The Astronomical Journal, 168:56 (20pp), 2024 August https://doi.org/10.3847/1538-3881/ad4da5

The Solar System Notification Alert Processing System (SNAPS): Asteroid Population Outlier Detection

Michael Gowanlock[1,2], David E. Trilling[1,2], Daniel Kramer[1,2], Maria Chernyavskaya[2], and Andrew McNeill[3]
[1] School of Informatics, Computing, and Cyber Systems, Northern Arizona University, P.O. Box 5693, Flagstaff, AZ 86011, USA; michael.gowanlock@nau.edu
[2] Department of Astronomy and Planetary Science, Northern Arizona University, P.O. Box 6010, Flagstaff, AZ 86011, USA
[3] Department of Physics, Lehigh University, 16 Memorial Drive East, Bethlehem, PA 18015, USA
Received 2023 October 27; revised 2024 May 8; accepted 2024 May 15; published 2024 July 5

Abstract

The Solar system Notification Alert Processing System (SNAPS) is a Zwicky Transient Facility (ZTF) and Rubin Observatory alert broker that will send alerts to the community regarding interesting events in the solar system. SNAPS is actively monitoring solar system objects and one of its functions is to compare objects (primarily main belt asteroids) to one another to find those that are outliers relative to the population. In this paper, we use the `SNAPShot1` data set, which contains 31,693 objects from ZTF, and derive outlier scores for each of these objects. SNAPS employs an unsupervised approach; consequently, to derive outlier rankings for each object, we propose four different outlier metrics such that we can explore variants of the outlier scores and add confidence to the outlier rankings. We also provide outlier scores for each object in each permutation of 15 feature spaces, between two and 15 features, which yields 32,752 total feature spaces. We show that we can derive population outlier rankings each month at Rubin Observatory scale using four Nvidia A100 GPUs, and present several avenues of scientific investigation that can be explored using population outlier detection.

Unified Astronomy Thesaurus concepts: Asteroids (72); Small solar system bodies (1469); Sky surveys (1464); Astroinformatics (78); GPU computing (1969)

1. Introduction

The Rubin Observatory's Legacy Survey of Space and Time (LSST; Ivezić et al. 2019) will have an major impact on time-domain astronomy. To prepare for this large all-sky survey, many researchers have been designing data processing pipelines (Coughlin et al. 2021; van Roestel et al. 2021) for precursor surveys, such as the Zwicky Transient Facility (ZTF; Bellm et al. 2018). While ZTF generates roughly a tenth of the LSST data volume, it and other LSST precursor surveys such as the Asteroid Terrestrial-impact Last Alert System (Tonry et al. 2018), Automated Survey for SuperNovae (Shappee et al. 2014), and the Catalina Real-Time Transient Survey (Drake et al. 2014) are instrumental for ensuring that the necessary preparations have been made to maximize the scientific return of LSST. Several alert brokers are currently being developed to prepare for LSST including Fink (Möller et al. 2020), ALeRCE (Förster et al. 2021; Sánchez-Sáez et al. 2021), ANTARES (Matheson et al. 2021), Lasair (Smith et al. 2019), among others.[4]

Recent LSST preparation efforts have focused on classifying astrophysical sources (Soraisam et al. 2020). In the context of solar system science, the vast majority of objects are asteroids, and so classification is less important in this context (e.g., classifying an asteroid versus comet is not very illuminating). Instead, a major focus of the solar system community is to detect interesting transient events in addition to detecting asteroids that are different relative to the population.

In this paper, we present our population outlier detection software infrastructure for the Solar system Notification Alert Processing System (SNAPS). Trilling et al. (2023) presented the first SNAPS data release (`SNAPShot1`), which contains 31,693 asteroids from ZTF. Each asteroid has been observed at least 51 times and has been assigned a rotation period and light-curve amplitude among other derived properties. Beyond solar system science, ZTF has been similarly used for deriving the rotation periods of fully convective stars (Lu et al. 2022). It is clear that the time-domain capabilities of new observatories are revolutionizing our understanding of the solar system and beyond.

At present, SNAPS is operating as a downstream broker from ANTARES (Matheson et al. 2021), and will continue to do so in the LSST era (more detail on SNAPS can be found in Trilling et al. 2023). Population outlier detection refers to detecting asteroids that have interesting properties relative to the other asteroids in the database. Because we have few examples of outlying objects in the solar system, we use an unsupervised machine-learning approach that detects interesting objects without the use of binary classification labels (i.e., inlier versus outlier). Our approach ranks the objects by assigning them an outlier score, and this will allow users of SNAPS to customize the number of objects that they may want to investigate further; for instance, a user may find an interesting object and then use telescopic follow-up observations for additional analysis. Furthermore, we use an ensemble of algorithms and outlier detection metrics to guide the search for outliers, as the methods may disagree on which objects have the highest outlier scores.

The paper is organized as follows. Section 2 describes the data sets used throughout the paper. Section 3 presents the population outlier detection system. Section 4 highlights key results by deriving population outlier scores for objects in our ZTF catalog, in addition to showing system performance using

[4] For an update-to-date list of LSST alert brokers, see the following URL: https://www.lsst.org/scientists/alert-brokers.

Table 1
Features Used for Outlier Detection in SNAPShot1

Feature	Description	NaN to Mean	Derived	μ	σ
G_BR	The Bowell G value in the r filter (Bowell et al. 1989).		✓	0.226	0.234
H_BR	The absolute magnitude of the object, in the Bowell HG system, in the r filter.		✓	14.108	1.877
G_BG	The Bowell G value in the g filter.		✓	0.216	0.255
H_BG	The absolute magnitude of the object, in the Bowell HG system, in the g filter.		✓	14.685	1.902
LCAMP	The amplitude of the light curve.		✓	0.322	0.300
ROTPER	The rotation period (in hours) of the object.		✓	101.150	643.548
GRCOLOR	The $g-r$ color of the object.		✓	0.596	0.146
SIGGRCOLOR	The 1σ error of the $g-r$ color.		✓	0.008	0.006
PERIPOWER	The normalized Lomb–Scargle periodogram power of the rotation period.		✓	0.514	0.289
NEOWISE_DIAM	The diameter of the object calculated by the NEOWISE survey.	✓		8.263	15.675
NEOWISE_VALBEDO	The albedo in the V band, calculated by the NEOWISE survey.	✓		0.196	0.185
MPC_A	The object's semimajor axis in astronomical units.			2.698	1.837
MPC_E	The object's eccentricity.			0.144	0.100
MPC_I	The object's inclination in degrees.			9.660	8.917
HAVG	The object's average absolute magnitude.		✓	14.045	1.830

Note. The table shows each feature and a description. The text describes that NEOWISE_DIAM and NEOWISE_VALBEDO do not provide values for all ZTF objects; therefore, those objects that are missing values, are replaced by the mean of the distribution of those features. The mean and standard deviation (μ, σ) of the features before normalization are shown as they are used to generate a synthetic data set used to assess population outlier detection at LSST scale.

this catalog, and expected performance at LSST scale. Section 5 shows example avenues of scientific investigation that can be carried out using the system. Finally, Section 6 discusses population outlier detection in the time-domain era of astronomy and future work directions.

2. Data Sets

In this section, we describe the data sets used in our evaluation, including SNAPShot1 (Trilling et al. 2023), which is a ZTF data set. We also outline a synthetic data set that is used to assess the performance of the system at LSST scale.

2.1. SNAPShot1

The first SNAPS data release is SNAPShot1, and a detailed description of this data release is outlined in Trilling et al. (2023). This data release contains $|D| = 31{,}693$ ZTF asteroids that have at least 51 observations, such that properties can be derived, including light-curve properties, such as rotation periods and amplitudes. When we describe the data normalization procedure in the next section, we will summarize the properties of each asteroid.

SNAPShot1 includes data from the NEOWISE catalog (Mainzer et al. 2019), and we used the NEOWISE_DIAM and NEOWISE_VALBEDO features as those properties may be useful for detecting outliers. However, some of the ZTF objects were not included in the NEOWISE catalog; consequently, in cases where values were undefined (using "NaN" values), we replaced those values with the mean of the distribution. This ensures that those objects with undefined values are considered typical, and therefore, will not be ranked as an outlier based on the NEOWISE_DIAM and NEOWISE_VALBEDO properties.

2.2. LSST Synthetic Data Set

The LSST catalog will contain roughly 5 million asteroids at the end of the 10 yr survey. To examine the performance of the system at LSST scale, we generate a synthetic data set with 5 million objects with the same $d = 15$ dimensions used for SNAPShot1 described in Table 1 and denote this data set as LSST5M. We replicate the features of each object in SNAPShot1 and then add noise to the values of each feature. The noise added to each feature is drawn from a Gaussian distribution using $\mu = 0$ where the standard deviation, σ, is that of each feature in SNAPShot1 (as shown in Table 1). While we do not expect that this data set will be exceedingly representative of the LSST catalog, it is sufficient for testing the performance of population outlier detection with SNAPS at LSST scale (see Section 4.5).

3. Finding Outlying Asteroids: An Unsupervised Approach

The two major categories of machine learning are supervised and unsupervised learning (Zhou et al. 2017). Supervised learning takes as input feature vectors with class labels and attempts to learn the function that maximizes classifying objects with the correct class label. In contrast, unsupervised learning does not use class labels and instead attempts to identify patterns in data.

Other time-domain alert brokers that will classify sources outside of the solar system, such as Fink (Möller et al. 2020), ALeRCE (Förster et al. 2021; Sánchez-Sáez et al. 2021), and ANTARES (Matheson et al. 2021). Regarding the solar system, the alert broker Fink (Möller et al. 2020) has been used to identify new candidate solar system objects (Le Montagner et al. 2023), and they also design a new phase function model that corrects for the geometric properties of asteroids, including spin coordinates, to better characterize sparse photometry from ZTF (B. Carry et al. 2024, in preparation). Unlike much of the prior work, source classification is not the target of our alert broker; rather, we are interested in identifying objects exhibiting interesting behavior either relative to its prior observational record, or as compared to the greater population of objects. Here, we refer to the former as *real-time* and the latter as *population* outlier detection, respectively. Since there are very few known examples of small bodies exhibiting interesting/transient behavior, we are focused on unsupervised approaches to outlier detection, as we do not have sufficient examples of outlier activity needed to train supervised approaches. Furthermore, since LSST is expected to detect many new objects which may exhibit exotic properties for which we have no known examples, we do not want to

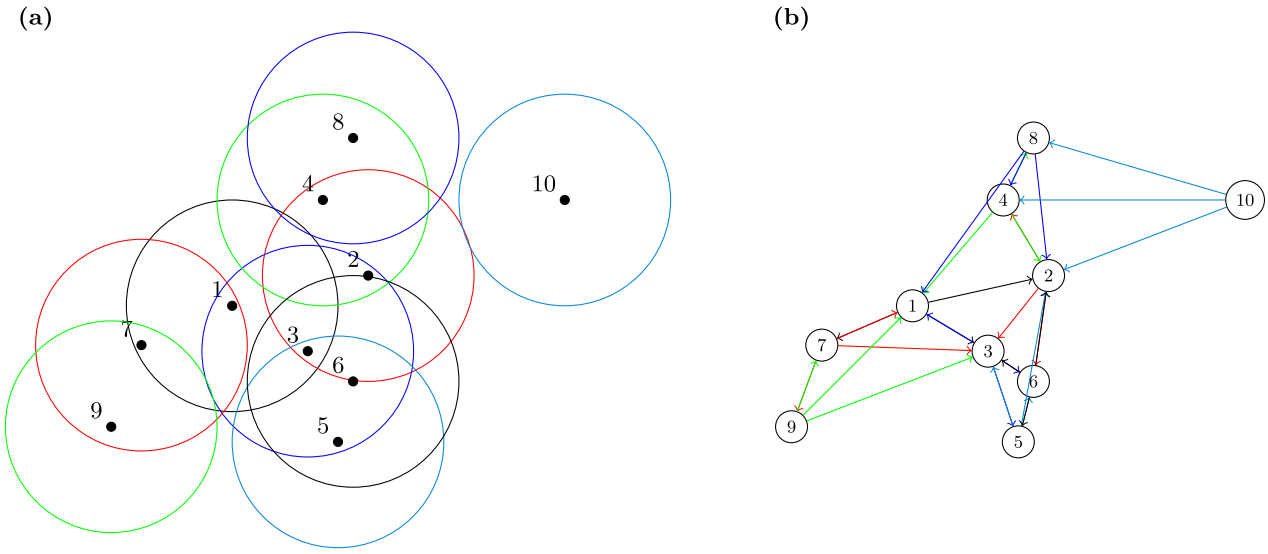


Figure 1. Illustrative example of a $d = 2$ feature space comparing outlier detection approaches for (a) DSSJ with $\epsilon = 0.67$ and (b) a graphic representation of *k*NNSJ with $k = 3$. (a) The points are shown as black dots, where a fixed radius is drawn around each point. (b) The *k*NN graph is shown where a directed edge between two nodes denotes the $k = 3$ neighbors for each point. Since $k = 3$ and there are 10 points, there are a total of 30 directed edges. Colors are used to improve readability in both subfigures. The DSSJ and *k*NNSJ approaches will be described in greater detail in Sections 3.5–3.7.

inadvertently exclude these objects from being detected as outliers, which may occur if a supervised approach is used.

3.1. Terminology

Because this paper includes material from several fields of computer science and astronomy, we outline terminology that is used throughout the paper as follows.

1. *Feature vector/point.* A feature vector is a set of numerical attributes that define an object, e.g., an asteroid (example properties include color, rotation period, and albedo). Because these feature vectors occupy a position in a *d*-dimensional feature space, they are also called *points*. Throughout this paper, we refer to asteroid properties as feature vectors and points.
2. *Data set/database.* A collection of feature vectors is denoted as D. The cardinality of the set (number of objects/asteroids) is denoted as $|D|$.
3. *Feature space.* The space encompassing all of the feature vectors (or points) in the data set, D.
4. *Dimensionality.* The number of features included in each feature vector (d). We examine $d \leqslant 15$.
5. *Self-join.* Performing a search on all feature vectors within a data set (D) as compared to each other.
6. *Distance similarity search.* This is also known as a range query or a radius search, where a search is conducted around a query point using a radius ϵ and all neighbors that are found within ϵ are returned. A distance similarity self-join (DSSJ) performs distance similarity searches on all points in the data set, D.
7. *k-nearest neighbors (kNN) search.* Finding the closest k points within the feature space from a given query point. A *k*NN self-join performs *k*NN searches on all points in a data set, D.
8. *Multi-GPU speedup.* A performance measure that describes the scalability of a GPU algorithm that uses multiple GPUs. It is the ratio of the time to compute on one GPU (T_1) to $T_{n_{GPU}}$, where $T_{n_{GPU}}$ is the time to compute on n_{GPU} GPUs. In this paper we evaluate up to $n_{\mathrm{GPU}} = 4$ and so the maximum possible speedup is $4\times$.
9. *Multi-GPU parallel efficiency.* A performance measure that describes how well a resource is utilized (the GPU in this context). It is the multi-GPU speedup above divided by n_{GPU}, and the maximum value is 1.0 (or 100%).

