FARMYARD: A Generic GPU-based Pipeline for Feature Discovery from Massive Planetary LiDAR Data

Shen Liang, Antoine Lucas, Stephen Porder, Gregory Sainton, Alexandre Fournier, Themis Palpanas

***Abstract*—In recent decades, with the placement of LiDAR remote sensing instruments in orbit, we now have global coverage of the bare-ground elevation on the Earth, Mars and beyond. Encoded in such planetary LiDAR data are interesting landscape features that promise to further our knowledge of planetary topography. However, discovery of such features entails 3 major challenges: 1) massive data; 2) the need for local multi-scale features; 3) sensitivity to interfering factors. To address these challenges, we propose FARMYARD, a generic pipeline for Feature Discovery From Planetary LiDAR Data Data. To our knowledge, this is the first time such a pipeline has been proposed, which provides a brand new methodology for comparative studies of planetary topography. Specifically, drawing on the parallel computing power of the Graphics Processing Unit (GPU), we propose a novel pseudo-on-pass sweep (POPS) framework for fast and memory-efficient feature extraction for massive planetary LiDAR data, a two-level division scheme for local regions with support for multi-scale features, and a Domain-Shifted Partition (DSP) scheme for feature evaluation that is robust against interfering factors. To showcase the utility of our FARMYARD pipeline, we deploy it to a real-world research project, which seeks to find topographical signatures of life by discovering features that can potentially distinguish between the Earth and alien worlds with no known life activity. We also highlight the efficiency of our POPS framework with experiments on both synthetic and real data, which can be thousands of times faster than its CPU-based counterpart, including a multi-core parallel solution.**

***Index Terms*—LiDAR, planets, topography, feature discovery, GPU**

I. INTRODUCTION

TOPOGRAPHY, the surface configuration of a planet, bears the imprint of the processes that shape it. Deciphering the "text" encoded in topography has been the goal of scholars for centuries (Darwin, Gilbert, others). In the past 15 years, the advent of orbital Light Detection And Ranging (LiDAR) technology has led to global high-resolution topographical data for Mars [1] and the Moon [2], and to some extent for other bodies such as Titan, the largest moon of Saturn [3]. Moreover, December of 2018 saw the arrival of the Global Ecosystem Dynamics Investigation[1] (GEDI) [4], a full-waveform LiDAR installed onboard the International Space Station (ISS). Over the nominal two-year mission, GEDI provides an estimated 10 billion waveform LiDAR observations between 51 Degrees N and 51 Degrees S, producing readings at both ground and vegetation height. Notably, GEDI is the first instrument to provide complete waveforms of LiDAR echoes from space. Thus, for the first time, we can literally see through the living skin of the Earth, obtaining bare ground readings that are not influenced by vegetation at the global scale.

Up until now, LiDAR has been used to characterize the surface properties, such as elevation, roughness, as well as canopy height or density in the case of the Earth. By nature, such studies offer a useful constraint on hypotheses about surface evolution, namely how lithology, tectonics, and climate on the Earth control the transport flux of rivers, or how the surface properties are affected by resurfacing mechanisms by Mars or the Moon [5], [6]. However, existing works are significantly lacking in *inter-planetary comparative studies*, which may hold the key to answering some of the most intriguing questions concerning planetary topography. For example, in 2006, Dietrich and Perron asked the question *is there a topographical signature of life* [5], and concluded that none could be detected from terrestrial data available back then. To answer this question, we can first compare the topographical features of the Earth and alien worlds (where no life is known to exist) to find characteristics unique to the topography of the Earth, and then study whether there is a relationship between such characteristics with terrestrial life. Such comparative studies are now made possible by the abundance of planetary LiDAR data [1]–[3] that provides global coverage with relatively high resolution. Moreover, with these data, we are now able to conduct apple-to-apple comparisons of planets. For example, to make meaningful comparisons between the Earth and other planets, we need bare-ground observations of the Earth as there is no known vegetation on alien worlds. This was previously impossible until the GEDI mission [3] which, as was mentioned, has enabled us to see through the terrestrial vegetation cover for the first time.

While we now have the high-quality LiDAR data needed to conduct inter-planetary comparative studies, we still lack the proper methodology needed to carry out such comparisons. Indeed, there have been strong geophysical perspectives to justify global topography field surveys since the 1990s on terrestrial planets, yet existing works on comparison of planetary bodies have mainly focused on exploring deep earth

---

S. Liang is at Data Intelligence Institute of Paris (diiP), Université Paris Cité, France

A. Lucas, G. Sainton and A. Fournier are at Université Paris Cité, Institut de physique du globe de Paris, France

S. Porder is at the Brown University, USA

T. Palpanas is at diiP and LIPADE, Université Paris Cité, France

Manuscript received on December 4, 2022

[1] https://gedi.umd.edu/

issues with spherical harmonic [7] and/or spherical spline [8] approaches. Such a methodology leads to very smooth global signals with limited high-frequency components, losing information that is present in the raw records which may be significant to subsequent comparative analysis.

In view of the drawbacks in existing works, in this paper, we present a brand new methodology for comparative studies with planetary LiDAR data. Specifically, the task we will undertake in this paper is defined as follows.

Definition 1: **Feature Discovery From Planetary LiDAR Data (FARMYARD):** given a set of planetary LiDAR data, find interesting features that can be used to distinguish between two or more geo-objects of the same category $\mathcal{T}$.
For example, suppose $\mathcal{T}$ is $planets$, and we have LiDAR data for the Earth and Mars, FARMYARD aims to find features that can be used to distinguish between the two planets. For the rest of the paper, we refer to $\mathcal{T}$ as the **Target Category (TC)**. It is worth noting that while the motivation of this work is to enable effective comparison of different planets, FARMYARD also supports comparison of other geo-objects by changing $\mathcal{T}$ from *planets* to other categories. For instance, one can change $\mathcal{T}$ to $landscapes$ to compare different landscapes, or to $geolocations$ to compare different regions on the same planet.

In our work, we draw on a machine learning based methodology and formulate FARMYARD as a classification problem. Specifically, we first extract certain candidate features from the dataset that can potentially distinguish between the geo-objects in question, and then conduct Cross Validation (CV) using some pre-defined classifiers upon these features, applying evaluation criteria for classification (e.g. Cohen's kappa coefficient [9], F1-score, etc.) to quantify their distinguishing power in terms of $\mathcal{T}$. However, this comes with the following challenges.

- **C1: Massive data.** Planetary LiDAR data often cover vast areas, making them big data in their raw form. For example, as was previously mentioned, the GEDI [4] planetary LiDAR is to conduct 10 billion measurements of terrestrial landscapes over its nominal two-year mission. Worse still, raw planetary LiDAR data is often irregularly-sampled (namely not aligned to a uniform raster), which is likely to pose an inconvenience to further analysis. Hence, it is necessary to transform them into raster data using some interpolation algorithm such as Kriging [10] and Inverse Distance Weighting (IDW) [11]. This can make the volume of the raster data far more massive than the already big raw data. For example, the dataset used in our experiments (see Section V) covers multiple geolocations on the Earth and Mars, where the ratio between the numbers of data points after and before the interpolation ranges from 2.972 for the terrestrial Sycamore drainage system, to 104.130 for the Martian Elaver drainage system. Like many other big data problems, the sheer volume of planetary LiDAR data calls for analytical algorithms with high efficiency.
- **C2: The need for local multi-scale features.** As was previously mentioned, the methodology applied by existing works largely leads to very smooth signals on a global scale [7], [8]. However, important topographical features often reside not in such global, low-resolution signals, but in local, high-resolution data. For example, the topographical differences between the Earth and Mars (or those among various landscapes) may well be exhibited in minute details of their local regions (e.g., specific volcanoes, drainage systems, landslides, impact craters, etc., which are scattered across the globe). Moreover, even for the same local region, its topographical features can exhibit commonalities [12] or individualities on multiple scales. For example, for an impact crater (or a landslide), its bottom may share similar features to the entire crater (or landslide), such as being mainly flat. By contrast, a volcano edifice may exhibit varying features from small scales to larger scales. Hence, delicate design choices are needed to capture such local multi-scale features across landscapes and planets.
- **C3: Sensitivity to interfering factors.** As is well-known to the machine learning community [13], due to the presence of interfering factors, a high classification score does not necessarily translate to high feature quality. For example, suppose we want to find features that can distinguish between the Earth and Mars, and we have a LiDAR dataset composed of two classes: one corresponds to volcanoes on the Earth, while the other corresponds to drainage systems on Mars. In this case, a high classification score does not necessarily mean that the extracted features can effectively distinguish between the Earth and Mars, for they may actually have been distinguishing between volcanoes and drainage systems. In other words, differences in *landscapes* may well interfere with feature discovery concerning the differences in *planets*. Such interference must be minimized.

