & Plaszczynski, S. 2018, FITS Data Source for Apache Spark, arXiv, doi: 10.48550/ARXIV.1804.07501

Percival, W. J., Nichol, R. C., Eisenstein, D. J., et al. 2007, The Astrophysical Journal, 657, 51, doi: 10.1086/510772

Peter J Rousseeuw, Stefan Van Aelst, K. V. D., & Gulló, J. A. 2004, Technometrics, 46, 293, doi: 10.1198/004017004000000329

Portillo, S. K. N., Speagle, J. S., & Finkbeiner, D. P. 2020, The Astronomical Journal, 159, 165, doi: 10.3847/1538-3881/ab76ba

Riess, A. G., Filippenko, A. V., Challis, P., et al. 1998, , 116, 1009, doi: 10.1086/300499

Rousseeuw, P. 1985, Mathematical Statistics and Applications Vol. B, 283, doi: 10.1007/978-94-009-5438-0_20

Rousseeuw, P. J., & Driessen, K. V. 1999, Technometrics, 41, 212, doi: 10.1080/00401706.1999.10485670

Scaramella, R., Mellier, Y., Amiaux, J., et al. 2014, in IAU Symposium, Vol. 306, Statistical Challenges in 21st Century Cosmology, ed. A. Heavens, J.-L. Starck, & A. Krone-Martins, 375–378, doi: 10.1017/S1743921314011089

Schwamb, M. E., Brown, M. E., Rabinowitz, D. L., & Ragozzine, D. 2010, , 720, 1691, doi: 10.1088/0004-637X/720/2/1691

Shao, M., Nemati, B., Zhai, C., et al. 2014a, , 782, 1, doi: 10.1088/0004-637X/782/1/1

—. 2014b, , 782, 1, doi: 10.1088/0004-637X/782/1/1

Shappee, B., Prieto, J., Stanek, K. Z., et al. 2014, in American Astronomical Society Meeting Abstracts, Vol. 223, American Astronomical Society Meeting Abstracts #223, 236.03

Smotherman, H., Connolly, A. J., Kalmbach, J. B., et al. 2021, The Astronomical Journal, 162, 245, doi: 10.3847/1538-3881/ac22ff

Smotherman, H., Bernardinelli, P. H., Portillo, S. K. N., et al. 2024, , 167, 136, doi: 10.3847/1538-3881/ad1524

Spergel, D., Gehrels, N., Baltay, C., et al. 2015, arXiv e-prints, arXiv:1503.03757. https://arxiv.org/abs/1503.03757

Strauss, R., Trilling, D. E., Bernardinelli, P. H., et al. 2024a, , 167, 135, doi: 10.3847/1538-3881/ad1526

Strauss, R., McNeill, A., Trilling, D. E., et al. 2024b, , 168, 184, doi: 10.3847/1538-3881/ad7366

Thain, D., Tannenbaum, T., & Livny, M. 2005, Concurrency - Practice and Experience, 17, 323

Tonry, J. L., Denneau, L., Heinze, A. N., et al. 2018, Publications of the Astronomical Society of Pacific, 130, 064505, doi: 10.1088/1538-3873/aabadf

Trilling, D. E., Gerdes, D. W., Jurić, M., et al. 2024, , 167, 132, doi: 10.3847/1538-3881/ad1529

Trilling, D. E., Gerdes, D. W., Jurić, M., et al. 2024, The Astronomical Journal, 167, 132, doi: 10.3847/1538-3881/ad1529

Trujillo, C. A., Jewitt, D. C., & Luu, J. X. 2001, , 122, 457, doi: 10.1086/321117

Trujillo, C. A., Fuentes, C., Gerdes, D. W., et al. 2024, , 167, 133, doi: 10.3847/1538-3881/ad1523

Tsiganis, K., Gomes, R., Morbidelli, A., & Levison, H. F. 2005, Nature, 435, 459, doi: 10.1038/nature03539

Wang, D. L., Monkewitz, S. M., Lim, K.-T., & Becla, J. 2011, in SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, 1–11, doi: 10.1145/2063348.2063364

Whidden, P. J., Kalmbach, J. B., Connolly, A. J., et al. 2019, The Astronomical Journal, 157, 119, doi: 10.3847/1538-3881/aafd2d

Zackay, B., & Ofek, E. O. 2017, The Astrophysical Journal, 836, 188, doi: 10.3847/1538-4357/836/2/188

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. 2010, in Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10 (USA: USENIX Association), 10. https://dl.acm.org/doi/10.5555/1863103.1863113

Zečević, P., Slater, C. T., Jurić, M., et al. 2019, Astronomical Journal, 158, 37, doi: 10.3847/1538-3881/ab2384

Željko Ivezić, Kahn, S. M., Tyson, J. A., et al. 2019, The Astrophysical Journal, 873, 111, doi: 10.3847/1538-4357/ab042c