3.2. Background and GPU-accelerated Outlier Detection Algorithms

Many of the pioneering efforts on outlier detection use the intuitive notion of an outlier, which defines an outlier as having a large fraction of the data set exterior to a search distance around a given point (Knorr & Ng 1997). Subsequent efforts have focused on the notion of neighbors around a point, where the definition of an outlier is a function of the properties of its closest neighbors (Campos et al. 2016). This second class of algorithms that use *k*NN have been shown to outperform other unsupervised learning methods (Campos et al. 2016). Given these two major approaches to outlier detection, we select two methods as follows: (1) DSSJ and (2) *k*-nearest neighbors self-join (*k*NNSJ).

The DSSJ operation performs distance similarity searches on all feature vectors (or points) within a data set. Each distance similarity search returns all points within a search distance, ϵ, of a point. Thus, the "self-join" refers to searching all points within a data set as compared to each other. *kk*NNSJ finds the *k*NN of all points in a data set. Compared to DSSJ, which uses a fixed search radius ϵ, the *k*NN algorithm employs varying search radii for each point in the data set to find at least k neighbors per point. For example, in a densely populated region of the data space, the search radius will be small such that it finds at least k neighbors, whereas in a sparsely populated region, the search radius will need to be larger. Examples of the DSSJ and *k*NNSJ approaches are outlined in Figure 1.

The typical approach for selecting parameters for an outlier detection algorithm (e.g., k in the *k*NNSJ approach) is to use the receiver operating characteristic (ROC; Campos et al. 2016).

However, ROC requires a set of labels that define inliers and outliers. Because we do not have a labeled set of outliers, we elect to use a different approach. We select a search radius ϵ for DSSJ and the number of neighbors k for kNNSJ by reaching a trade-off between overfitting to the local data density or underfitting to the global data density of each feature space. While this computation adds additional complexity to the detection of outliers across all of the feature spaces, it ensures that we are not arbitrarily selecting the parameters ϵ or k.

3.3. Population Outlier Detection and Accelerating Scientific Discovery

One facet of population outlier detection is the examination of multidimensional feature spaces. The features must be normalized to ensure that some properties are not given more weight than others. Otherwise, some features may dominate the feature space, and obfuscate other features from contributing to the degree that an object is considered an outlier. Therefore, we normalized all features to be in the range [0, 1]. We used all of the `SNAPShot1` features outlined in Table 1, where the nonnormalized mean and standard deviation values are reported.

To address a wide range of science cases, and to accelerate the scientific discovery process, we assign objects an outlier ranking across all permutations of the 15 feature spaces (Table 1 shows each feature), where the dimensionality $d \geqslant 2$. In this context, population outlier detection serves two purposes: (1) to filter those objects that are likely to be outliers from the typical population of asteroids and (2) to accelerate the scientific discovery process by providing additional information that may be of interest to a researcher.

Regarding (2) above, the notion of accelerating the scientific discovery process by augmenting a human researcher with machine support has been described in other works, which address providing multiple data variants to a user or guiding the scientific discovery process through data prioritization (Wagstaff et al. 2013; Pankratius et al. 2016). Population outlier detection across numerous multidimensional feature spaces provides both data prioritization and variant exploration.

To enable the permutation approach described above, SNAPS will provide information regarding each permutation of the $d = 15$ features/dimensions where $d \geqslant 2$. Therefore, the total number of feature spaces in `SNAPShot1` is as follows:

$$\sum_{i=2}^{15} \binom{15}{i} = 32,752. \quad (1)$$

Below we describe the methods used for outlier detection and how we rank each object within each feature space.

3.4. Outlier Detection Methods and Object Ranking

We present two outlier detection methods each of which have two different metrics, where each outputs a ranked list of outlier objects. The ranked list method allows us to compare the rankings between outlier detection approaches. Because the different outlier detection methods are likely to return different rankings for each object, this allows us to have several outlier detection criteria that can be used to determine whether a given object should be investigated in greater detail. This may include telescopic follow-up to obtain additional information about an object. This approach allows SNAPS users to determine their own thresholds for decision making.

We let the list of ranked objects for a given method be denoted as R where the rank of object i is denoted as $r_i \in R$ where $i = 1,\ldots,|D|$. Object i with the minimum ranking value $r_i \in R$, argmin($r_i \in R$), has the greatest probability of being an outlier, whereas the object with the maximum value, argmax ($r_i \in R$) is considered the most typical object in the feature space. We highlight that some of the outlier detection methods yield ties between objects, where two or more objects may be assigned the same outlier score, whereas other methods may not produce ties between object rankings.

3.5. Distance Similarity Self-join

The DSSJ method finds all points within a search radius, ϵ, around each point in a data set. Intuitively, points having a significant number of points within ϵ are considered typical, and those with few points within ϵ will be considered outliers.

3.5.1. Ranking Metrics

We outline two ranking metrics for DSSJ as follows. One metric is *distance-oblivious* where all points within the search radius, ϵ, of a query point are considered equal, and thus their distances to the query point are not included in the ranking. The other metric is *distance-aware* where the distances of points to the query point within the search radius ϵ are considered in the ranking calculation. These two metrics yield different rankings for the DSSJ method.

1. *Distance-oblivious.* The ranking function, R, is simply a ranking from the fewest to the greatest number of neighbors each point has within the search distance ϵ.
2. *Distance-aware.* The ranking function, R, is a ranking from the largest mean distance between an object and its points within ϵ and the smallest mean distance. Intuitively, if a given point has its neighbors at large distances, then this implies that it is an outlier.

3.5.2. Overfitting versus Underfitting: On The Selection of The Search Radius ϵ

The selection of ϵ will directly impact the outlier ranking of each object in the data set. Consider two extremes: (1) when $\epsilon \approx 0$, there will be few (if any) neighbors found within ϵ around a given point/feature vector, and (2) as $\epsilon \rightarrow \infty$ each feature vector will find all other feature vectors in the feature space. Both of these cases are impractical for finding outliers. The search radius ϵ must not be too small such that too few neighbors are found, while not being too large such that too many neighbors are found.

Considering the extreme cases above, a small value of ϵ implies that the local density of a feature vector determines whether it is an outlier, whereas a large value of ϵ implies that the global density makes this determination. Thus, in the former case, the data will be overfit to the local density, and in the latter case, the data will be underfit. Here, we provide a simple solution for selecting ϵ, which considers the pragmatic nature of finding outlying objects in the context of large-scale astronomical surveys, by reaching a trade-off between underfitting and overfitting.

Consider two ranked lists, R^A and R^B, and n, which refers to some top fraction of the outliers in each list. In this paper we set $n = 0.01|D|$ (1% of the data set). If the top-n outlying objects in R^A and R^B are permuted and a given object in R^A has a similar

position in R^B, then the object has been found to be an outlier in both ranked lists. Likewise, objects that are not within the top-n objects in both lists are inliers, and are thus typical objects in the data set.

Given the above, we compare the similarity of two ranked lists of outliers, R^A and R^B, by counting the total fraction, $f \in [0, 1]$, of feature vectors belonging in the top-n in both lists, or more formally their intersection:

$$f = |R^A \bigcap R^B| \cdot n^{-1}, \tag{2}$$

where $r_i \in R^A < n$ and $r_i \in R^B < n$.

To reach a trade-off between underfitting and overfitting, we search a grid of evenly spaced ϵ values, G, where each value of ϵ is denoted as ϵ_j where $j = 1, 2, \ldots, |G|$, and $\epsilon_j \in [\epsilon_{\min}, \epsilon_{\max}]$. $\epsilon_{\min}$ and $\epsilon_{\max}$ refer to the minimum and maximum values of ϵ searched, respectively, and their selection will be described later in Section 3.5.3.

For each value of ϵ_j, we derive a corresponding ranking of outliers, denoted as R_j, where R_1 corresponds to the ranking given by $\epsilon_{\min}$ and $R_{|G|}$ corresponds to the ranking given by $\epsilon_{\max}$. $\epsilon_{\min}$ refers to a small search radius, and thus would produce a ranked list that overfits the data. Likewise, $\epsilon_{\max}$ refers to a large search radius that would underfit the data. To find a good value of $\epsilon_j \in [\epsilon_{\min}, \epsilon_{\max}]$, we compute the similarity (using Equation (2)) between R_1 and R_j, and the similarity between $R_{|G|}$ and R_j. These two sets of similarity values, corresponding to those generated as compared to R_1 and $R_{|G|}$, are denoted as $U^{\min}$ and $U^{\max}$, respectively, where $|U^{\min}| = |U^{\max}| = |G|$.

Finally, the selected value of ϵ is given by the value of j that minimizes the difference in similarity values, which is described below:

$$\epsilon^{\text{select}} = \epsilon_{argmin(|U_j^{\min} - U_j^{\max}|)}. \tag{3}$$

Illustrative example. To better illustrate these concepts, consider the following concrete example that computes $U^{\min}$ and $U^{\max}$ with $|G| = 3$ (i.e., for illustrative purposes we only consider computing the fraction of points that match for three values of ϵ). Here, $|G| = 3$ implies that $j = 1$, 2, 3, and that $\epsilon_{\min} = \epsilon_1$ and $\epsilon_{\max} = \epsilon_3$.

The set of top-n outlying points found by ϵ_1, ϵ_2, and ϵ_3 are as follows, where we set $n = 10$. We color code the points that are common between sets:

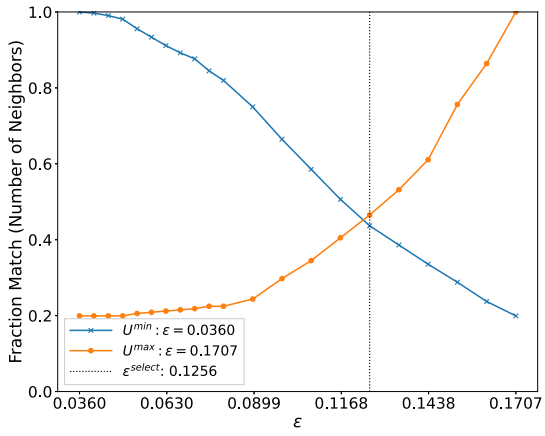
$$\epsilon_1 = \epsilon_{min} = \{1, 2, 3, 4, 5, 16, 17, 18, 19, 20\},$$
$$\epsilon_2 = \{1, 2, 3, 4, 8, 9, 10, 100, 101, 102\},$$
$$\epsilon_3 = \epsilon_{max} = \{1, 2, 8, 9, 10, 11, 12, 13, 14, 15\}.$$

Using Equation (2) we compute $U^{\min}$ and $U^{\max}$, which compares the similarity between the top-n points found by $\epsilon_{\min}$ and $\epsilon_{\max}$, respectively. Specifically, $U^{\min}$ is computed by comparing ϵ_1 to those found by ϵ_1 (itself), ϵ_2, and ϵ_3. $U^{\max}$ is computed by comparing ϵ_3 to ϵ_1, ϵ_2, and ϵ_3 (itself). We highlight that because we compare the same set to itself, this guarantees that there is a 1.0 match fraction in $U^{\min}$ and $U^{\max}$.

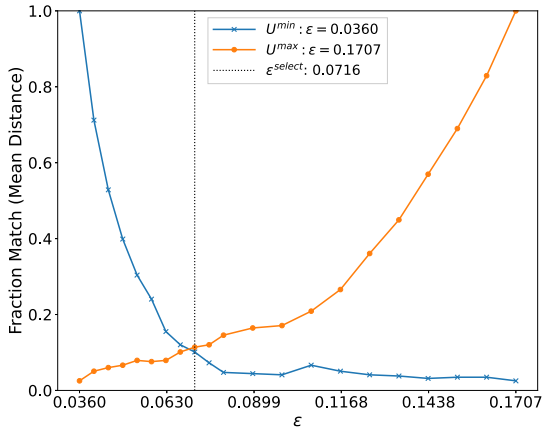
Using the example values above and Equation (2) we compute $U^{\min}$ and $U^{\max}$ as follows:

$$U^{\min} = \{10/10, 4/10, 2/10\} = \{1.0, 0.4, 0.2\},$$
$$U^{\max} = \{2/10, 5/10, 10/10\} = \{0.2, 0.5, 1.0\}.$$

Observe that because we compare the smallest value of ϵ to increasingly larger values of ϵ, the match fraction declines. And similarly because we compare the largest value of ϵ to increasingly larger values of ϵ, the match fraction increases.



(a)



(b)

Figure 2. Example plots showing computing the best value of ϵ on an example $d = 5$ feature space, which are the first five features in Table 1. $U^{\min}$ ($U^{\max}$) refers to the fraction (f) of the top-n objects that are in both ranked lists for $\epsilon = 0.0360$ ($\epsilon = 0.1707$) and the ϵ values on the horizontal axis. The value of ϵ^{select} is shown as the vertical dotted line for (a) the number of neighbors metric (*distance-oblivious*), and (b) the mean distance metric (*distance-aware*).

This produces an intermediate value of ϵ that will be selected that reaches a trade-off between the smallest and largest ϵ values.

Figure 2 shows a plot of this selection procedure on one of the feature spaces, where (a) shows the number of neighbors metric and (b) shows the mean distance metric (as defined in Section 3.5.1). The plot shows how similar the rankings are between the ϵ value on the horizontal axis as compared to the smallest ($U^{\min}$) and largest ($U^{\max}$) ϵ values, which are those that are expected to overfit and underfit the data, respectively. Thus, as ϵ increases, the fraction match for $U^{\min}$ decreases, whereas it increases for $U^{\max}$. A trade-off for ϵ^{select} is reached where $\epsilon_{\min} < \epsilon^{\text{select}} < \epsilon_{\max}$. Note that ϵ is evenly sampled on the horizontal axis, which is somewhat misleading. While the axis shows *linear* increases in ϵ, this leads to a *nonlinear* increase in the search volume, which is a function of the data dimensionality. Thus, it is similar to a logarithmic sampling of the search volume on the horizontal axis. However, fine-grained sampling is unnecessary at high values of ϵ, as a large value of ϵ^{select} is unlikely to minimize the difference between U_j^{min} and U_j^{max} (Equation (3)). We will examine this in more detail when we show the distribution of ϵ^{select} values across all feature spaces in Section 4.2.