To address these challenges, we propose the following 3 solutions that will be implemented in our FARMYARD pipeline.

- **S1: Memory-efficient Graphics Processing Unit (GPU) based parallelism.** To address Challenge **C1**, namely the problem with massive data, we draw on the power of GPU-based parallelism [14] in all potential computational bottlenecks in the FARMYARD pipeline, including raster interpolation, feature extraction and K-fold CV. This results in a (nearly) fully GPU-based pipeline. Most notably, we present **a generic GPU-based framework for raw feature extraction** of planetary raster data that is compatible with a wide range of core feature extraction methods. At the heart of this framework is a novel **Pseudo-One-Pass Sweep (POPS)** routine that allows for highly efficient feature extraction with limited memory resources.
- **S2: Two-level division of local Regions-Of-Interest (ROIs).** To address Challenge **C2**, namely the problem with local multi-scale features, we select multiple local regions scattered around the globe on which we will extract features. We then apply a novel two-level division scheme to the ROIs derived from these regions to generate the training/testing examples for CV, which can simultaneously address the need for multi-scale features

and balance the quality and quantity of the CV examples.

- **S3: Domain-Shifted Partition (DSP) of diverse data.** To address Challenge **C3**, namely the problem with interfering factors, we first ensure that the entire dataset is diverse enough for these factors to be taken into account. We then devise a novel Domain-Shifted Partition (DSP) strategy for the dataset which can lead to train-test splits that maximize intra-class diversity in terms of the interfering factors. This can in turn make the classification scores better reflect the classifier's robustness to interference, thus better evaluating feature quality.

Before we move on, we would like to note 2 points: First, we are NOT proposing a specific feature extraction method for planetary LiDAR data. Rather, we are proposing a *generic* pipeline that can be instantiated using existing (and future) feature extraction methods (which we call *core methods*). Second, unless otherwise stated, we limit our discussion to traditional hand-crafted features (namely non-deep learning features) in this work. The reason for this is that while deep learning can often yield higher classification scores, its black-box nature can prevent the user from interpreting the learned features and thus gaining insight into the geoscientific knowledge behind them.

In summary, the main contributions of this paper are as follows.

- We present a generic GPU-based pipeline for Feature Discovery From planetary LiDAR Data (FARMYARD). To the best of our knowledge, this is the first time such a pipeline has been proposed, which presents a brand new methodology for comparative studies of planetary topography that can effectively exploit local high-resolution information. Exploiting the parallel processing power of the GPU, FARMYARD promises raster interpolation, multi-scale decimation and feature evaluation of topographical data with efficiency never reached before.
- As the 1st highlight of the FARMYARD pipeline, we propose a generic framework for GPU-based raw feature extraction, which as far as we know is another first attempt at planetary LiDAR data analysis. Exploiting a novel Pseudo-One-Pass Sweep (POPS) routine, this framework can achieve memory- and time-efficient raw feature discovery with compatibility with a wide range of core feature extraction methods.
- As the 2nd highlight of the FARMYARD pipeline, we propose a two-level ROI division scheme to produce CV examples with balanced quality and quantity while enabling multi-scale feature extraction of local regions.
- As the 3rd highlight of the FARMYARD pipeline, we propose a Domain-Shifted Partition (DSP) strategy for train-test splits. This results in feature evaluation that is robust against interfering factors, which can in turn lead to better assessment of feature quality.
- We conduct extensive experiments to showcase the utility of our FARMYARD pipeline in an ongoing project called PARKER that seeks to find topographical signatures of terrestrial life activities, and provide suggestions to future studies on planetary LiDAR based on our results on real-world data. We also examine the efficiency of our POPS framework, which can be hundreds or even thousands of times faster than its CPU-based counterpart, including a CPU-based multi-core parallel solution with Message Pass Interface (MPI) [15].

The rest of this paper is organized as follows: Section II provides an overview of planetary LiDAR data. Section III presents our FARMYARD pipeline. Section IV details how we leverage the GPU to accelerate this pipeline. Section V demonstrates the experimental results. Section VI concludes the paper.

II. PLANETARY LIDAR DATA

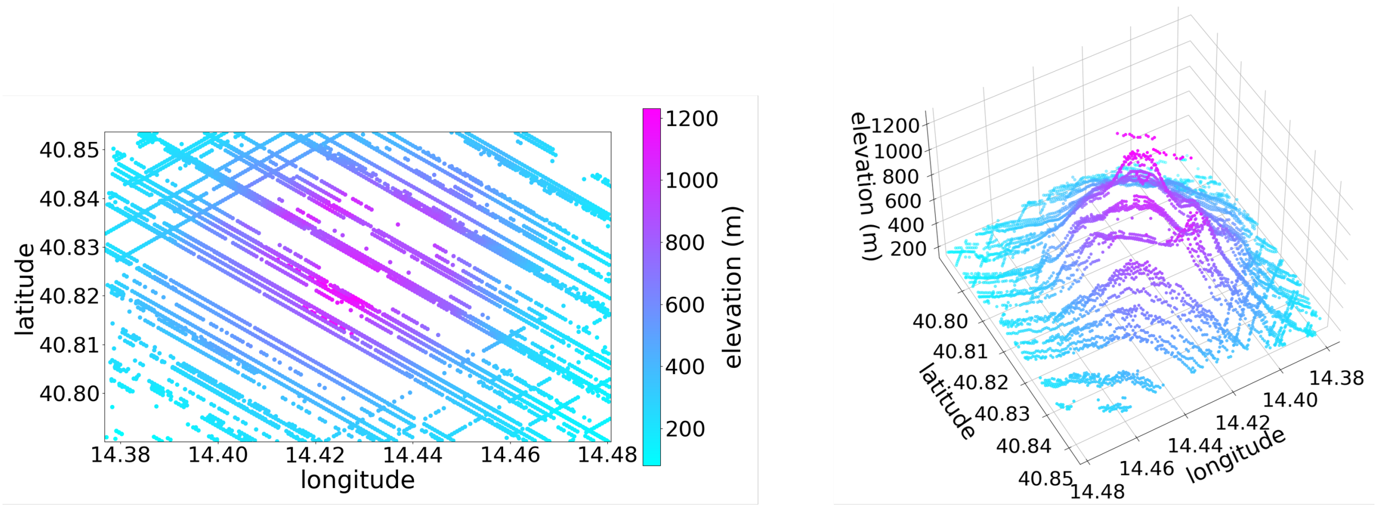
Light detection and ranging (LiDAR) laser systems have been put into orbit around Mars, the Moon and more recently around the Earth. They share the same physical concept and formalism. Specifically, planetary LiDAR comes in the form of represented as 3D point clouds, in which each point is represented by a triplet (x, y, z) where x is the longitude, y is the latitude, and z is the elevation of that point. Fig. 1 shows a LiDAR point cloud of the Vesuvius volcano in Italy.

Next, we present brief introductions of some of the existing planetary LiDAR data repositories.

1) GEDI [4]: Global Ecosystem Dynamics Investigation (GEDI) is a full-waveform topographic LiDAR installed onboard the International Space Station (ISS). The GEDI instrument uses three lasers that produce 8 ground tracks. Each ground track is a series of footprint samples separated by 60 m in the along-track direction. The distance between adjacent ground tracks is 600m. The sole GEDI observable is the returned laser waveform, which records the three-dimensional distribution of intercepted surfaces within the extent of the illuminated laser footprint (23 ± 2m). Measurements are acquired over the land surface during day and night conditions (one ISS orbit is about 90 minutes) within the observational limit of the ISS (51.6 Degrees N to S latitude). This orbit results in greater measurement density at high latitudes and fewer measurements near the equator. The total number of land-surface measurements anticipated over the nominal two-year mission life is 10 billion. A significant advantage of GEDI is that apart from producing observations at vegetation height, it can also see through the vegetation and produce observations of the ground surface, thus avoiding biases induced by top-of-canopy inferences of underlying topography [16], [17].