In this section, we omitted how $\epsilon_{\min}$ and $\epsilon_{\max}$ are selected. In what follows, we describe how we select these values.

3.5.3. Selection and Computation of the ϵ Search Grid

A major challenge of using DSSJ for outlier detection is that the values of $\epsilon_{\min}$ and $\epsilon_{\max}$ vary for each feature space. For instance, with lower-dimensional data, the search radius needed to find an average number of neighbors per point is smaller than for high-dimensional data, as the total volume of the feature space increases exponentially with the number of dimensions. Furthermore, since the data in the feature spaces are unlikely to follow any well-defined data distributions, the values of ϵ need to be found numerically as they cannot be found analytically.

To address this, we define selectivity, which is the mean number of neighbors within ϵ found in a data set for DSSJ (Gallet & Gowanlock 2021). The definition is $s = (|A| - |D|) \cdot |D|^{-1}$ where $|D|$ is the number of feature vectors and $|A|$ is the total result set size (the total number of pairs of points within ϵ of each other) of the self-join operation. Thus, it computes the mean number of neighbors within ϵ in a data set, excluding a point/feature vector finding itself. When processing `SNAP-Shot1`, we set $s_{\min} \approx 0.001|D|$ and $s_{\max} \approx 0.15|D|$, corresponding to a minimum and maximum selectivity of 0.1% and 15% of the data set. The results of DSSJ using these two selectivity values will overfit and underfit the data, respectively.

To find the values of $\epsilon_{\min}$ and $\epsilon_{\max}$ that yield the minimum and maximum selectivity values described above, we use the method outlined in Gowanlock (2021), which was utilized to select a search distance ϵ that finds on average at least k neighbors for each point in a data set for the kNNSJ algorithm. The method is lightweight as it samples the data set to first estimate the mean distance between points in a data set (ϵ_{mean}), which is used as the upper bound on a practical search radius to find $\epsilon_{\max}$ (or $\epsilon_{\min}$). Then, the distances between a sample of points and points within the data set are computed to create a histogram of distances between points. Using this histogram, a cumulative distance distribution histogram is computed, which yields a relationship between the search distance and the average number of neighbors that will be found per point in the data set. We use this histogram to select the search radii $\epsilon_{\min}$ and $\epsilon_{\max}$. We refer the reader to Gowanlock (2021) for further details.

After $\epsilon_{\min}$ and $\epsilon_{\max}$ are found, the ϵ grid G is computed, which is used to derive the rankings for all feature vectors in the feature space and a value of ϵ is selected that reaches a trade-off between overfitting and underfitting (Section 3.5.2). We use a computationally efficient approach to compute the result sets (A) for each $\epsilon_j \in G$. Consider that the result set A for $\epsilon_{\max}$ contains all of the neighbors for lower values of ϵ as well. Thus, we execute DSSJ once for $\epsilon_{\max}$ and use this result set to derive the result sets for all $\epsilon_j \in G$. Figure 3 shows an example of three search radii (ϵ_1, ϵ_2, and ϵ_3), where $\epsilon_1 = \epsilon_{\min}$ and $\epsilon_3 = \epsilon_{\max}$. The figure shows that all result sets for $\epsilon_j < \epsilon_{\max}$ can be derived from $\epsilon_{\max}$. Thus, independently executing DSSJ for all $\epsilon_j \in G$ is unnecessary (and would be very computationally expensive), as the result set for $\epsilon_{\max}$ can be filtered to compute the result sets for smaller search distances.

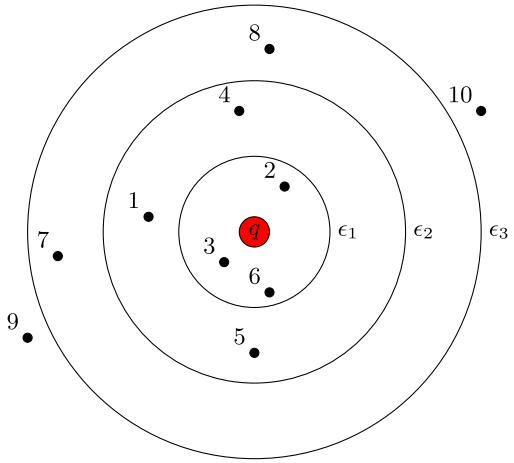


Figure 3. Illustration of several result sets for query point q in a $d = 2$ feature space, where data points are shown as black dots and q is shown as the red dot. Three evenly spaced search distances are shown (ϵ_1, ϵ_2, and ϵ_3), which are illustrated as circles centered on query point q. The result sets (A) are as follows: $A(\epsilon_1) = \{2, 3, 6\}$; $A(\epsilon_2) = \{1, 4, 5\} \cup A(\epsilon_1)$; and $A(\epsilon_3) = \{7, 8\} \cup A(\epsilon_1) \cup A(\epsilon_2)$. Points 9 and 10 are not within any of the result sets. Thus, the result sets for ϵ_1 and ϵ_2 can be directly derived from that of ϵ_3.

3.6. k-nearest Neighbors Self-join

Intuitively, the kNNSJ algorithm can be employed for unsupervised outlier detection where an outlier is characterized as having a significant number of its kNN located at large distances.

3.6.1. Ranking Metrics

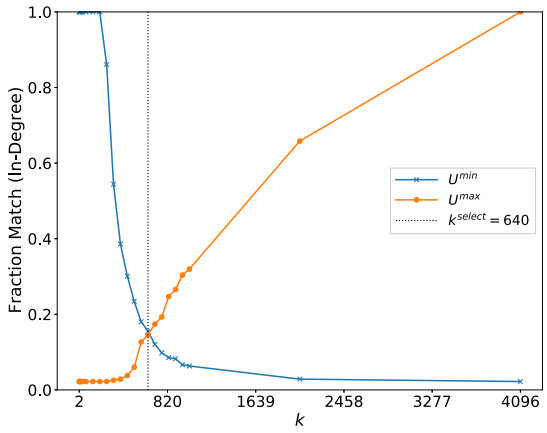
Similarly to DSSJ, the kNNSJ algorithm has two ranking metrics outlined as follows, where one metric is *distance-oblivious* and one is *distance-aware*. This yields two different rankings for the kNNSJ method.

1. *Distance-oblivious.* The ranking function R is simply an in-degree ranking of the kNN graph (Hautamaki et al. 2004), which is also known as its reverse nearest neighbors. The in-degree refers to the number of occurrences that a point is found within another point's set of kNN. The fewer the number of occurrences, the more likely the point is an outlier.
2. *Distance-aware.* The ranking function is a ranking of mean distances between a point and its kNN. Intuitively, if a given point has its k neighbors at large distances, then this implies that it is an outlier.

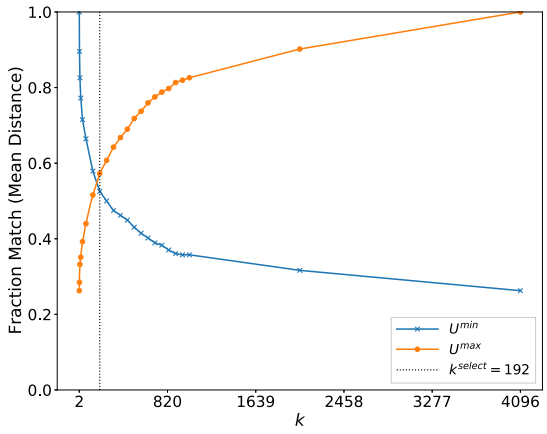
3.6.2. On The Selection of k and the k-nearest Neighbors Search Grid

For brevity we briefly provide an overview of the procedure for kNNSJ because the method is analogous to that of DSSJ. We highlight similarities and key differences between outlier detection with kNNSJ compared to DSSJ.

Similar to DSSJ, the detection of outliers with the kNNSJ algorithm is dependent on the value of k. Unlike the DSSJ algorithm, it is more straightforward to detect outliers with the kNNSJ algorithm because we do not need to perform a search to find $\epsilon_{\min}$ and $\epsilon_{\max}$. A low value of k will overfit the data and a large value of k will underfit the data. We reach a trade-off using an analogous method to DSSJ, which searches a grid of k values where similarity is measured as compared to $k_{\min}$ and $k_{\max}$, and the value of k which reaches a trade-off between



(a)



(b)

Figure 4. Example plots showing the computation of the best value of k using $k_{\min} = 2$ and $k_{\max} = 4096$ on the same feature space shown in Figure 2. $U^{\min}$ ($U^{\max}$) refers to the fraction (f) of the top-n objects that are in both ranked lists for $k = 2$ ($k = 4096$) and the k values on the horizontal axis. The value of k^{select} is shown as the vertical dotted line for (a) the in-degree metric (*distance-oblivious*), and (b) the mean distance to each point's k neighbors (*distance-aware*).

under- and overfitting is selected. Likewise, we use a computationally efficient approach to compute the grid of k values, denoted as $k_j \in G$ where $j = 1, 2, \ldots, |G|$. For each value of k_j, we compute the k neighbors for $k_{\max}$, which contain the neighbors for all $k_j < k_{\max}$. When processing `SNAPShot1`, we set $k_{\min} = 2$ and $k_{\max} = 4096$.

Figure 4 shows a plot of this selection procedure on one of the feature spaces, where (a) shows the in-degree metric and (b) shows the mean distance metric (as defined in Section 3.6.1). The plot shows how similar the rankings are between the k value on the horizontal axis as compared to the smallest and largest values of k ($k_{\min} = 2$ and $k_{\max} = 4096$), which are those that are expected to overfit and underfit the data, respectively. Thus, as k increases, the fraction match for $U^{\min}$ decreases, whereas it increases for $U^{\max}$. A trade-off for k^{select} is reached where $k_{\min} < k^{\text{select}} < k_{\max}$. Note that k is unevenly sampled on the horizontal axis. This is because it is unlikely that k^{select} will be large, so we do not perform a fine-grained sampling at larger values of k. Furthermore, we find that k^{select} is typically bound to a particular range of k values for the in-degree and mean distance metrics, and so we finely sample those regions. We will show the distribution of k^{select} across all feature spaces in Section 4.2.

Table 2
The Metric Values (Rankings) for the Example Shown in Figure 1

Point	DSSJ No. of Neighbors	DSSJ Mean Distance	kNN In-Degree	kNN Mean Distance to k Neighbors
1	2 (2)	0.6185 (1)	5 (4)	0.7197 (5)
2	2 (2)	0.6117 (3)	6 (5)	0.6435 (7)
3	4 (3)	0.5541 (5)	6 (5)	0.5254 (8)
4	2 (2)	0.5196 (7)	3 (3)	0.6537 (6)
5	2 (2)	0.5224 (6)	2 (2)	0.7209 (4)
6	2 (2)	0.3864 (9)	3 (3)	0.4933 (9)
7	2 (2)	0.6149 (2)	2 (2)	0.7768 (3)
8	1 (1)	0.4562 (8)	2 (2)	0.9133 (2)
9	1 (1)	0.5758 (4)	1 (1)	1.0334 (1)
10	0 (0)	0.0000 (0)	0 (0)	1.4839 (0)

Note. A ranking of 0 indicates the most outlying object.

3.7. Summary: Comparison of Outlier Approaches

The DSSJ and kNNSJ methods each have two metrics which yield four total outlier detection rankings for each object (Sections 3.5.1 and 3.6.1). We briefly compare and contrast these methods; however, their characteristics will be better understood with an example on real-world data that we will show in Section 4.4.

Figure 1 shows illustrative examples of (a) DSSJ and (b) kNNSJ, where the positions of the points are identical in both subfigures. Table 2 shows the corresponding metric values and the resulting rankings for each. Observe that the rankings for each method vary. Only point 10 has the same rank (0) across the four methods, because it has no neighbors within ϵ and it has an in-degree value of 0. Furthermore, the DSSJ number of neighbors metric and the kNNSJ in-degree metric have numerous rankings that are the same between points because the values are nonnegative integers. In contrast, the DSSJ mean distance and kNN mean distance to k neighbors metrics are real numbers so the probability that two values are the same is low, and therefore it is unlikely that two points will share the same rank. The only exception is when no neighbors are found within the search radius ϵ for the DSSJ mean distance metric (e.g., point 10 in Figure 1), in which case it will be assigned a rank of 0. From this example it appears that there is little consensus among the outlier detection techniques; however, this is primarily because there are only 10 points in the example. On real-world data there is much more consensus between the four rankings, and this will be discussed in greater detail in Section 4.4.

3.8. High-level Population Outlier Detection System: Description and Optimizations

In the prior sections, we discussed the outlier detection methods. Here, we describe the GPU-accelerated implementations that are used in the system, and the integration of the constituent components of the system. The implementation uses shared libraries written in C/C++/CUDA using Python (NumPy) interfaces.

3.8.1. GPU-accelerated Implementations

We employ two state-of-the-art DSSJ and *k*NNSJ algorithms, both of which use GPU acceleration. Gowanlock & Karsin (2019) proposed a DSSJ algorithm for moderate- to high-dimensional data. To enable searching in high-dimensional feature spaces and avoid the curse of dimensionality problem[5] where index searches become increasingly exhaustive, Gowanlock & Karsin (2019) proposed indexing the data in c dimensions, where $c < d$. Thus, a representation of the data in c dimensions avoids exhaustive index searches in high dimensions. We configure our software to index in $c = d$ dimensions when $d \leqslant 6$, and $c = 6$ when $d > 6$. For example, when we process the feature space with the greatest dimensionality ($d = 15$), we only index $c = 6$ dimensions.