2) MOLA [1]: Mars was scanned globally by the Mars Orbiter Laser Altimeter (MOLA) onboard MGS from 1996 to 2001. This global data set offers half a billion echoes from the surface with a ground spatial sampling of about 460m at the equator with a beam diameter of about 75m in diameter. Note that subsequent missions offer higher resolution from stereo imagery, e.g., HRSC onboard Mars-Express, HiRISE onboard Mars Reconnaissance Orbiter, and CaSISS onboard Trace Gas Orbiter. Therefore, the topography can be derived at 1m sampling very locally in completion with the global MOLA LiDAR echoes.

3) LOLA [2]: Similarly to Mars, the Moon is being scanned since 2009 with the Lunar Orbiter Laser Altimeter (LOLA),



(a) 2D

(b) 3D

Fig. 1. Planetary LiDAR point cloud in 2D (a) and 3D (b) of the bare ground over Vesuvius volcano in Italy, extracted from the GEDI [4] repository.

a 5 beams laser altimeter similar to MOLA. LOLA offers 6.5 billion measurements of global surface height with a vertical precision of ∼10cm, and a mean ground spatial sampling of 100 m at the equator and about 10m at the poles [18]. As on Mars, this data set is being completed with stereo images using LROC sensors from which Digital Terrain Models (DTM) at 1m/pixel at the local scale can be derived from the orbit. Mars and the Moon's surface have been visited (by landers, rovers and humans, the latter only for the Moon) and small-scale and high-resolution DTMs are also available very locally.

III. THE FARMYARD PIPELINE

We are finally in a position to introduce our FARMYARD pipeline, which is shown in Fig. 2. The pipeline starts with data collection where we collect a set of raw point cloud data covering various geolocations from planetary LiDAR repositories such as those mentioned in the previous section. We then conduct data preprocessing for them with the following three steps: ROI selection where we select subsets of points that correspond to regions of interest (ROIs) from each geolocation, coordinate reference system (CRS) unification where we project the CRSes of the raw points from the different LiDAR repositories to a uniform CRS, and raster interpolation where we interpolate irregularly-sampled points into regularly-sampled raster data. On the other hand, we separately conduct feature definition where we define the features to extract via a *feature template* on which we will elaborate later. With the data preprocessed and features defined, we conduct feature extraction where we first extract raw features from the raster data and then aggregate them to obtain the final features. Afterwards, we perform feature evaluation in two ways: First, we partition the dataset into training and testing data for Cross Validation (CV) and obtain CV scores such as Cohen's kappa coefficient [9] and F1-score which quantify the distinguishing power of the extracted features. Second, we visualize the features for manual inspection. The user can then apply the CV scores and visualized features to further analysis on which we will elaborate in Section III-F.

For the rest of this section, we will elaborate on the aforementioned procedures one by one.

A. Data Collection

The starting point of our FARMYARD pipeline is to collect raw 3D point cloud data of multiple local regions scattered across the globe from planetary LiDAR repositories such as the ones introduced in Section II. Here an important principle is to make the data as diverse as possible. For example, suppose we are trying to discover features that can distinguish between the Earth and Mars, it is imperative that we collect data from diverse geolocations and across diverse landscapes on both planets. This can give us a comprehensive representation of the two geo-objects in question. As we will show in Section III-E, this is crucial to finding features that are truly relevant to the differences between the two geo-objects.

B. Data Preprocessing

After data collection, we preprocess the collected raw data according to the following procedures, as illustrated in Fig. 3) in the case of the Vesuvius volcano.

1) ROI selection: The user may want to select regions of interest (ROIs) from the raw data. For example, a user may want to only keep a certain landscape at a geolocation where multiple landscapes co-exist, or the boundaries pre-defined by the data source are too rough for certain geo-objects. In such cases, the user can manually crop out polygonal ROIs at that geolocation. Note that if a user-selected ROI is not rectangular, we use its rectangular bounding box instead for the ensuing procedures for ease of processing, and map the extracted features back to the original non-rectangular ROI with a simple step between feature extraction and feature evaluation. See Section III-D for more.

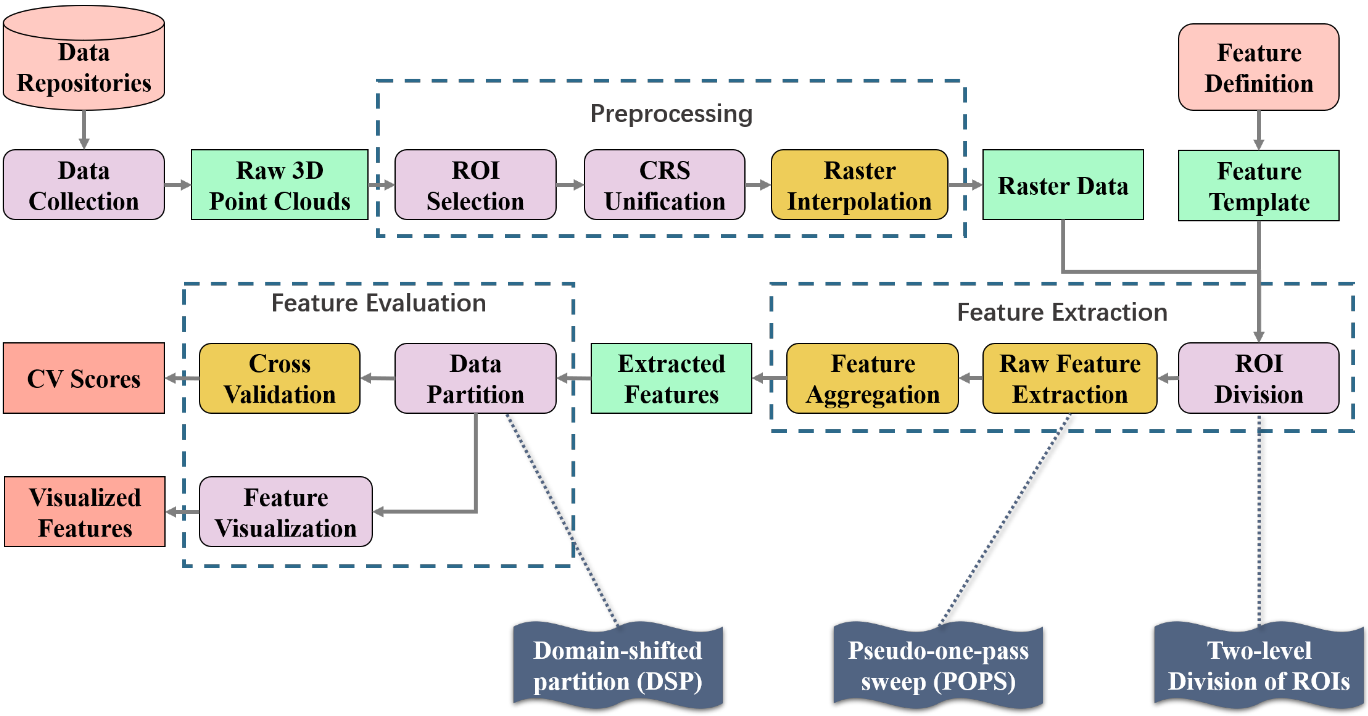


Fig. 2. The workflow of our FARMYARD pipeline, with its 3 design highlights shown in dark blue banners, and its GPU-accelerated procedures shown in golden boxes. Note that these GPU-accelerated procedures are the main efficiency bottlenecks of our pipeline, thus moving them onto the GPU makes our pipeline almost entirely GPU-based.

2) CRS unification: The raw data points in the dataset often follow different coordinate reference systems (CRS) as they originate from different data repositories produced for different planets. We need to unify the CRS so that all data points are comparable. Specifically, we recommend a Lambert Conformal Conic Alternative (LCCA) projection, as it can be well adapted to the variety of latitude's cover as well as the extent of the ROIs.

3) Raster interpolation: After CRS transformation, we interpolate the raw data in each ROI to raster with uniform resolution. The reason for raster interpolation is two-fold: First, the raw cloud is irregularly sampled, thus the sampling protocol differs from ROI to ROI. Like the discrepancy with CRS, the discrepancy with sampling needs to be addressed to make all ROIs comparable with each other. Second, for some ROIs, the raw clouds may be too sparse to extract meaningful features, thus they must be made denser with interpolation. There are many existing tools for topographical point cloud interpolation, such as IDW [11], Kriging [10], etc. We will not dive into the details of them for brevity.