Several optimizations were subsequently made to the algorithm outlined in Gowanlock & Karsin (2019), including exploiting instruction-level parallelism and changing the order in which query points are processed to decrease load imbalance in the GPU kernel as outlined by Gowanlock et al. (2023). This optimized algorithm is employed in our software.[6]

Gowanlock (2021) describes a hybrid CPU + GPU *k*NNSJ algorithm for low-dimensional data, where $d \lesssim 8$. The algorithm distributes query points to either the CPU or GPU based on the expected amount of work required of each query point. In short, query points in low-density regions are assigned to the CPU and those in high-density regions are assigned to the GPU. Because query points in high-density regions require a significant number of distance calculations to refine the candidate set (those points returned by an index but that need to be refined using distance calculations), it is preferable to assign these queries to the GPU instead of the CPU because the former architecture has much higher throughput for this operation. The other facet of distance calculation complexity is the data dimensionality, d. For moderate- to high-dimensional data, the GPU significantly outperforms the CPU, and so there are cases where the CPU + GPU algorithm yields an insignificant performance gain over a GPU-only approach. Consequently, because the data we are using have $d = 15$ dimensions, it is preferable to use the GPU-only version of the algorithm proposed by Gowanlock (2021). While not shown in that paper, the same method described for the DSSJ above is employed, which indexes the data in $c < d$ dimensions to allow for *k*NN searches in high dimensions. We use the publicly available source code of this algorithm.[7]

3.8.2. Computational Optimizations

We describe several optimizations that are used to improve the performance of the system. These methods are presented independently from the results, but they will be of particular interest when interpreting the results. Thus, the reader may wish to read Section 4 and then return to this section later.

Multiprocessing each feature space. As described in Section 3.3, there are 32,752 total feature spaces across $d = 2 - 15$ dimensions. We use the multiprocessing library in Python to offload the processing of each of these feature spaces. Thus, each feature space is computed independently and concurrently with the others.

Oversubscription of GPU resources. It is well known that there is overhead when communicating between the host (which contains the CPU and main memory) and the GPU (Capodieci et al. 2017). For instance, the *k*NNSJ and DSSJ methods will need to transfer their respective result sets from the GPU back to the main memory. During this time, the GPU will not have anything to compute and so the resource will be underutilized. Underutilizing the GPU will significantly increase the total time needed to compute all of the feature spaces.

To address the underutilization problem, we oversubscribe the GPU and assign a maximum of o processes to the GPU at a given time. Therefore, when one processes finishes computing its task (a GPU kernel invocation) and starts transferring its results back to the host, another process can start executing its GPU kernel. The oversubscription value, o, cannot be too large, otherwise the GPU global memory capacity will be exceeded, and on our platform the global memory of a single GPU is 40 GiB. This will be described in greater detail in Section 4.3.

Multi-GPU scalability. Because the feature spaces are independent, it is straightforward to assign a feature space to a given GPU, where either the DSSJ or *k*NNSJ algorithms are executed on the feature space. Thus, our system exploits the four GPUs on our hardware platform (the platform is described in additional detail in Section 4.1). Because the feature spaces are of varying dimensionality, the amount of work needed to compute each feature space varies. We schedule the computation of feature spaces to GPUs using dynamic scheduling, where the least-loaded GPU is assigned the next feature space. This yields good load balancing across the four GPUs. We describe multi-GPU scalability in greater detail Section 4.3.

Nested parallelism of tasks across the CPU and GPU. Recall from Sections 3.5.3 and 3.6.2 that we require finding a good value of ϵ and k for DSSJ and *k*NNSJ, respectively, such that we do not arbitrarily select a value for these parameters that underfits or overfits the data. We use the result set for $\epsilon_{\max}$ and $k_{\max}$ to derive all result sets for lower values of ϵ_j and k_j, respectively. The GPU computes these result sets, and transfers them back to the host, where we then compute the result sets for the smaller values of $\epsilon_j < \epsilon_{\max}$ and $k_j < k_{\max}$ in parallel on the CPU. This allows the CPU to contribute to the overall computation instead of simply orchestrating data transfers to/from the GPU and performing other minor host-oriented tasks that are required for GPU computation.

4. Results

4.1. Experimental Methodology

Our platform consists of two Intel(R) Xeon(R) Platinum 8358 CPUs (64 total physical cores) with a base clock speed of 2.60 GHz and 512 GB of main memory. The platform is equipped with four Nvidia A100 GPUs, each of which has 40 GB of global memory. The source code uses the following programming languages: Python, C/C++, and CUDA. All host C/C++ code is compiled with the O3 compiler optimization flag, and all CUDA code was compiled with CUDA v11.7. The C/C++ and CUDA code uses 32 bit floating point precision. All performance-related experiments use an average of three time trials; however, the variance in the time measurements is insignificant, and error bars on these

[5] See Zimek et al. (2012) for an overview of the curse of dimensionality problem.
[6] https://github.com/mgowanlock/gpu_self_join
[7] https://github.com/mgowanlock/hybrid_k_nearest_neighbor_self_join

Table 3
Parameters Used throughout the Evaluation

Parameter	Description
n ($0.01\|D\|$)	The number of points considered when computing the similarity between two ranked lists. 1% of the data set are used as we are interested in selecting those objects that are outliers.
G	The grid of ϵ or k values that are searched for DSSJ and kNNSJ, respectively.
$s_{\min}$ ($\approx 0.001\|D\|$)	The minimum selectivity used when defining the ϵ grid.
$s_{\max}$ ($\approx 0.15\|D\|$)	The maximum selectivity used when defining the ϵ grid.
ϵ^{select}	The selected value of ϵ that reaches a trade-off between underfitting and overfitting the feature space.
$k_{\min}$ (2)	The minimum value of k used when defining the grid of k values.
$k_{\max}$ (4096)	The maximum value of k used when defining the grid of k values.
k^{select}	The selected value of k that reaches a trade-off between underfitting and overfitting the feature space.
o	The oversubscription factor, which is the maximum number of processes that are assigned to one GPU at a time.

Note. The values of the parameters are given in parentheses where applicable.

plots would be too small to observe so we do not report this information.

The platform is located within our institution's computer cluster, Monsoon, and we ensure that our experiments are carried out without interference from other users running on the node. The population outlier detection pipeline requires storing output (i.e., writing the rankings for each feature space to disk and diagnostic plots), and this occurs over the network where files are written to a parallel file system. We include this and all other overheads in the end-to-end response time such that we accurately capture the performance of the system.

For convenience, Table 3 summarizes the parameters used throughout the evaluation, which were defined in prior sections. Some parameters will be varied in the sections that evaluate the performance of the algorithms.

4.2. Distribution of the Selected Values of ϵ and k

We begin by examining the selected values of ϵ for the DSSJ algorithm. Figure 2 showed selecting a good value of ϵ on a single feature space by reaching a trade-off between underfitting and overfitting the data. Here, we present the distribution of ϵ^{select} values across all 32,752 feature spaces for the number of neighbors and mean distance metrics. In other words, we show the resulting ϵ^{select} values from an analogous analysis to that shown in Figure 2, but summarized across all feature spaces.

Note that it is not possible to show the distribution using the ϵ^{select} values because the range $[\epsilon_{\min}, \epsilon_{\max}]$ varies across each individual feature space. Thus, we show the index j of the selected value of $\epsilon_j \in G$, denoted as $\arg(\epsilon^{\text{select}})$. For example, in Figure 2(a), instead of reporting $\epsilon^{\text{select}} = 0.1256$, we report $\arg(\epsilon^{\text{select}}) = 15$.

Figures 5 and 6 plot $\arg(\epsilon^{\text{select}})$ as a function of the dimensionality of the feature spaces for the number of neighbors and mean distance metrics, respectively. Across all feature spaces, we find that ϵ^{select} is not found toward the extremes of the distribution of ϵ^{select} values for either metric, indicating that the range of selectivity parameters given by $s_{\min}$ and $s_{\max}$ do not need to be expanded to find a suitable ϵ^{select} for each feature space. Comparing Figures 5 and 6, on average, the former requires selecting a larger ϵ^{select} than the latter. We attribute this to the difference in ranking functions. The ranking function for the number of neighbors metric allows two or more points to share the same rank if they have the same number of neighbors within ϵ. Thus a larger ϵ allows for the rankings to be more differentiated (i.e., fewer points have the same rank as ϵ increases), and so the under/overfitting trade-off is reached at a larger value of ϵ. In contrast, the mean distance metric will not have two (or more) points with the same rank, as the probability of two points having the same mean distance to its neighbors is unlikely. This also explains why the distribution of $\arg(\epsilon^{\text{select}})$ is clustered in Figure 5, whereas the distribution is more diffuse in Figure 6.

We examine the distribution of the selected values of k for the kNNSJ algorithm. Figures 7 and 8 plot k^{select} as a function of the dimensionality of the feature spaces for the in-degree and mean distance metrics, respectively. The range of k values considered using the under/overfitting methodology is given by $k_{\min}$ and $k_{\max}$, which yields the range [2, 4096]. For both metrics, the selected values of k are well below $k_{\max} = 4096$, and at most k^{select} is found within the first $\sim$25% of the distribution at $k^{\text{select}} \lesssim 1000$. We observe similar trends between the in-degree metric (Figure 7) and the number of neighbors metric for DSSJ (Figure 5) where the distribution of k^{select} is clustered, and this is due to points having the same rank if they have the same in-degree.

4.3. Performance Results

We report the performance of the system by examining two metrics: (1) how performance varies as a function of oversubscription (Section 3.8.2) and (2) how performance varies as a function of the number of GPUs (Section 3.8.2). As outlined in Table 3, we use $s_{\max} \approx 0.15|D|$ for DSSJ and $k_{\max} = 4096$ for kNNSJ. These values largely dictate the amount of work computed, and these values constitute a large fraction of the size of the `SNAPShot1` data sets, which only contains $|D| = 31{,}693$ objects. Thus, we selected these parameters conservatively to show the worst-case performance of the system, as it is clear from Figures 5–8 that ϵ^{select} and k^{select} are found well below $\epsilon_{\max}$ and $k_{\max}$, respectively.

Recall that the oversubscription factor refers to the maximum number of processes that can be assigned to one GPU at a time. Because using the GPU requires data transfers between the host, and other host-side tasks be performed before and after the main GPU kernels are executed, it is beneficial to allow multiple processes to concurrently access the GPU (and associated host-side functionality). For instance, this allows PCIe data transfers to be overlapped with GPU computation, thus mitigating this data transfer cost.

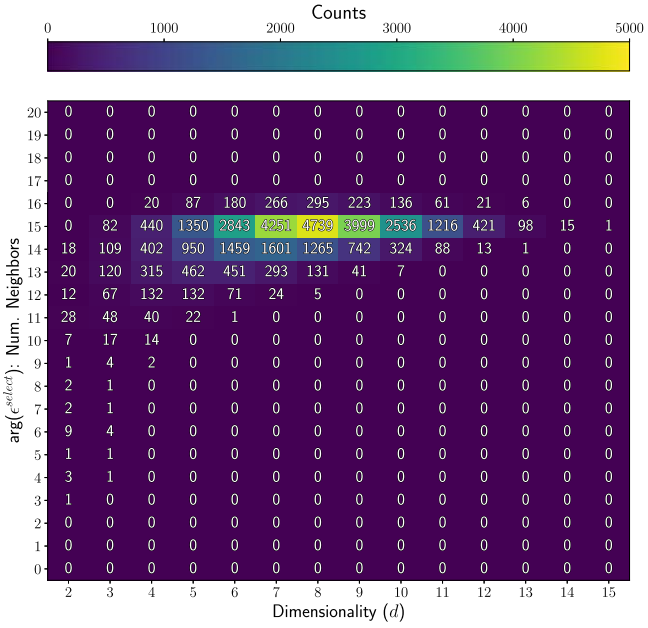


Figure 5. The distribution of ϵ^{select} values for the number of neighbors metric across all 32,752 feature spaces (Section 3.3). Because the ϵ values can vary significantly between feature spaces, where ϵ tends to increase with dimensionality, we show the index j of the selected value of $\epsilon_j \in G$, denoted as arg(ϵ^{select}).

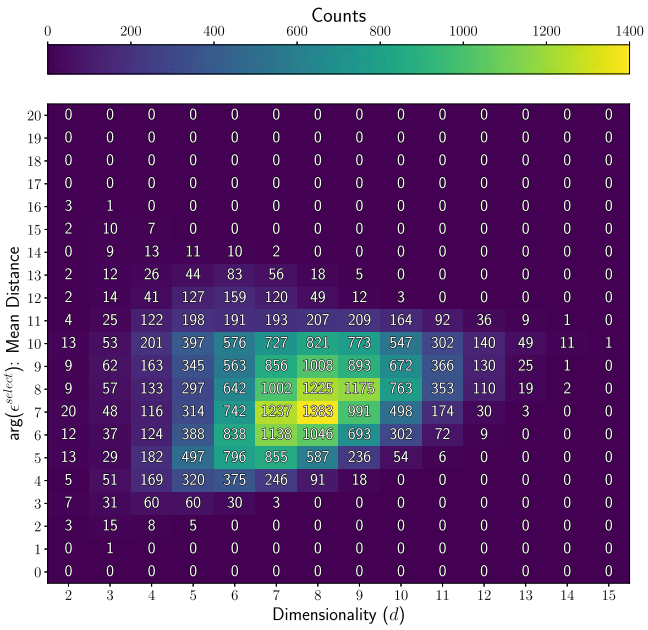


Figure 6. The same as Figure 5, but for the mean distance metric.

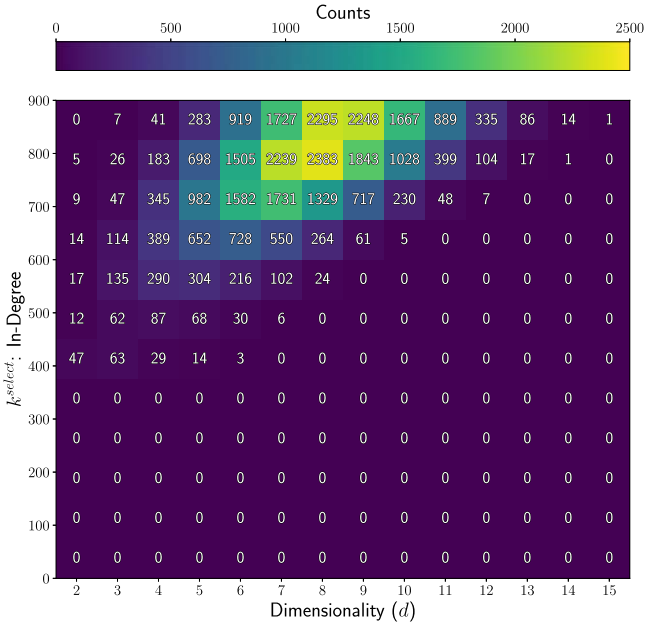


Figure 7. The distribution of k^{select} values for the in-degree metric across all 32,752 feature spaces (Section 3.3). To improve readability, the vertical axis is truncated and does not show all k^{select} values, which are mostly zeros. The plot shows 99.5% of the k^{select} values across all feature spaces.