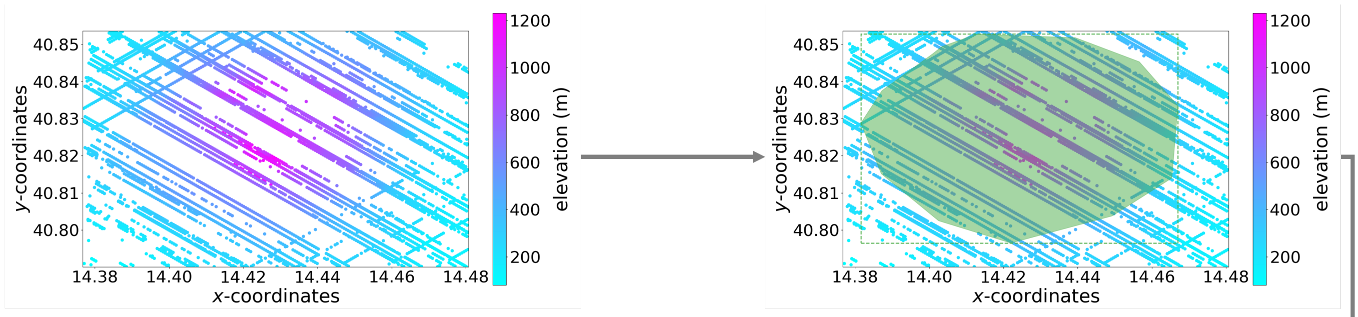
C. Feature Definition

We define the features to extract by means of a **feature template** which is a triplet $(core, dim, featScale)$ comprised of the following 3 elements:

- A **Core method *core*:** a specific feature extraction method used to extract features from a given set of 3D points. Here we provide two examples of such core methods.
  - PCA: In [12], Brodu and Lague proposed a Principle Component Analysis (PCA) based approach for feature extraction from 3D point clouds, where PCA is applied to the set of points and 3 eigenvalues λ_1, λ_2 and λ_3 are obtained which represent the variances on the 3 dimensions after PCA projection. The features are the explained variance ratios of the first 2 dimensions $\frac{\lambda_1}{\lambda_1+\lambda_2+\lambda_3}$ and $\frac{\lambda_2}{\lambda_1+\lambda_2+\lambda_3}$. In our implementation, we further transform these 2 values into their coordinates in a ternary graph, using the latter as the final features.
  - $STAT$: This method extracts statistics of the elevation (namely z) values for the set of points as feature values. Specifically, we use the following 6 statistics: standard deviation (std), minimum (min), first quartile (Q1), median, third quartile (Q3), and maximum (max)[1].
- **Dimensionality *dim*:** either *2D* or *3D*. In 2D, the core method is applied to all points in a 2D square bounding box along x and y. In 3D, the core method is applied to points in a 3D (namely x, y and z) spherical area.
- A set of **Feature scales *fScales*:** one or several length values. For each of these values, if dim is 2D, it refers to the edge size of the square bounding box; if dim is 3D, it refers to the diameter of the spherical area. Together with dim, $fScales$ define the sets of points to apply the core method to. For example, if dim is 2D and $fScales$ is 300m, 400m and 500m, the core method will be applied to points in bounding boxes with edge sizes of 300m, 400m and 500m individually, and the feature

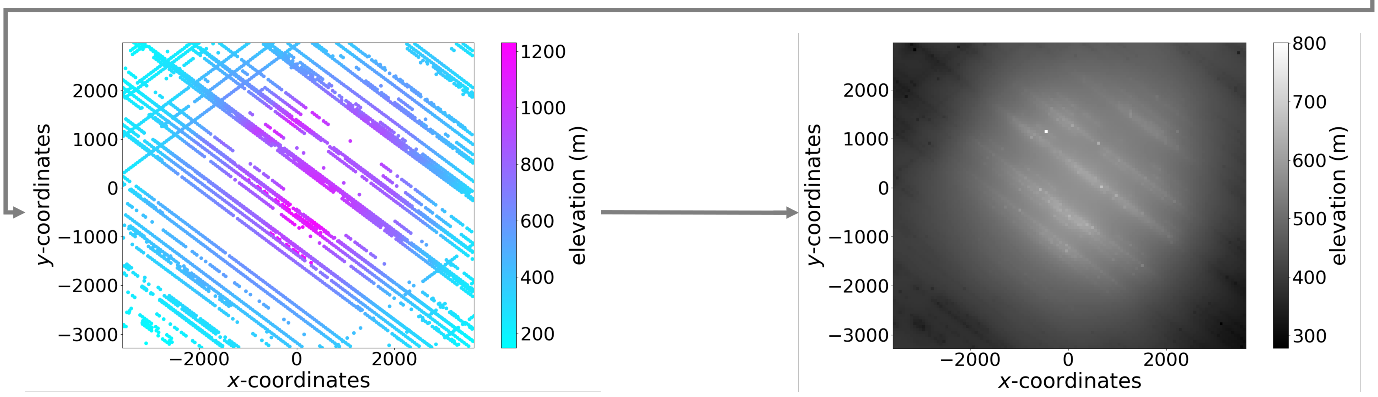
---

[1] See Section III-D for why we do not use the mean value as a feature here.



(a) Raw 3D Point Cloud

(b) After ROI selection



(c) After CRS Unification

(d) After Raster Interpolation

Fig. 3. An illustration of data preprocessing for Vesuvius volcano, with data from the GEDI repository [4]. For Region-Of-Interest (ROI) selection (b), the shaded area indicates the non-rectangular polygonal ROI selected by the user, while the rectangular area enclosed by the dashed lines indicates its bounding box.

values obtained on each feature scale are concatenated to form the final feature vector.

Before moving on, we note that we consider the location of the bounding boxes or spheres (which correspond to 2D and 3D, respectively) to be irrelevant to the definition of the features. The rationale is that a feature with high distinguishing power between multiple geo-objects should remain consistent everywhere within each of these geo-objects. Also, note that we allow for multiple feature scales. This is because the same features can exist across multiple scales, which makes a multi-scale setting better suited to capture them [12].

D. Feature Extraction

With regularly sampled raster data of all ROIs and the feature template defined, we can now conduct feature extraction. This procedure is shown in Algorithm 1. Specifically, for each ROI (line 3 in Algorithm 1), we take the following 3 steps to extract its features.

1) ROI division: For a given ROI, we opt not to extract a uniform set of features for all data points in it for 2 reasons: First, there may be so few ROIs that it is hard to directly use them as training and testing examples for ensuing Cross Validation (CV) procedure (see Section III-E). Second, the need for multi-scale features makes it impractical to extract features upon an ROI with a fixed size. In view of these, we devise a **two-level ROI division scheme** (Fig. 4) to divide the ROI into smaller sub-regions on which we extract features with the core method. Specifically, on the top level, we handle the problem of ROI scarcity in line 4 in Algorithm 1, where we divide each ROI into **examples** that will later be used for CV.

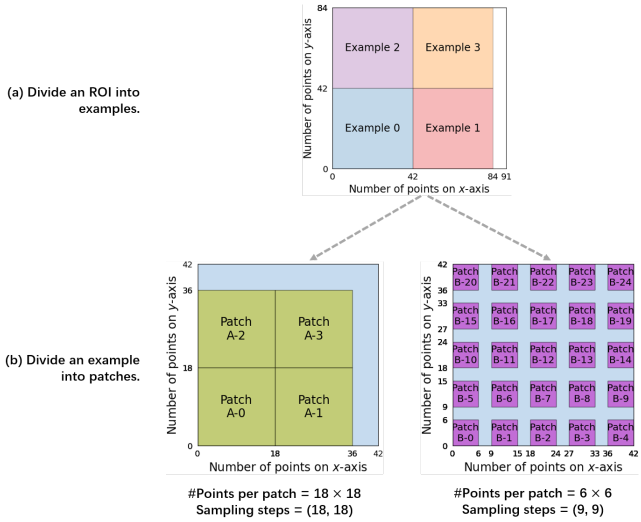

(a) Divide an ROI into examples.
Example 2
Example 3
Example 0
Example 1
Number of points on y-axis
Number of points on x-axis
0 42 84 91
(b) Divide an example into patches.
Patch A-2
Patch A-3
Patch A-0
Patch A-1
0 18 36 42
#Points per patch = 18 × 18
Sampling steps = (18, 18)
Patch B-20 Patch B-21 Patch B-22 Patch B-23 Patch B-24
Patch B-15 Patch B-16 Patch B-17 Patch B-18 Patch B-19
Patch B-10 Patch B-11 Patch B-12 Patch B-13 Patch B-14
Patch B-5 Patch B-6 Patch B-7 Patch B-8 Patch B-9
Patch B-0 Patch B-1 Patch B-2 Patch B-3 Patch B-4
0 6 9 15 18 24 27 33 36 42
#Points per patch = 6 × 6
Sampling steps = (9, 9)

Fig. 4. Two-level division of a Region-Of-Interest (ROI). (a) Dividing an ROI of size 91×84 into examples of size 42×42 with an example sampling step of 25. (b) Further dividing an example into patches of size 18×18 with a feature sampling step of 10 (*left*), and size 6×6 with a feature sampling step of 9 (*right*). Note that this is merely a toy example for illustration. Actual ROIs and examples are hundreds or thousands of times larger than what is shown here.

Algorithm 1: extractFeats($rois$, $exScale$, $exStep$, $core$, dim, $fScales$, $stsRatio$): feature extraction

```
Input  : raster data for all ROIs $rois$, example scale
         $exScale$, example sampling step $exStep$,
         core method $core$, dimensionality $dim$,
         feature scales $fScales$, feature stride-to-scale
         ratio $stsRatio$
Output: aggregated feature vectors of all examples in
         all ROIs $aggFeats$
1  $aggFeats = []$;
2  $lcv =$ getLenCoreVec($core$) ;  // The length of
     the output vector of $core$. For
     example, $lcv = 2$ for $PCA$ as it
     outputs 2 ternary coordinates.
3  foreach $roi$ in $rois$ do
       // Divide an ROI into examples.
4      $examples =$ roiToEx($roi, exScale, exStep$);
5      $curAggFeats = []$;
6      foreach $ex$ in $examples$ do
7          $featVec = []$;
8          foreach $fScalep$ in $fScales$ do
9              $fStep =$ round($fScale \times stsRatio$);
               // Divide an example into
                  patches.
10             $patches =$ exToPat($ex, fScale, fStep$);

               // Extract raw features.
11             $curRawFeats = []$;
12             foreach $patch$ in $patches$ do
13                 if $dim ==$ '$2D$' then
14                     $data = patch$;
15                 else if $dim ==$ '$3D$' then
16                     $data =$
                         sphereAroundCenter($patch$);
17                 $data =$ meanNorm($data$);
18                 $raw =$ runCore($data, core$);
19                 $curRawFeats$.append($raw$);

               // Aggregate features.
20             foreach $i = 0 : lcv - 1$ do
21                 $cur = curRawFeats[:, i]$;
22                 $agg =$ getStats($cur$);
23                 $featVec$.concat($agg$);
24         $curAggFeats$.append($featVec$);
25     $aggFeats$.append($curAggFeats$);
26     return $aggFeats$;
```

Each example is a square sub-region of the ROI that resides in one of all possible locations of a 2D sliding window, which moves rightwards and upwards from the bottom-left corner of the ROI (Fig. 4a). This process requires two user-defined parameters: the first is an example scale parameter $exScale$ that dictates the edge size of the sliding window (namely that of each example). The second parameter is an example sampling step parameter $exStep$ that dictates the stride size of the sliding window as it moves horizontally and vertically.

On the bottom level of the division, we further divide each example into **patches** (line 10 in Algorithm 1) on which we later apply the core method to. To facilitate multi-scale feature extraction, we obtain these patches by applying multiple 2D sliding windows to the example. For each window, its edge size is $fScale$ which is one of the feature scales in $fScales$ in the feature template (line 8). Similar to that used in the top level of the division, these windows move rightwards and upwards from the bottom-left corner of each example (Fig. 4b). We introduce a user-defined stride-to-scale ratio parameter $stsRatio$ to control the stride size of the sliding windows. Specifically, if the edge size of a window is $fScale$, then its stride size is $fScale \times stsRatio$ (line 9).

2) Raw feature extraction: With the ROIs divided into examples, and then patches after the two-level division, we now apply the core method to the points in each patch to obtain a set of raw features (lines 11-19 in Algorithm 1). Specifically, if dim is 2D in the feature template, all points in the patch are fed into the core method $core$; if dim is 3D, then we obtain the spherical area around the center point of the patch with a diameter of $fScale$ and apply $core$ to it. Also note that we mean-normalize the data points on each of the x-, y- and z-axes before feature extraction (line 17) to avoid interference from inter-patch (and indeed inter-example and inter-ROI) distinctions in geolocation and elevation. This is also why in the core method $STAT$ (see Section III-C) we do not extract the mean value, as it is always 0 for mean-normalized data.

3) Feature aggregation: After raw feature extraction for each patch, our next move is to obtain the final feature vector for each example. Note that it is not advisable to simply concatenate all the raw features to obtain the feature vector for two reasons: First, as mentioned in Section III-C, the feature template does not consider the locations of the bounding boxes or spheres, thus each example-level feature (namely each value in the final feature vector) should be agnostic to patch positions. Second, it is most likely that an example can have a large number of patches in it, thus a simple concatenation of all patch-level raw features will result in a very long feature vector, which can lead to the well-known curse of dimensionality [19] problem in later CV steps. In view of these, we opt to obtain the feature vector by aggregating the raw features. Specifically, in lines 20-23, for all patches of a fixed $fScale$ (namely edge size), we aggregate each dimension of their raw features using 7 statistics (line 22): mean, and the 6 statistics used in the core method $STAT$[1] (see Section III-C), namely std, min, Q1, median, Q3, max. The feature vector of each example is comprised of the aforementioned statistics for all indices in the raw features of all patch sizes in it (namely all indices of $curRawFeats$ in Algorithm 1, see line 20). Therefore, the length of this feature vector is agnostic

---

[1] Note that $STAT$ is not to be confused with the feature aggregation method we describe here. The former is a raw feature extraction method that aggregates the original 3D data points in each patch to obtain patch-level raw features, while the latter is used to further aggregate the patch-level raw features into example-level features.

to both the positions and the number of patches. The final return of Algorithm 1 is the feature vectors of all examples (line 24) in all ROIs (line 25).

Before we move on, we note 2 points: First, we discuss how to set the parameters $exScale$, $exStep$ and $stsRatio$ for the two-level ROI division scheme. We begin with the example scale $exScale$. Obviously, $exScale$ should at least be equal to the largest $fScale$ in $fScales$ so that an example can hold at least one largest patch. With this criterion met, the choice of $exScale$ is essentially a trade-off between quality and quantity: The larger $exScale$ is, the more patches it can hold, and the more robust its aggregated features are likely to be. However, this also means fewer examples for ROIs of fixed sizes, which can limit the performance of the classifiers used in CV. Similar trade-offs hold for $exStep$ and $stsRatio$, as the larger the stride sizes are, the smaller inter-dependency between adjacent examples (patches) is, and the more robust the ensuing CV process is likely to be. On the other hand, increasing the stride sizes will lead to fewer examples (patches), which will hinder classifier performance. The user needs to carefully consider such trade-offs when setting these parameters. Especially, we strongly recommend against setting $exStep$ smaller than $exScale$, as this will cause examples to overlap, breaking the minimal level of independence between examples in CV. However, we do allow patches within an example to overlap as this does not violate example-wise independence.

Second, the aforementioned feature extraction procedure only allows for rectangular ROIs. We now discuss how to cater to non-rectangular polygonal ROIs. Recall that we substituted these non-rectangular ROIs with their rectangular bounding boxes in ROI selection (see Section III-B1). To map back to the original ROIs, we simply remove all examples that are not enclosed by any of the original ROIs after feature extraction while keeping all the rest.

E. Feature Evaluation

Having obtained the feature vectors for all examples, we now move on to feature evaluation. Specifically, for a given set of features extracted under a certain feature template, suppose we wish to evaluate how well these features can distinguish between geo-objects from **Target category (TC)** (see Section I) $\mathcal{T}$, we assign a label to each example under $\mathcal{T}$. For instance, with $\mathcal{T}$ being *planets*, we label each example with the planet it comes from, such as Earth or Mars. We then partition the labeled examples into several folds, and feed them into a Cross Validation (CV) routine. The CV scores (to be discussed later in this section) are used as indicators of feature quality (namely their distinguishing power). Next, we separately discuss how to partition the labeled examples, and how to obtain the CV scores based on the partitioned data.