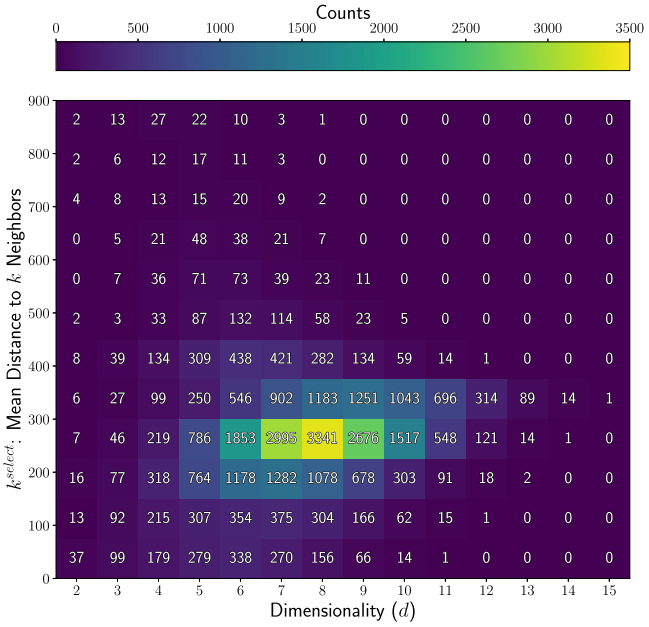


Figure 8. The same as Figure 7, but for the mean distance to k neighbors metric. The plot shows 98.5% of the k^{select} values across all feature spaces.

Figure 9 plots the response time (hr) using one GPU as a function of the oversubscription factor (o) to find ϵ and compute the outlier rankings for DSSJ. From the figure we find that with $o = 1$ (a maximum of a single process concurrently accessing the GPU), the response time is 12.65 hr. However, if we use $o = 8$ the response time is 2.64 hr, which is 379% faster than when no oversubscription is used ($o = 1$). This demonstrates that the GPU is underutilized when $o = 1$. Furthermore, we find that selecting $o = 8-12$ achieves similar performance, but when $o = 16$ the response time begins to increase.

We find that using more than one GPU for DSSJ does not improve performance. This is because there are significant host-side (CPU/main memory) tasks that are computed before the GPU kernel is executed and after the GPU returns the results to the host. Thus, the limiting factor for computing DSSJ is not insufficient GPU resources, rather it is due to memory pressure and contention, which delay the execution of GPU kernels. This is largely a result of using Python as the "glue" programming language, where the bottleneck is not computation, but processing minor tasks in Python that require formatting the data needed for the C/C++ libraries and transferring results back to Python for further analysis.

We present the performance results for *k*NNSJ. The *k*NNSJ algorithm is more computationally expensive than DSSJ, as the

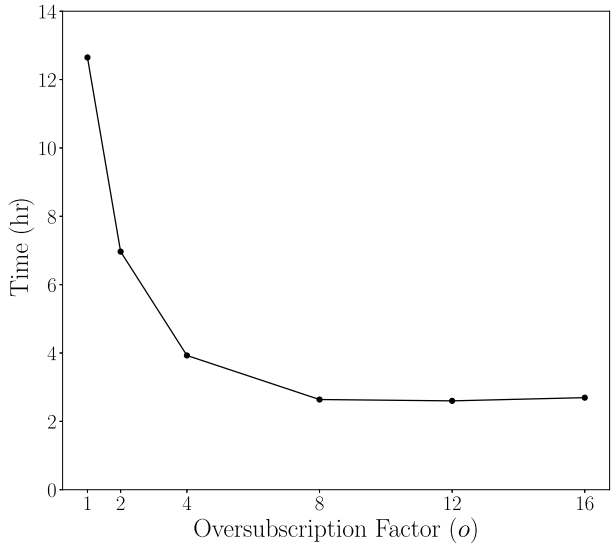


Figure 9. The response time (hr) to compute ϵ^{select} and the corresponding rankings for both DSSJ metrics as a function of the oversubscription (o) factor using one GPU. This includes the total time to find ϵ^{select} for both outlier detection metrics, and compute the rankings for all $|D| = 32{,}752$ feature spaces.

search radius needed to find at least k neighbors will vary for each point in the data set. For this reason, compared to DSSJ the kNNSJ algorithm achieves performance gains when multiple GPUs are used.

Figure 10(a) plots the response time (hr) for kNNSJ as a function of the oversubcription factor (o) using four GPUs. For the kNNSJ algorithm, we find that oversubscription significantly improves performance over not using oversubscription where using $o = 5$ is 83% faster than not using oversubscription ($o = 1$).

Figure 10(b) shows the response time (hr) as a function of the number of GPUs where we set $o = 5$ (the value of o that achieved the best performance in panel (a)). The speedup for 2–4 GPUs is 1.81×, 2.25×, and 2.47×, respectively. We find that multi-GPU scalability is limited: a respectable performance gain is observed with two GPUs, but poor scalability is observed with three and four GPUs. This is due to the same reasons described above for DSSJ where host-side tasks limit multi-GPU scalability.

As we will show in Section 4.5, the multi-GPU performance is much better when examining the much larger `LSST5M` data set.

4.4. Visualization and Comparison of Outlier Detection Methods

In this section, we show example visualizations of feature spaces and the detection of outliers. The visualizations illustrate the importance of using an ensemble of algorithms for outlier detection, as they all yield different outlier rankings, which make some methods more or less suitable for a given scientific investigation. Because it is difficult to visualize feature spaces in more than three dimensions, we limit the visualizations to $d \leqslant 3$.

Figure 11 shows a $d = 2$ feature space (rotation period versus $g - r$ color) for the DSSJ method in panels (a)–(b) and the kNNSJ method in (c)–(d). In the figure, we arbitrarily denote 1% of the points as outliers; in practice a user can select any

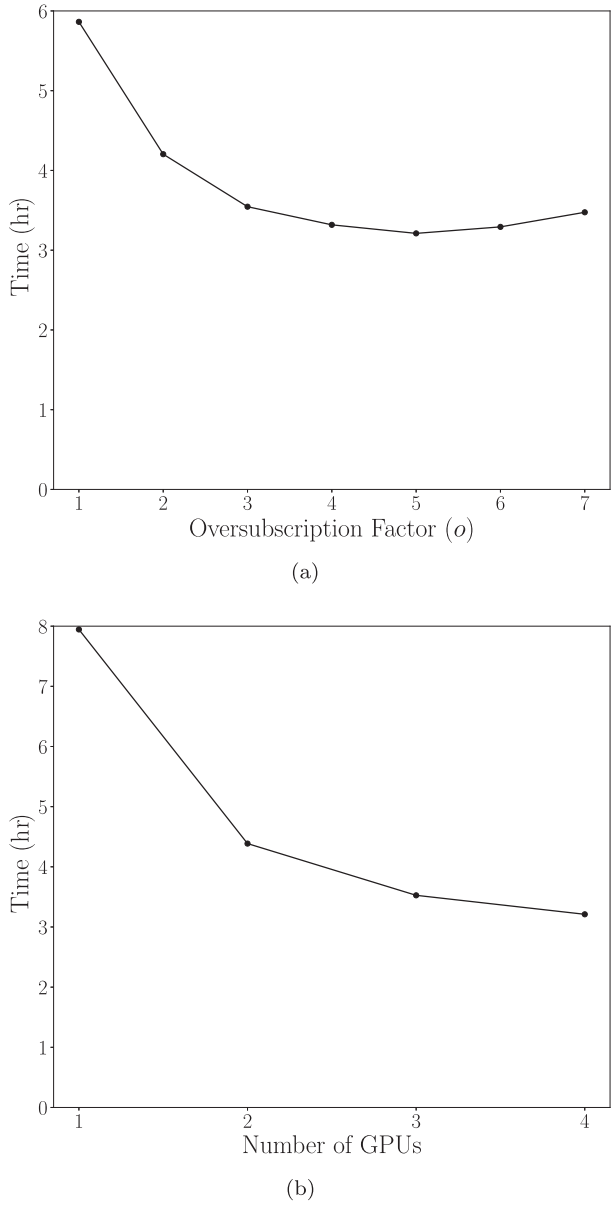
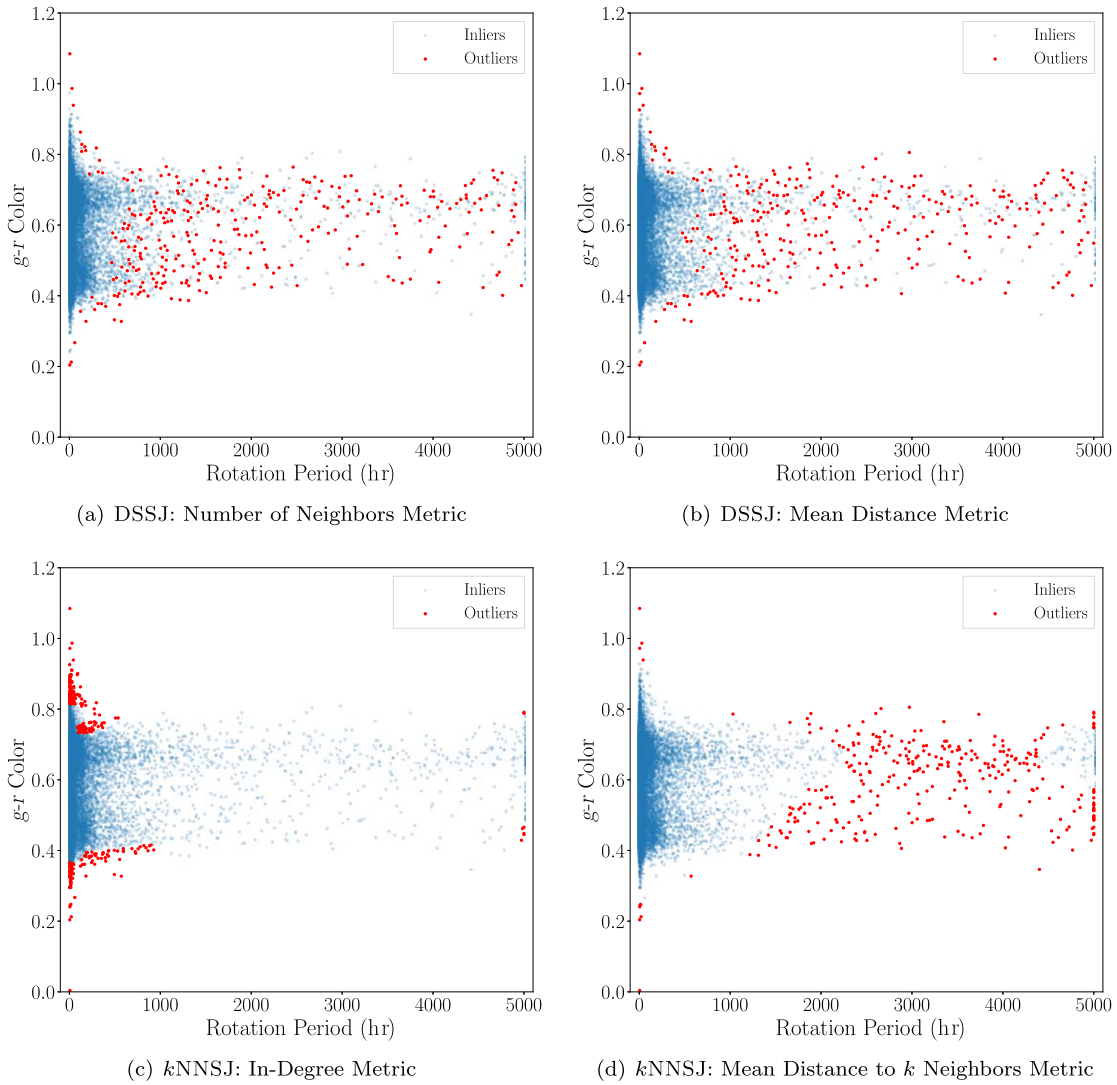


Figure 10. Performance results for kNNSJ showing the total time to find k^{select} for both outlier detection metrics, and compute the rankings for all $|D| = 32{,}752$ feature spaces. (a) The response time (hr) to compute k^{select} and the corresponding rankings for both kNNSJ metrics as a function of the oversubscription (o) factor with four GPUs. (b) The response time as a function of the number of GPUs, demonstrating the scalability when $o = 5$.

percentage of points to be outliers. Furthermore, Figure 12 shows histograms of the metric values that are used to compute the outlier rankings for the corresponding plots in Figure 11. This information is helpful for understanding why outliers are selected in each case. We make the following observations.

1. Figure 11(a) (DSSJ: number of neighbors metric) clearly finds points in the underdense regions of the feature space, but they are interspersed with other points that are considered inliers in this space. Had a larger outlier threshold been selected, more points in the sparse regions would be selected as outliers. This is because the method allows for two or more points to share the same rank. In this case, >1% of the points have a rank of 0 (they have no neighbors), but only the first 1% of these points are



(a) DSSJ: Number of Neighbors Metric

(b) DSSJ: Mean Distance Metric

(c) kNNSJ: In-Degree Metric

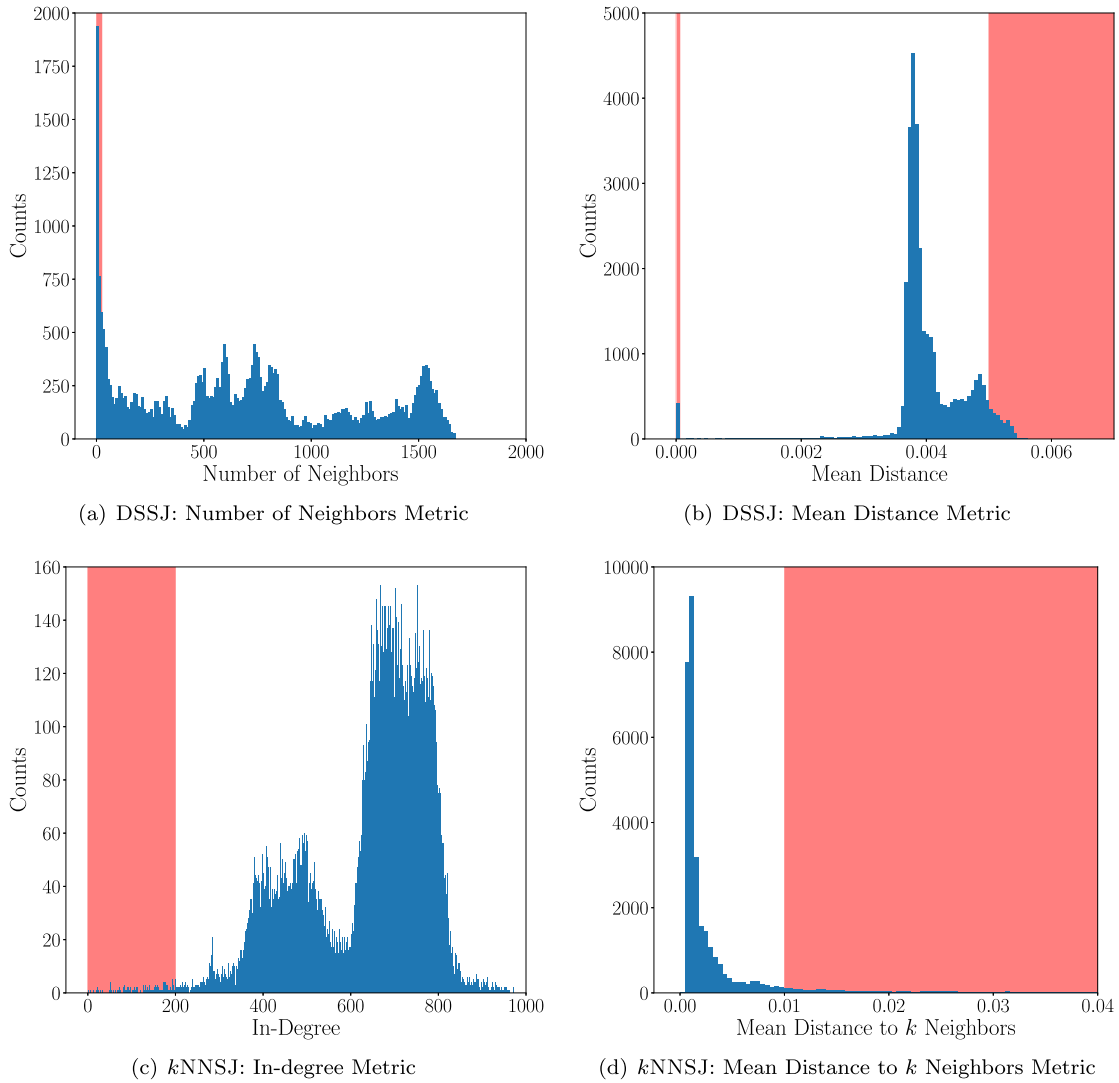
(d) kNNSJ: Mean Distance to k Neighbors Metric

Figure 11. Comparison of the four outlier detection methods on an example $d = 2$ feature space, showing the rotation period as a function of $g - r$ color. (a)–(b) show the two metrics for DSSJ and (c)–(d) show the two metrics for kNNSJ. The points that fall within the top 1% of the ranked list of outliers are shown as solid red points in each plot. The inliers (99%) are shown as the translucent blue points.

considered outliers. Figure 12(a) shows this more clearly, where there is a large number of points (>1% of the total fraction of points) that do not have any neighbors, but not all of these points are classified as outliers. An example point of this type can be clearly observed at $(x, y) \approx (0, 0)$ in Figure 11(a).

2. Figure 11(b) (DSSJ: mean distance metric) is very similar to that of Figure 11(a), except that some points tend to swap outlier/inlier classifications.
3. Figure 11(c) (kNNSJ: in-degree metric) clearly selects outliers that are clustered together as shown at a rotation period $\lesssim$ 500 hr. This is expected as points located in a similar region of a feature space are likely to have a similar in-degree number (recall that the in-degree, or reverse nearest neighbors, refers to the number of instances that a point is considered a neighbor of another point in its set of kNN). Counterintuitively, the outliers selected by this metric are clustered at a rotation period $\lesssim$ 500 hr, and thus appear to be inliers. However, the reason they are not inliers is because the inlying points in this region have few of these outlying points in their set of kNN. Interestingly, this metric finds outliers that are largely different than both of the DSSJ methods in Figures 11(a)–(b).
4. Figure 11(d) (kNNSJ: mean distance to k neighbors metric) is interesting to compare with the two DSSJ methods in Figures 11(a)–(b). With this method, the outlying points at a rotation period $\gtrsim$2200 and $\lesssim$4500 hr are all outliers, and are not interspersed with inliers as shown in Figures 11(a)–(b). The reason the outliers are all selected in this region is because the kNN algorithm requires each point to find k neighbors, which decreases the stochastic variance in this set of k points. Therefore, the outliers will be selected in regions where there are similarly low densities. In contrast the DSSJ methods do not require finding a minimum number of neighbors and so the set of neighbors found within ϵ of each point will have greater variation compared to each other, and therefore DSSJ is more susceptible to stochastic variation in local density.

In summary, each of the outlier detection methods selects outliers based on different criteria. Thus, it may be useful for



(a) DSSJ: Number of Neighbors Metric

(b) DSSJ: Mean Distance Metric

(c) kNNSJ: In-degree Metric

(d) kNNSJ: Mean Distance to k Neighbors Metric

Figure 12. Histograms of the distribution of metric values for the corresponding plots in Figure 11. The red shaded regions are illustrative examples of where outliers are found in the respective distributions. The mean distance metric for DSSJ (b) yields two regions where outliers are found. Points with no neighbors within their search radius will yield a mean distance of zero and so all points that do not have any neighbors are assigned a rank of 0. And those points that have large mean distances to their neighbors will also be outliers.

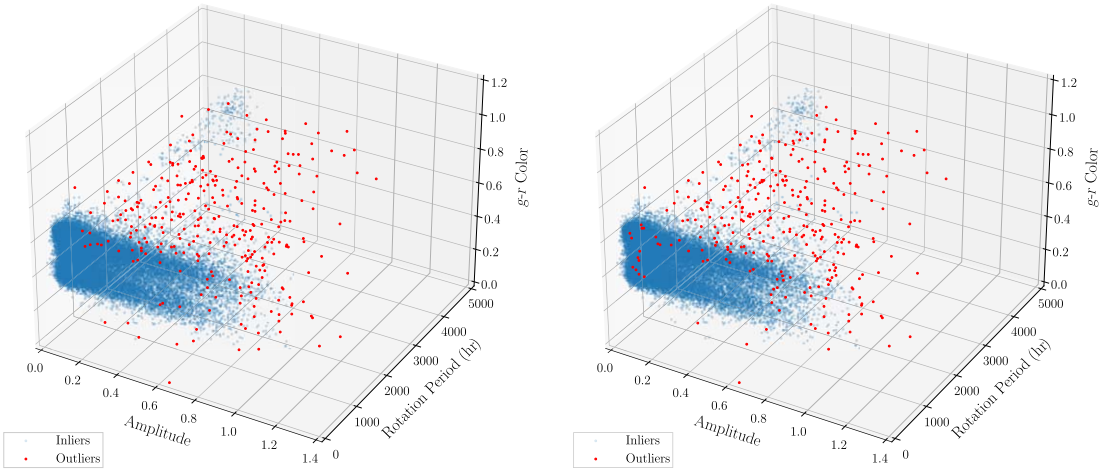
the user to select the outlier method that best suits their science case; for instance, from Figure 11 a user may want to find asteroids that are fast rotators (short rotation periods) that have an unusual color. In this case, the kNN algorithm with the in-degree metric would be best. Another option is to limit outliers to those that have low rankings in multiple outlier detection methods, which can be computed by averaging the outlier scores across methods.

Figure 13 shows a scatterplot of outliers detected in a $d = 3$ feature space (light-curve amplitude, rotation period, and $g - r$ color). Many observations are similar to that of Figure 11 as described above. To better understand the assigned outlier labels in Figure 13, for the interested reader, the Appendix presents a comparison of rankings for this feature space. It is informative for understanding the nuances of the four outlier detection metrics, and illustrates why using an ensemble of methods/metrics are needed to identify outliers. With a single method, we may inadvertently omit detecting interesting solar system objects, and our approach mitigates this potential drawback.

4.5. Viability of Population Outlier Detection at LSST Scale

LSST will observe roughly 5 million asteroids over the 10 yr survey; therefore, the response time and scalability measurements reported in the prior sections will not directly reflect the expected performance of the system when it is deployed on the LSST data stream. In this section, we report the performance of SNAPS population outlier detection on the `LSST5M` data set, which is representative of the LSST data set at the end of the 10 yr survey (Section 2.2).

In this experiment, for DSSJ we set $s_{\min} = 32$ and $s_{\max} = 256$ and for kNNSJ we set $k_{\min} = 2$ and $k_{\max} = 256$. These maximum values are significantly lower than those used for `SNAPShot1`; recall that in Section 4.3, we selected large values of $s_{\max}$ and $k_{\max}$ using `SNAPShot1` to report the conservative (worst-case) performance. `SNAPShot1` has significant variations in the data densities of each feature space because it only contains 31,693 objects and therefore, the feature spaces are poorly sampled. The 5 million objects detected by LSST will reduce the degree of sparsity in the feature spaces, which will reduce the variance in the feature space data densities. This



(a) DSSJ: Number of Neighbors Metric

(b) DSSJ: Mean Distance Metric

(c) kNNSJ: In-Degree Metric

(d) kNNSJ: Mean Distance to k Neighbors Metric

Figure 13. The same as Figure 11, but for a $d = 3$ feature space (light-curve amplitude, rotation period, and $g - r$ color).

implies that smaller $s_{\max}$ values for DSSJ and $k_{\max}$ neighbors for kNNSJ will be needed to adequately sample the feature spaces when operating SNAPS at LSST scale. Furthermore, as observed in Figures 5–8, ϵ^{select} and k^{select} are generally closer to smaller values of ϵ and k rather than larger values, so using smaller ϵ^{select} and k^{select} parameter values at LSST scale should still yield good sets of outlier rankings.

Due to the excessive execution times required of this experiment, we only report the time to compute the first 5% of the 32,752 feature spaces and then extrapolate this time to the expected response time when computing all of the feature spaces. These results are reported in Table 4. We achieve a multi-GPU speedup (using four GPUs over one GPU) of 3.68× and 3.95× on DSSJ and kNNSJ, respectively. Consequently, we achieve a very high ⩾92% parallel efficiency, where the parallel efficiency is the speedup divided by the number of GPUs. Recall that on `SNAPShot1` (Section 4.3), the multi-GPU performance was poor because the ratio of GPU to CPU (host-side) work was low. However, with 5 million data points, there is significantly more work to compute on the GPU and so the abovementioned ratio is much higher, leading to greater multi-GPU scalability. Extrapolating the results in Table 4 to all 32,752 feature spaces, we can compute population outliers for both DSSJ and kNNSJ at LSST scale in roughly 12 days using four GPUs. This will allow SNAPS to provide users updated population outlier detection rankings once per month over the 10 yr survey.

Table 4
The Response Time and Multi-GPU Speedup for Distance Similarity Self-join and k-nearest Neighbors Self-join on `LSST5M`

Algorithm	Response Time (hr) 1 × GPU	Response Time (hr) 4 × GPUs	GPU Speedup
DSSJ	10.83	2.94	3.68
kNNSJ	45.58	11.55	3.95
Total Time	56.41	14.49	

Note. The same permutations of the `SNAPShot1` $d = 15$ feature space are employed. The response times refer to computing 5% of the 32,752 feature spaces for both one and four GPUs. The speedup of using four GPUs over one GPU is reported. The oversubscription factor for DSSJ and kNNSJ is $o = 5$.

We highlight several caveats to this analysis. This level of performance is an estimate, as we only computed 5% of the feature spaces, estimated the selectivity and *k*NN parameter values, estimated the total number of asteroids detected by LSST at the end of the survey, and used a synthetic ZTF-like data set, which may not be representative of the LSST catalog. Furthermore, we anticipate that the GPUs SNAPS will use to compute population outlier scores will be upgraded over the next ∼10 yr, which will decrease the computation time needed for this task. Despite these estimates, we believe that SNAPS population outlier detection will scale to LSST data volumes even if some of these assumptions change during the 10 yr survey.

5. Example Avenues of Scientific Investigation

One of the science goals of SNAPS is to map "edge cases" within the asteroid population, which reveal unusual physical properties that in turn suggest constraints on the origin and evolution of our solar system. In this section, we suggest various scientific investigations that can be carried out using population outlier detection and multidimensional feature space exploration.

5.1. Outliers in a Single Dimension

The simplest outlier detection scheme is one that examines just a single dimension. (1) There are five objects in `SNAPShot1` with $H < 5$, which corresponds to diameters larger than around 300 km. These rare objects—some 0.02% of `SNAPShot1`—turn out to be dwarf planets and large trans-Neptunian objects in the outer solar system, and not main belt asteroids. These objects have different compositions and histories and therefore require a different analysis than the rest of the sample. (2) Some objects in `SNAPShot1` have a derived rotation period of exactly 5000 hr, which is the maximum period searched. These solutions are likely incorrect, and probably represent objects with very low light-curve amplitudes where no true periodic signal can be identified. Leaving these likely false solutions aside, there are 550 objects (1.7% of SNAPShot1) with periods > 1000 hr and <5000 hr. This population was first identified in Erasmus et al. (2021), who identified the Yarkovsky–O'Keefe–Radzievskii–Paddack (YORP) effect, which alters the rotation state of an asymmetric object due to thermal torques (Rubincam 2000; Bottke et al. 2006), as the most likely origin mechanism for these very long periods. This serves as an existence proof for YORP creating slow rotation rates, and puts strong constraints on the initial distribution of rotation rates in the asteroid belt, and the collision rate in the main belt. These constraints in turn constrain important details about the formation and evolution of the solar system.

The amplitude of a light curve indicates the shape of the asteroid, through $\Delta\mathrm{mag} = 2.5 \log \frac{a}{b}$, where a and b are the major and minor axis lengths, respectively, of the ellipsoidal asteroid shape. There are 160 objects in `SNAPShot1` that have light-curve amplitudes > 1 mag, which imply very elongated asteroids where the long side is at least 2.5× longer than the small side. Elongated asteroids are interesting because they may require internal strength (McNeill et al. 2018). The existence of these very elongated objects—present at the level of 0.5% in `SNAPShot1`—demands that asteroid evolution models explain the creation and persistence of such objects.