1) Data partition: On partitioning the examples, a straightforward solution is *Random Partition (RP)*, namely randomly assigning each example to one of K folds for CV where K is a user-defined parameter. However, RP cannot address Challenge **C3** in Section I, namely the problem with interfering factors. For instance, suppose we have examples with labels Earth and Mars, derived from the TC *planet*. For each label, we have examples that correspond to 3 *landscapes*: volcano, drainage and crater. In this case, RP is likely to result in partitions where examples with all *landscapes* coexist in both the training and testing sets, In this case, even if we get a high CV score, we cannot tell whether this is because the features can distinguish between different *planets* or *landscapes*.

To tackle this issue, we devise the following **Domain-Shifted Partition (DSP)**[1] strategy, which entails the following steps.

Step 1: Non-Target Categories (NTC) identification Identify as many **Non-Target Categories (NTCs)** as possible, with each NTC indicating a potential interfering factor for feature evaluation. For instance, if the TC is $planets$, the set of NTCs may include $landscapes$, $geolocations$, etc.

Step 2: Label assignment. Under each NTC, assign labels to all examples. For example, under $landscapes$, a label may be volcano, drainage or crater; under $geolocations$, a label may be bounding box coordinates, or a textual designation for a specific region, such as Etna for Mount Etna in Italy on the Earth, Olympus for Olympus Mons on Mars, Borku which is a region in Central Africa on the Earth, Warego for the Warego drainage basin on Mars, and Gosses_Bluff for the Goss Bluff crater in Australia on Earth. For the rest of this section, we refer to the labels under TC as **Target Labels (TL)**, and those under NTCs as **Non-Target Labels (NTL)**.

Step 3: Data partition Partition the data into K folds, such that the following 2 criteria are met:

- **TL inclusivity:** for the TC, all TLs must appear in all folds. This is the common requirement by all data partition strategies for CV: the examples in one fold must cover all TLs.
- **NTL mutual exclusivity:** *for each NTC, any NTL under it should only appear in one fold.* This is a *unique* requirement by our DSP strategy: for any pair of examples that are drawn from the training and testing sets respectively, we do NOT allow them to share the same NTL under any NTC.

For instance, suppose the TC is $planets$, the NTCs are $landscapes$ and $geolocations$, and we have examples whose labels are (in order of $planets$, $landscapes$ and $geolocations$) as follows.

- Example a: Earth, volcano, Etna;
- Example b: Earth, drainage, Borku;
- Example c: Earth, landslide, Sycamore;
- Example d: Mars, crater, Warego[2];
- Example e: Mars, volcano, Olympus;
- Example f: Mars, drainage, Warego.

Suppose $K = 2$, then there are two ways to divide these examples into 2 folds: 1) examples a, c, e for one fold,

---

[1] DSP is inspired by the inter-patient setting in electrocardiogram-based arrhythmia detection in the biomedical engineering literature [20], while its namesake comes from the domain adaptation [21] problem in machine learning, where a domain refers to a set of examples from the same distribution and different domains tend to have inter-domain distribution discrepancies that lead to significant performance decay for classifiers trained in one domain to classify examples in another domain.

[2] There are craters located near the Warego drainage basin.

examples b, d, f for the other fold; 2) examples a, e for one fold, examples b, c, d, f for the other fold. In both cases, the aforementioned 2 criteria are met: On the one hand, *TL inclusivity* is guaranteed as either one of the two folds covers both the TLs Earth and Mars. On the other hand, *NTL mutual exclusivity* is guaranteed, for each one of the NTLs under the NTC *landscapes*, namely volcano, drainage, landslide and crater, only appears in one fold, which also holds true for the NTC *geolocations*.

Unlike RP, DSP can effectively screen influence from any interfering factors indicated by the NTCs. To illustrate, in the aforementioned instance, the TLs are Earth and Mars under the TC *planets*, while the NTLs under NTC *landscapes* are volcano, landslide, drainage and crater. The coexistence of these 4 NTLs in the training and testing sets makes it hard to identify whether the features are indicative of differences in terms of *planets* or *landscapes*. However, with DSP, if any one of the 4 NTLs appears in the training set, it cannot appear in the testing set. Therefore, features that are mostly relevant to differences among the 4 NTLs will be of little use to training a classifier that will do well in testing. Rather, only features that are actually related to the differences between Earth and Mars are likely to yield high CV scores.

The DSP strategy highlights the need for diverse data, which was mentioned in Section III-A. Specifically, we need the raw data to encompass as many NTCs as possible. Only in this way can we take into account enough interfering factors to accurately evaluate the relevance of the features to the TC.

2) Cross validation (CV): After data partition, we conduct CV, performing K runs over the K folds, among which each run has a $K-1:1$ train-test split and yields a classification score. The average classification score over all runs is the final CV score. For the choice of classifiers, commonly-used options such as support vector machines (SVMs), XGBoost [22], k-nearest neighbor (k-NN), logistic regression and artificial neural networks are all valid choices.

For the choice of classification score, we recommend using a combination of Cohen's kappa coefficient [9] (κ) and F1-score, as these two metrics are both robust against imbalanced datasets, which is common in real-world scenarios. κ is a single-value metric that measures the agreement between the predicted labels and the groundtruth labels. The range of κ is [-1, 1], the higher it is, the better the classification performance. Specifically, a κ of 1 indicates perfect agreement; a κ of 0 indicates no agreement other than those that are expected by chance; a κ that is negative indicates either no relationship between the predicted and groundtruth labels, or a tendency of the two disagreeing with each other. On the other hand, F1-score is a per-label metric (that is, there is one F1-score for each label) that can reflect the classifier's accuracy in identifying examples with each label. The higher the F1-score is, the more accurate. In our case, we recommend using κ to quantify the overall quality of the features as defined by the feature template, and inspecting the F1-scores to gain deeper insight into their per-label distinguishing power.

Apart from choosing CV scores that can better cope with imbalanced datasets, it is also advisable to balance the training set before training to actively counter this problem. Possible data balancing methods include oversampling methods such as SMOTE [23] and ADASYN [24], as well as undersampling methods such as random undersampling and undersampling with instance hardness threshold [25], which are implemented in the Imbalanced-learn[1] [26] package.

3) Feature visualization: Aside from quantitative evaluation with CV, it is also important for domain experts to manually inspect the extracted features. To facilitate this, it is advisable to visualize both the raw and aggregated features across different TLs. This can be done using visualization tools such as t-distributed Stochastic Neighbor Embedding (t-SNE) [27], or adopting customized visualization methods that are compatible with the core feature extraction method.

F. Further Analysis

With the CV scores and visualized features, we can now perform further analysis as needed. Here we present 3 instances of such analysis.

1) Feature ranking: A straightforward type of analysis is to decide the best feature template among multiple candidates. For example, we can use the CV scores of these candidates to obtain an objective ranking, or inspect the visualizations to give a subjective assessment.

It is worth noting that both the CV scores and the visualization are subjected to influence from the configurations of FARMYARD (e.g. the choice of interpolation method and its parameters, the choice of data balancing method before classification, the choice of classifier and its parameters, the choice of visualization method and its parameters, etc.). Therefore, it is advisable that the user run the FARMYARD pipeline under multiple configurations and comprehensively examine the results across all configurations.

Concretely, when comparing the CV scores (especially κ), we highly recommend performing statistical tests to compare the statistical differences of the multiple candidate feature templates across all configurations. Specifically, drawing on practices in the time series data mining literature [28], [29], we first conduct a Friedman test (with a significance threshold $p = 0.05$) to check whether there are significant differences among the rankings of all candidates. If so, we then use Wilcoxon signed rank test ($p = 0.05$) with Holm correction to divide them into groups such that candidates in the same group have no significant differences, while those in different groups do. The results of the aforementioned two statistical tests can be represented by a visualization method called *Critical Difference Diagram* (CDD) [30], on which we will further elaborate in Section V-A.