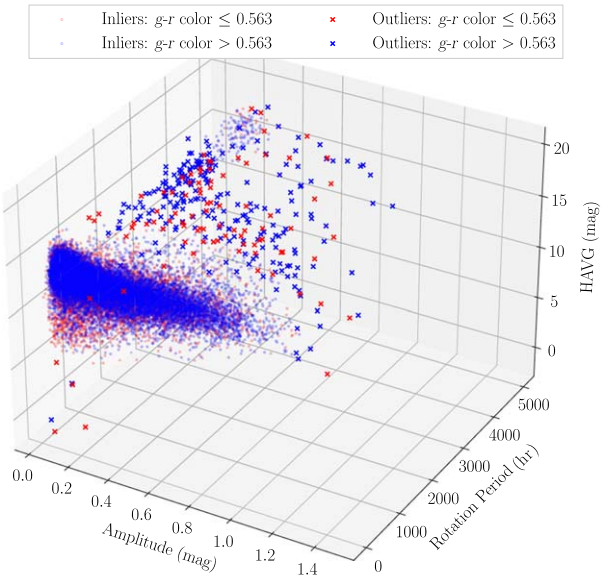


Figure 14. Outliers detected using the *k*NNSJ mean distance to k neighbors metric rankings in the light-curve amplitude, rotation period, and HAVG feature space. The inliers (99% of the objects) are plotted as translucent circles and the outliers with "x" markers (1% of the objects). Red and blue markers denote C-type ($g-r$ color $\leqslant 0.563$) and S-type ($g-r$ color > 0.563) asteroids, respectively. Two-dimensional slices through this space are shown in Figure 15.

5.2. Outliers in Multiple Dimensions

Figure 11 shows outlier sets for our four methods, in two dimensions. Most of the outlying points clearly have either the most extreme rotation periods or colors, which could be identified through a one-dimensional search. However, there are some objects that are neither extreme in period nor color—for example, objects near {500, 0.38}—that are identified in all four methods as outliers. Neither this period nor color is extreme, but this combination of properties places these objects in the outermost envelope of the distribution of objects in period–color space. Thus, outliers can help define details of the multidimensional distributions. The existence of objects near {500, 0.38} may indicate something about the internal structure and strength of those objects. Figure 13 adds amplitude to create a three-dimensional space, again helping to define the envelope of the distributions of these properties.

Figures 14 and 15 show outliers in the parameter space of light-curve amplitude, rotation period, and HAVG (a proxy for size). We then separate S- and C-type asteroids using $g-r = 0.563$ as the boundary between the two taxonomic types (the boundary can be observed in Figure 4 in Trilling et al. 2023). The outliers are roughly equally divided between C- and S-type asteroids, suggesting that, to the degree that composition and internal properties correspond to asteroid color, unusual properties in this multidimensional space are not biased in favor of a particular asteroid composition. These figures show a group of objects that are particularly unusual, with very long periods (2000–4000 hr), relatively small amplitudes (0.2 mag), and unremarkable H magnitudes (around 15, so a few kilometers in diameter). This three-dimensional outlier detection scheme reveals this population of very slowly rotating asteroids that are nearly round (low amplitude) and have sizes around 3 km, more or less. Asteroid dynamical models must include a mechanism for the existence of these bodies.

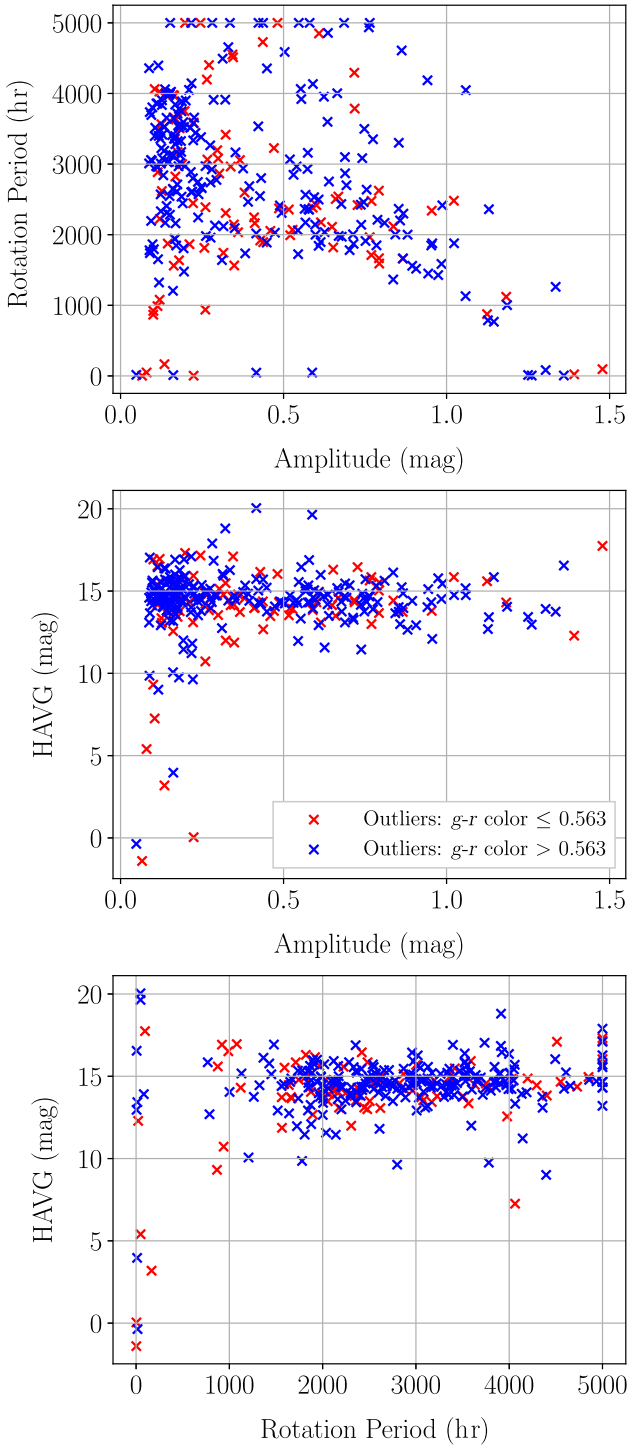


Figure 15. The same as Figure 14, except that only the outliers are shown. The panels ordered from top to bottom are as follows: rotation period as a function of amplitude, HAVG as a function of amplitude, and HAVG as a function of rotation period. Outliers in this space are equally divided between C-type and S-type asteroids, indicating that composition is not a significant driver of unusual behaviors in these three dimensions.

Figure 16 shows this same parameter space (amplitude, rotation period, and HAVG), though with a different color coding to emphasize a particular subset of outliers: objects with amplitudes $\geqslant 1$ mag and periods $\geqslant 500$ hr. These are objects that are quite elongated and slowly rotating, in both cases showing unusual properties. In general, the maximum amplitude that a single elongated body can have is around 0.9 mag (Strauss et al. 2023). Amplitudes greater than this probably indicate binary or contact binary systems. Our population outlier detection approaches may be detecting binary asteroids in the main belt—and, intriguingly, slowly rotating binaries—which implies a relatively small amount of angular momentum. The origin and evolution of these bodies is an interesting question and may provide further clues to the overall evolutionary history of the asteroid belt. Furthermore, within this group, the ratio of C:S asteroids is 12:5, which may not be statistically significant but is at least suggestive that the mechanism of creating such unusual asteroids may depend on composition. The sizes are 1–10 km (H is estimated to be 12.5–17.5), potentially a key clue to the evolution of these bodies.

There are many other examples of multidimensional investigations that can probe the properties of asteroids. As an example of another extended science case, our team is identifying outliers in minimum required strength in a space that is defined by a number of asteroid physical properties (M Chernyavskaya et al. 2024, in preparation). In summary, the various techniques presented here to characterize large data sets are powerful tools that provide evidence that would not otherwise be available and allow us to carry out experiments about the formation and evolution of the solar system.

6. Discussion and Conclusions

This paper has presented the population outlier detection functionality of the SNAPS alert broker. We summarize our contributions as follows.

1. We have constructed an asteroid outlier detection system that is demonstrated on the first SNAPS data release, `SNAPShot1`, which employs state-of-the-art GPU algorithms. We showed that the system provides outlier detection capabilities in addition to feature space exploration and data prioritization, which will help researchers rapidly identify objects of interest that may warrant further investigation.
2. We utilize a feature space permutation approach that allows users to select objects of interest based on ZTF measurements, adjacent catalogs, and derived properties. We show that this approach can assist in data exploration, prioritization, and visualization activities to accelerate the scientific discovery process.
3. We demonstrate that the outlier detection system is currently able to scale to the ZTF data rate. Through measurements on a synthetic data set, we expect that the system will be able to derive population outliers at LSST scale and will be able to provide outlier detection rankings roughly once per month. The monthly data will be archived and made publicly available on our website to enable tracking how objects evolve over time relative to the population of objects.
4. We highlight several preliminary scientific results, and example avenues of scientific investigation that will be enabled by SNAPS population outlier detection capabilities.

We demonstrate the grasp of these approaches with a few example science cases, of the countless investigations that are possible. As described above, our calculated outlier lists will be published on our SNAPS web page and updated monthly,

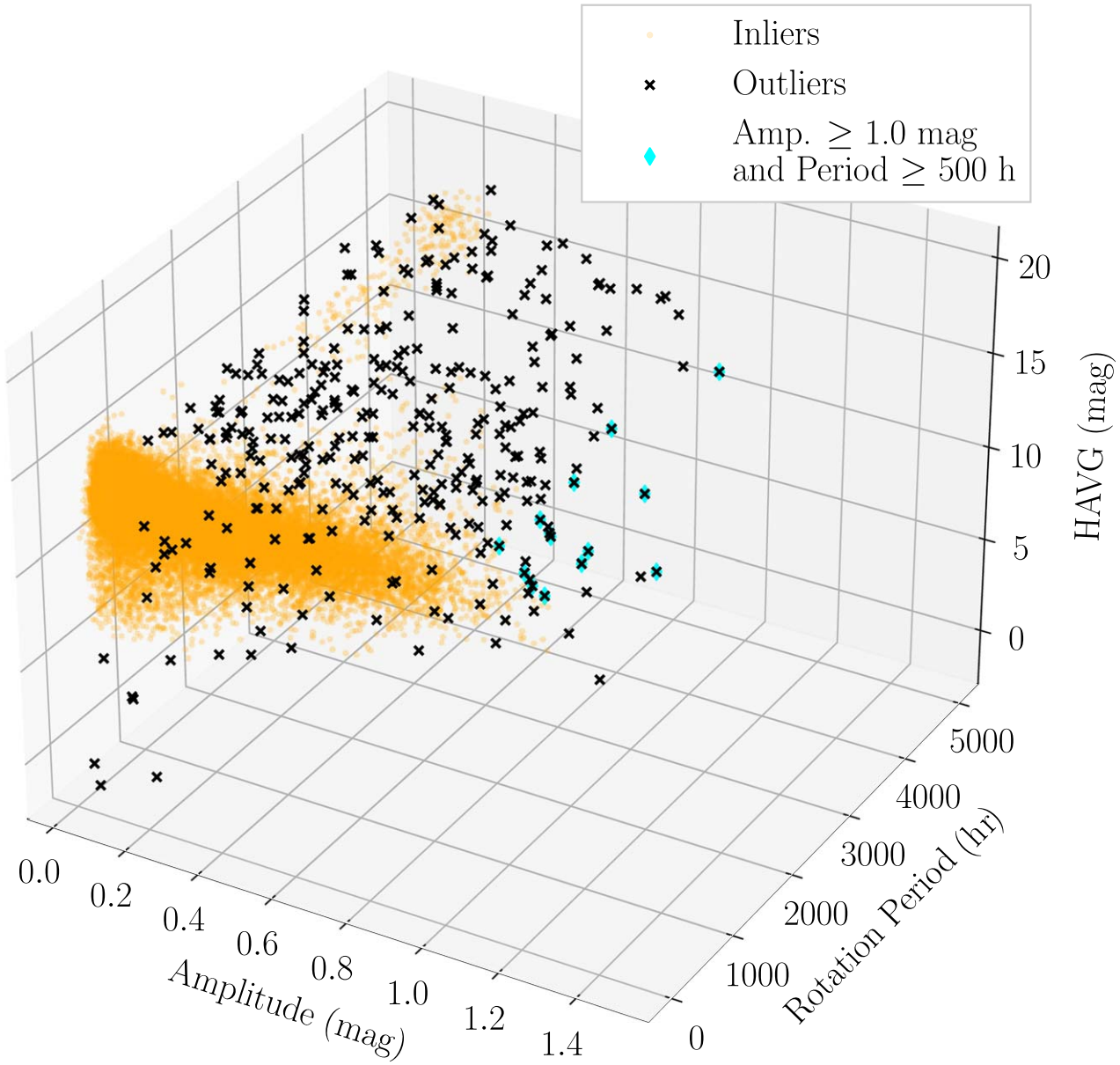


Figure 16. Outliers detected using the DSSJ number of neighbors metric in the light-curve amplitude, rotation period, and HAVG feature space. The inliers (99% of the objects) are plotted as translucent circles and the outliers are plotted with "x" markers (1% of the objects). We highlight with cyan diamond markers objects with a light-curve amplitude $\geqslant$ 1 mag and rotation period of $\geqslant$500 hr. There are only 14 objects with these properties and our method is able to detect all of these as outliers denoted by the 14 diamond markers imposed behind the "x" markers.

allowing all users to pursue their own experiments with our large and growing catalog.

It is clear that deriving outlier rankings across all feature spaces is computationally expensive. There are several methods that could be employed to reduce this computational cost. For example, while we derived outliers for 15 features in `SNAPShot1`, we are aware that some of these features are redundant or correlated, and so we could reduce the total number of feature spaces that we provide to users by eliminating some of these feature spaces. However, the SNAPS team will aim to provide outlier rankings for at least all permutations of $d = 15$ features; however, these features will change slightly when we begin receiving data from the Rubin Observatory. At present, we believe that providing $d = 15$ features should provide a reasonable level of outlier detection and data space exploration for the community.

We showed that population outlier detection with SNAPS is expected to scale to LSST data volumes assuming that the current functionality is sufficient for the LSST era. We predict that the community will desire additional functionality, such as providing binary classification labels (inlier versus outlier) instead of only using ranked lists of objects. The SNAPS team intends to add this functionality to the system in the future.

A distinct but complimentary service that SNAPS could provide to the community is clustering each of the feature spaces. This task is similar to the outlier detection methods described here, as clustering typically relies on similarity searches (finding nearby points in a feature space). This would allow for the automated detection of groups of objects that may be of interest. A canonical example would be to derive asteroid families as a function of orbital elements (Parker et al. 2008)

while also examining additional features/dimensions that may be less obvious to explore.

We identified that one bottleneck in the system is using Python to invoke the GPU algorithms for each feature space. Since we process multiple feature spaces concurrently, this yields nonnegligible memory pressure, which impacts the rate at which we can process a given feature space on a GPU. Thus, future work includes running a single process that executes all of the feature spaces on the GPU for DSSJ and *k*NNSJ. By undertaking this research direction, we anticipate that there will be interesting opportunities for data reuse, such as providing the set of *k*NN from *k*NNSJ to DSSJ for its computation, thus potentially eliminating a significant fraction of the total DSSJ cost. Another direction that can be undertaken to improve performance is to leverage the tensor cores on many modern GPUs to perform Euclidean distance calculations (Gallet & Gowanlock 2022). These and other algorithmic advances are critical for reaching LSST scale while expanding the services offered by SNAPS, as many of the computational bottlenecks identified in this paper cannot be solved by hardware advances alone over the next $\gtrsim$10 yr.

At present, other alert broker teams are preparing for LSST by developing software infrastructure to classify objects. We welcome collaboration with other teams that anticipate requiring the population outlier detection methods described here for their respective science domains.

Acknowledgments

We thank the anonymous referee for a very thorough review of the manuscript which significantly strengthened the paper. M.G. is supported by the National Science Foundation under grant No. 2042155. D.E.T., M.G., and M.C. are supported by the National Science Foundation under Grant No. 2206796. This work is supported by the Arizona Board of Regents and the Technology Research Initiative Research Fund (TRIF) Small Research Equipment Acquisition Program (SREAP).

ZTF is a public–private partnership, with equal support from the ZTF Partnership and from the U.S. National Science Foundation through the Mid-Scale Innovations Program (MSIP). The ZTF partnership is a consortium of the following universities and institutions (listed in descending longitude): TANGO Consortium of Taiwan; Weizmann Institute of Sciences, Israel; Oskar Klein Center, Stockholm University, Sweden; Deutsches Elektronen-Synchrotron & Humboldt University, Germany; Ruhr University, Germany; Institut national de physique nucléaire et de physique des particules, France; University of Warwick, UK; Trinity College, Dublin, Ireland; University of Maryland, College Park, USA; Northwestern University, Evanston, USA; University of Wisconsin, Milwaukee, USA; Lawrence Livermore National Laboratory, USA; IPAC, Caltech, USA; and Caltech, USA.

Software: NumPy (Harris et al. 2020) and pandas (pandas development team 2020).

Appendix
Comparison of Rankings between Outlier Detection Methods

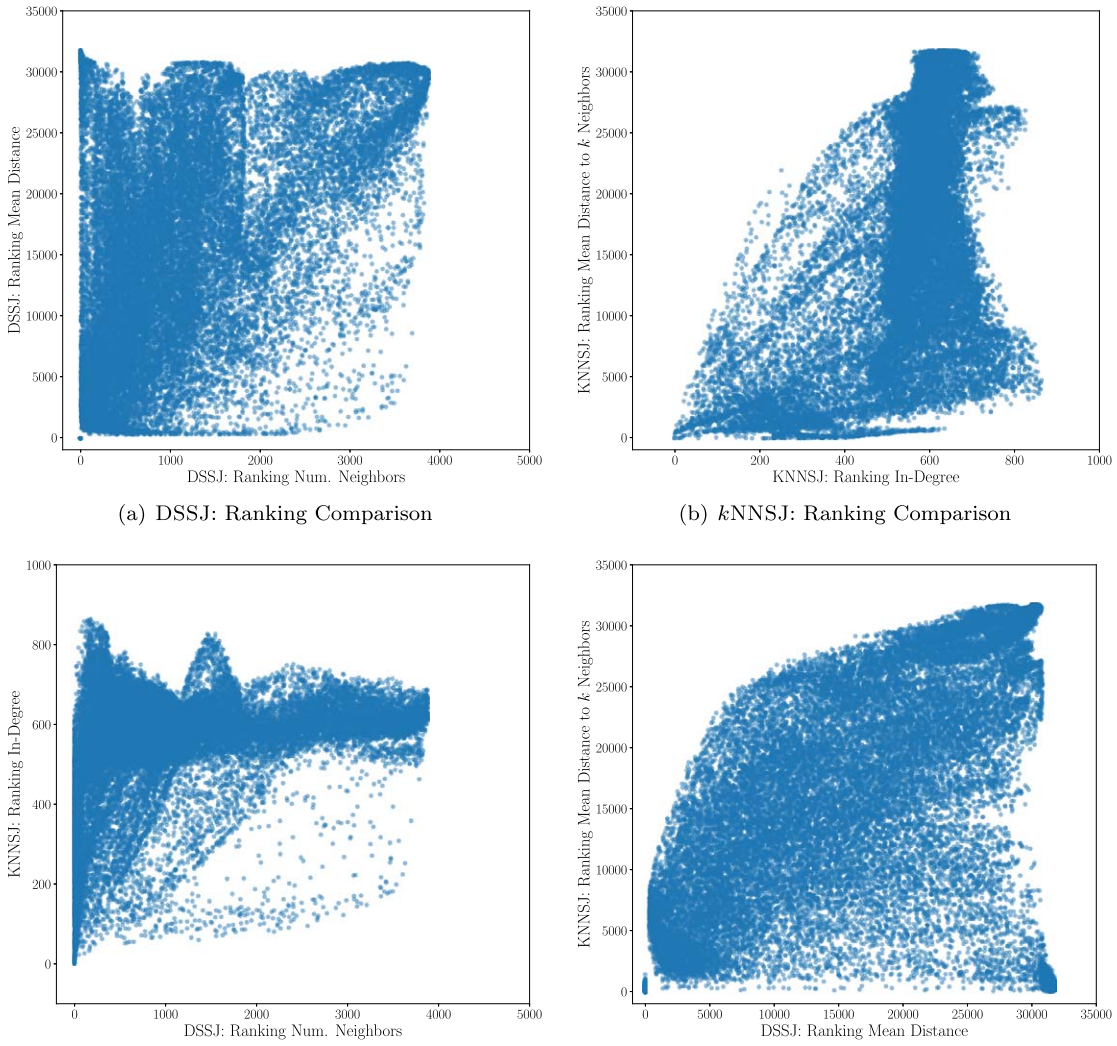
From Figure 13 we observed that there may be minimal overlap between those points that are selected as outliers between the outlier detection metrics. Figure 17 shows another perspective on the comparison of rankings that were shown in Figure 13, where we plot the rankings for each object as a scatterplot. Figure 17 shows the following subplots: (a) shows a ranking comparison for the two DSSJ metrics, (b) shows a comparison for the two *k*NNSJ metrics, (c) shows a comparison of the two *distance-oblivious* metrics, and (d) shows a comparison between the two *distance-aware* metrics. Recall that in all figures, the outliers are those with a low rank value.

Figure 17(a) shows a comparison of DSSJ metrics where we observe that many of the points that have few neighbors (with a rank $\gtrsim$ 0) have a large range of possible ranks using the mean distance metric. This shows that a large mean distance to neighboring points may be achieved when there are few or many points within the search radius ϵ of a given point.

Figure 17(b) shows a comparison of *k*NNSJ metrics. Here we find that there are consistencies between outlier scores for both metrics when the rankings are $\gtrsim$0. Interestingly, there is an overabundance of in-degree rankings with values between $\sim$500−700, although these points will not be considered outliers in this feature space.

Figure 17(c) compares the rankings derived by the *distance-oblivious* metrics for DSSJ and *k*NNSJ. We find that there is little correlation between the rankings of these two methods. This is because it is possible to have a low in-degree value in the *k*NN graph, but yet have several neighbors found within the search radius ϵ. Despite this, there are still numerous points that have ranks near 0 in both metrics, which would indicate that they are outliers using both metrics.

Figure 17(d) compares the rankings derived by the *distance-aware* metrics for DSSJ and *k*NNSJ. We clearly observe that there is a deficit of points assigned a low mean distance rank and a high ($\gtrsim$10,000) mean distance to k neighbors rank. Compared to the other plots, the rankings appear to be the most correlated, although there are still major differences in the assigned rankings.



(a) DSSJ: Ranking Comparison

(b) kNNSJ: Ranking Comparison

(c) DSSJ vs. kNNSJ: Distance-Oblivious Comparison

(d) DSSJ vs. kNNSJ: Distance-Aware Comparison

Figure 17. Scatterplots showing a comparison of rankings for the same $d = 3$ feature space shown in Figure 13. (a) Ranking comparison for the two DSSJ metrics. (b) Ranking comparison for the two kNNSJ metrics. (c) Comparison of the two *distance-oblivious* metrics. (d) Comparison of the two *distance-aware* metrics. Note that across all figures, the *distance-oblivious* metrics (DSSJ: number of neighbors and kNNSJ: in-degree) have numerous points that share the same rank, and so they have a smaller range of possible rank values compared to the *distance-aware* metrics (DSSJ: mean distance and kNNSJ: mean distance to k neighbors).

ORCID iDs

Michael Gowanlock https://orcid.org/0000-0002-0826-6204
David E. Trilling https://orcid.org/0000-0003-4580-3790
Daniel Kramer https://orcid.org/0000-0002-6676-1713
Maria Chernyavskaya https://orcid.org/0000-0002-6292-9056
Andrew McNeill https://orcid.org/0009-0005-9955-1500

References

Bellm, E. C., Kulkarni, S. R., Graham, M. J., et al. 2018, PASP, 131, 018002
Bottke, W. F. J., Vokrouhlický, D., Rubincam, D. P., & Nesvorný, D. 2006, AREPS, 34, 157
Bowell, E., Hapke, B., Domingue, D., et al. 1989, in Asteroids II, (Tucson, AZ: Univ. Arizona Press), 524
Campos, G. O., Zimek, A., Sander, J., et al. 2016, Data Mining and Knowledge Discovery, 30, 891
Capodieci, N., Cavicchioli, R., Valente, P., & Bertogna, M. 2017, in Proc. 25th Int. Conf. Real-Time Networks and Systems (New York: ACM), 48
Coughlin, M. W., Burdge, K., Duev, D. A., et al. 2021, MNRAS, 505, 2954
Drake, A. J., Graham, M. J., Djorgovski, S. G., et al. 2014, ApJS, 213, 9
Erasmus, N., Kramer, D., McNeill, A., et al. 2021, MNRAS, 506, 3872
Förster, F., Cabrera-Vives, G., Castillo-Navarrete, E., et al. 2021, AJ, 161, 242
Gallet, B., & Gowanlock, M. 2021, DSE, 6, 39
Gallet, B., & Gowanlock, M. 2022, in 2022 IEEE 29th Int. Conf. High Performance Computing, Data, and Analytics (HiPC) (Piscataway, NJ: IEEE), 135
Gowanlock, M. 2021, JPDC, 149, 119
Gowanlock, M., Gallet, B., & Donnelly, B. 2023, in Computational Science—ICCS 2023, ed. J. Mikyška et al. (Cham: Springer), 357
Gowanlock, M., & Karsin, B. 2019, in Proc. 15th Int. Workshop on Data Management on New Hardware, DaMoN'19 (New York: ACM)
Harris, C. R., Millman, K. J., van der Walt, S. J., et al. 2020, Natur, 585, 357
Hautamaki, V., Karkkainen, I., & Franti, P. 2004, in Proc. 17th Int. Conf. Pattern Recognition, 2004. ICPR 2004., Vol. 3 (Piscataway, NJ: IEEE), 430
Ivezić, Ž., Kahn, S. M., Tyson, J. A., et al. 2019, ApJ, 873, 111
Knorr, E. M., & Ng, R. T. 1997, KDD, 97, 219
Le Montagner, R., Peloton, J., Carry, B., et al. 2023, A&A, 680, A17
Lu, Y. L., Curtis, J. L., Angus, R., David, T. J., & Hattori, S. 2022, AJ, 164, 251
Mainzer, A., Bauer, J., Cutri, R., et al. 2019, Technical Report, NASA
Matheson, T., Stubens, C., Wolf, N., et al. 2021, AJ, 161, 107
McNeill, A., Fitzsimmons, A., Jedicke, R., et al. 2018, AJ, 156, 282
Möller, A., Peloton, J., Ishida, E. E. O., et al. 2020, MNRAS, 501, 3272
pandas development team, T. 2020, pandas-dev/pandas: Pandas, Zenodo, doi:10.5281/zenodo.3509134
Pankratius, V., Li, J., Gowanlock, M., et al. 2016, IISys, 31, 3
Parker, A., Ivezić, Ž., Jurić, M., et al. 2008, Icar, 198, 138
Rubincam, D. P. 2000, Icar, 148, 2
Sánchez-Sáez, P., Reyes, I., Valenzuela, C., et al. 2021, AJ, 161, 141

Shappee, B. J., Prieto, J. L., Grupe, D., et al. 2014, ApJ, 788, 48
Smith, K. W., Williams, R. D., Young, D. R., et al. 2019, RNAAS, 3, 26
Soraisam, M. D., Saha, A., Matheson, T., et al. 2020, ApJ, 892, 112
Strauss, R., Trilling, D. E., Bernardinelli, P. H., et al. 2023, arXiv:2309.04034
Tonry, J. L., Denneau, L., Heinze, A. N., et al. 2018, PASP, 130, 064505
Trilling, D. E., Gowanlock, M., Kramer, D., et al. 2023, AJ, 165, 111
van Roestel, J., Duev, D. A., Mahabal, A. A., et al. 2021, AJ, 161, 267
Wagstaff, K. L., Lanza, N. L., Thompson, D. R., Dietterich, T. G., & Gilmore, M. S. 2013, in Twenty-Seventh AAAI Conf. Artificial Intelligence (Washington, DC: AAAI Press)
Zhou, L., Pan, S., Wang, J., & Vasilakos, A. V. 2017, Neurocomputing, 237, 350
Zimek, A., Schubert, E., & Kriegel, H.-P. 2012, Stat. Anal. Data Mining, 5, 363