2) Feature qualification: Another simple type of analysis is to decide whether a feature template can yield features whose quality meets the user's expectations. For example, the user can set thresholds for κ and the F1-scores, and judge whether a feature template is qualified by comparing its CV scores against this threshold. The user can also manually inspect the visualized features to give a subjective assessment of the feature quality. Again, it is advisable that the user conduct

---

[1] https://imbalanced-learn.org/stable/

multiple runs under different FARMYARD configurations for both CV and feature visualization.

3) Bias analysis: Another possible analysis to perform is to see if the results are biased toward certain labels, which can be achieved by looking into the per-label F1-scores. Specifically, if the F1-scores of some labels are much higher than those of the other labels, then it suffices to say that there exists a bias towards the former. However, we cannot be certain of the cause of such a bias, which may be down to the feature template, the dataset composition (especially data imbalance), the FARMYARD configurations, or a combination of some or all of the aforementioned factors. The user would probably need further investigation (e.g. scrutinizing the feature visualization results, etc.) to pinpoint the cause. Again, we recommend conducting multiple runs on different configurations. This can at least help determine whether such a bias exists only for certain FARMYARD configurations, or are common to the majority of them, thus providing clues to the cause of the bias.

Besides the aforementioned analyses, the user can perform other types of analysis as needed. It is also advisable to conduct multiple analyses for a more comprehensive insight.

IV. GPU Acceleration for FARMYARD

So far, we have discussed the general workflow of our FARMYARD pipeline without looking into its efficiency, which is most likely to be unreasonably low under a CPU-based sequential implementation. Specifically, in most cases, the computational bottlenecks of our pipeline are raster interpolation, raw feature extraction, feature aggregation and cross validation (namely the procedures highlighted by golden boxes in Fig. 2). To break these bottlenecks, we adopt CUDA-based GPU parallelism to accelerate these procedures. In particular, for the first time to our knowledge, we propose a GPU-accelerated framework for raw feature extraction. It is also worth noting that as the aforementioned procedures are the main bottlenecks of our pipeline, moving them onto the GPU essentially makes our pipeline almost entirely GPU-based.

For the rest of this section, we will first briefly introduce CUDA-based GPU acceleration, and then go through how we accelerate the aforementioned bottlenecks one by one.

A. CUDA-based GPU Acceleration

CUDA [14] is a general-purpose parallel computing platform and programming model that can effectively leverage NVIDIA GPUs for parallel processing of massive data. In particular, GPUs are especially suitable to process multiple data slices in parallel with a single set of operations, thanks to their single-instruction-multiple-threads (SIMT) architecture. Specifically, a GPU has massive numbers of threads executing in parallel on multiple streaming multiprocessors (SMs), and each thread executes the same operations on different data. These threads are grouped into **blocks**, which are assigned to SMs to be executed independently from each other. The index of each block and that of each thread in the block are identified by CUDA built-in variables $blockIdx.x$ and $threadIdx.x$[1].

Arguably, the most important factor to consider when designing CUDA-based algorithms is the memory hierarchy. Under CUDA, each GPU thread has its **registers**, all threads in a block can access the same block-wise **shared memory**, and all threads across all blocks can access the same **global memory**. To process data using CUDA, it must first be loaded into the global memory from the CPU memory, and then be fetched to registers or shared memory by individual threads. Likewise, the output of the CUDA program can only be transferred to CPU memory via global memory. The memory size increases in the order of per-thread registers, per-block shared memory and global memory, with the global memory being much larger than the former two. For example, for the Quadro RTX 6000 GPU, the maximum number of 32-bit registers per thread is 255, the maximum amount of shared memory per block is 64KB, while its global memory size is 24GB. As with access time, accessing registers and shared memory is much faster than accessing global memory. Thus, global memory accesses should be avoided whenever possible. Rather, when handling data that cannot be fitted into registers, it is advisable to cache the data in shared memory[2]. Moreover, all data in the shared memory of a block can be accessed by all threads in that block, thus shared memory offers an effective way for intra-block communication. Also, CUDA offers the **__syncthreads()** function which serves as an intra-block synchronization barrier where any thread must wait till all other threads in the same block have reached this barrier to proceed. See [14] for more on CUDA programming.

B. GPU-accelerated Raster Interpolation

As an integral part of data preprocessing, raster interpolation can be extremely time-consuming under a sequential implementation. Fortunately, GPU acceleration for topographical point cloud data has attracted much attention from researchers in recent years, and there are many off-the-shelf options [31]–[33] for the user to choose from. For brevity, we opt not to elaborate on these methods here.

C. GPU-accelerated Raw Feature Extraction

The sequential version of our feature extraction workflow, as illustrated in Algorithm 1, can be too inefficient to handle massive datasets. Its main bottleneck is the nested loops in lines 6-24. This is because while the number of ROIs is usually limited, they can often cover large areas. Therefore, for each of them, both the number of examples and the number of patches in each example can be huge, and looping over all of them can be highly time-consuming. Fortunately, note that we conduct the same operations (namely the same core method for raw feature extraction, and the same statistics for feature aggregation) to all patches and examples, which makes the feature extraction procedure amiable to the aforementioned SIMT architecture of the GPU. For the rest of this section, we

---

[1] The indexing of GPU blocks and threads can be either 1D, 2D or 3D. In this paper, we only consider the case with 1D.

[2] NVIDIA GPUs also have memory space named *local memory*, which is essentially global memory space that is private to each thread to cater to variables that cannot be fitted into registers. The size of local memory is limited and its access time is comparable with regular global memory, thus such accesses should also be avoided whenever possible.

focus on GPU-based raw feature extraction. We will discuss GPU-based feature aggregation in the next section.

For raw feature extraction, given the two-level nature of both the ROI division scheme (example-patch) and the GPU architecture (block-thread), it is intuitive to let each GPU block handle one example, and let each thread in the block handle one patch in the example. The former is advisable as all examples have the same size with the same number of patches, thus the workloads across all blocks are naturally balanced. However, it is not practicable to allocate one patch to one thread, for the number of threads in each block is limited to 1024 [14], which is usually surpassed by the number of patches. This would require each thread to iteratively process multiple patches, which brings about the following issue: under our multi-scale setting, a naïve iterative method is to process each feature scale one by one, letting each thread process one or more patches for each scale. However, this comes with the problem of idle threads. For example, suppose we have a 10000×10000 example, and the feature scales are 250, 500, 1000 with the stride-to-scale ratio set to 1 for all 3 scales, thus the number of patches for each scale is 1600, 400, 100. Suppose we use a block of 1024 threads to process it. To process the first scale with 1600 patches, we need two iterations, in the second of which only $1600 - 1024 = 576$ threads are busy, while nearly half of the available threads sit idle. Moreover, when processing the second and third scales, only 400 and 100 threads are busy, leading to an even greater waste of computational resources. It is thus highly desirable to allow different scales to be processed simultaneously to keep as many threads busy for as long as possible.

Another problem the multi-scale setting brings about is many data points in the example will be accessed multiple times, and directly accessing them from the GPU global memory is highly inefficient. Thus we need to leverage the shared memory for faster access, yet it is often the case that the example is so large that it cannot be fitted into the shared memory all at once. This calls for a well-designed shared memory management method that can both accommodate the limited size of the shared memory and the aforementioned need for simultaneous multi-scale processing.

In view of these problems, we present the GPU-based solution shown in Algorithm 2, which can extract raw features from all patches of all scales in all examples with very limited numbers of repeated accesses to global memory.

Specifically, as mentioned earlier, each GPU block is assigned a single example (lines 3-4). Raw feature extraction is performed in a **Pseudo-One-Pass Sweep (POPS)**[1] of the example (lines 5-24). This process is illustrated in Fig. 5, which revisits the example in Fig. 4b. To extract features from this example, we introduce the concept of *superpatch*, which is a rectangular area in the example that can fit into the shared memory while being able to hold the largest of the patches in the example. Specifically, the size of the superpatch is dictated by two user-defined parameters $spScaleX$ and $spScaleY$ which set the upper bounds for the edge sizes of the superpatch on the x- and y-axes. The general idea of POPS is to use several superpatches to sequentially (that is, only one superpatch is used at a time) fetch data points in the example from global memory to shared memory. Each superpatch covers several patches in the example, thus we can let threads in the block extract raw features from the covered patches while the latter reside in shared memory. This process goes on until all patches are covered, that is, until all raw

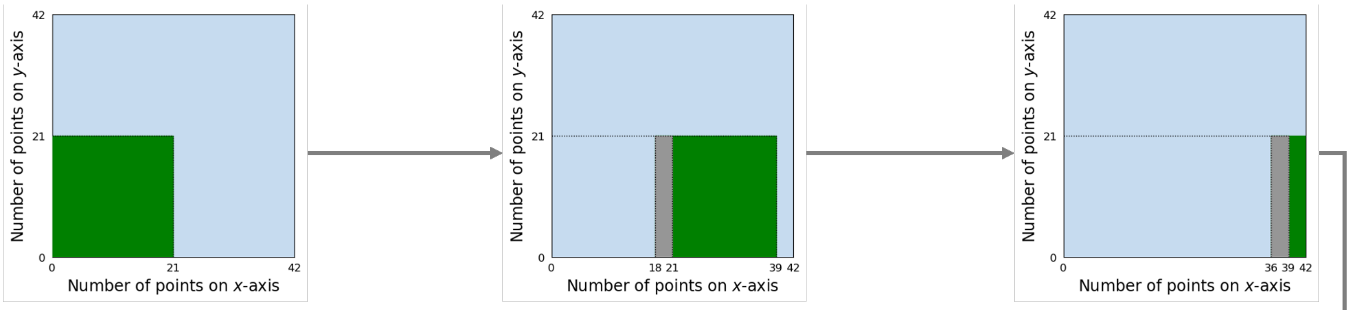
Algorithm 2: extrRawGPU($examples$, $exScale$, $core$, dim, $fScales$, $stsRatio$, $spScaleX$, $spScaleY$): GPU-based raw feature extraction

```
Input  : examples from a given ROI $examples$,
         example scale $exScale$, core method $core$,
         dimensionality $dim$, feature scales $fScales$,
         feature stride-to-scale ratio $stsRatio$,
         maximum superpatch scales on the $x$- and
         $y$-axes $spScaleX$ and $spScaleY$
Output: raw features for all patches in all examples
         in the ROI $rawFeats$

   // initialize return value in global memory
1  $l$ = getLenRawFeats($|examples|, fScales, stsRatio$);
2  $rawFeats$ = initArray($l$);
3  foreach GPU block do in parallel
4      $bid = blockIdx.x$; $ex = examples[bid]$;
       // Pseudo-One-Pass Sweep
5      $sp$ = initSupPat($ex, spScaleX, spScaleY$);
6      while True do
7          __syncthreads();
           // Apply core method to all patches in the superpatch.
8          foreach thread in block do in parallel
9              $tid = threadIdx.x$;
10             while True do
11                 $patch$ = getNextPatch($sp, tid$);
12                 if $patch \neq NULL$ then
13                     $cur$ = applyCore($patch, core, dim$);
14                     writeToRawFeats($cur, rawFeats, bid, tid$);
15                 else
16                     break;
17         __syncthreads();
           // Move superpatch.
18         $params = [sp, exScale, fScales, stsRatio]$;
19         if canGoRight($params$) then
20             $sp$ = goRight($ex, params, spScaleX$);
21         else if canGoUp($params$) then
22             $sp$ = goUp($ex, params, spScaleY$);
23         else
24             break;
25 return $rawFeats$;
```

---

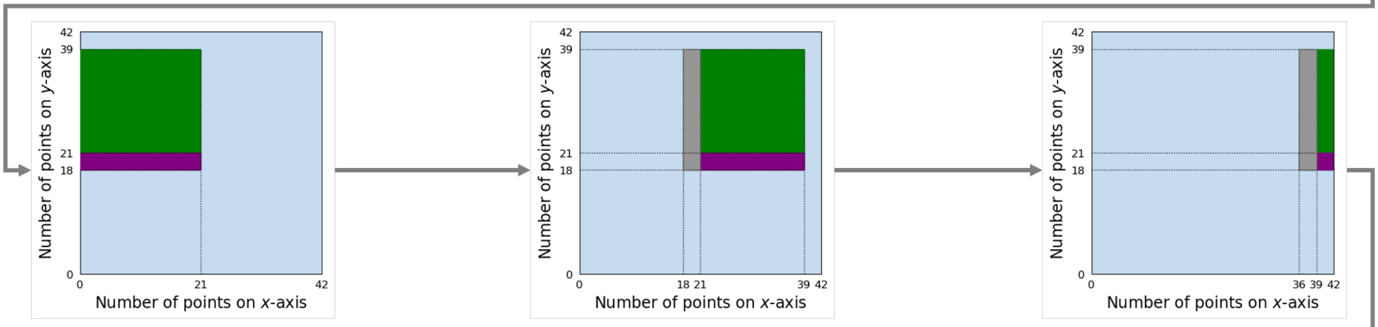
[1] We will explain the namesake of POPS later.



Superpatch 0 covering Patches A-0, B-0, B-1, B-5, B-6

Superpatch 1 covering Patches A-1, B-2, B-3, B-7, B-8

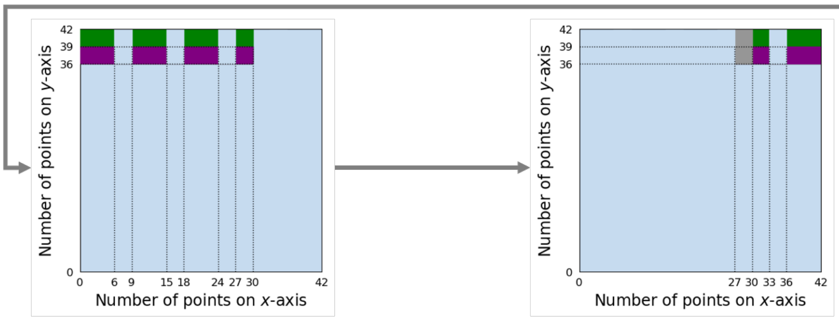
Superpatch 2 covering Patches B-4, B-9



Superpatch 3 covering Patches A-2, B-10, B-11, B-15, B-16

Superpatch 4 covering Patches A-3, B-12, B-13, B-17, B-18

Superpatch 5 covering Patches B-14, B-19



Superpatch 6 covering Patches B-20, B-21, B-22

Superpatch 7 covering Patches B-23, B-24

Fig. 5. An illustration of our Pseudo-One-Pass Sweep (POPS) framework. For the 42×42 example that is divided into 18×18 and 6×6 patches shown in Fig. 4b, we use superpatches with a maximum size of 21×21 to move over all patches in it, fetching data points from global memory to shared memory. Within the superpatchs, the green regions indicate data points that have *not* been covered by any previous superpatches. The grey regions indicate data points that have been covered by the last superpatch in the same row, but have *not* by any superpatches in the previous row. Our incremental data transfer strategy ensures that we do not need repeated global memory accesses to re-fetch these data points to the current superpatch. The purple regions indicate data points that have been covered by a superpatch in a previous row. *Only* these data points require repeated global memory accesses for the current superpatch.

features are extracted for all the patches. Note that throughout this process, all superpatches reside in only one chunk of shared memory with a fixed size.

As a concrete example, we look into Fig. 5. Here, with $spScaleX$ and $spScaleY$ being both 21, the first superpatch (Superpatch 0) is initiated at the bottom-left corner of the example (line 5 in Algorithm 2). This can be done by letting each thread in the block transfer one or several data points from global memory to the chunk of shared memory dedicated to the superpatch. Starting here, we iteratively conduct the following 2 steps (lines 6-24).

Step 1: apply the core method to all patches of all scales in the current superpatch (lines 8-16 in Algorithm 2). Here we let each thread extract features for one or several patches. For example, in Fig. 5, Superpatch 0 covers Patches A-0, B-0, B-1, B-5, B-6. Suppose we have 3 available threads. Here we let each thread handle each one of the first 3 patches. Plus, we let the first 2 threads handle the remaining 2 patches B-5 and B-6. Note that we allow different threads to handle patches of different feature scales simultaneously, thus avoiding the aforementioned waste of resources problem with sequential processing of each feature scale[1].

Step 2: move to the next superpatch (lines 18-24 in Algorithm 2). There are two ways of moving. One is the *go-right* move (lines 19-20), namely, move to the next superpatch in the same row along the x-axis. The x-coordinate of the

---

[1] In reality, to avoid running into the *bank conflict* [14] problem with shared memory, we may let a few threads sit idle while the others finish processing patches with the previous feature scale. The number of such idle threads is small (smaller than 16), leading to minimal waste of resources.

leftmost points in the next superpatch is that of the first patch(es) in this row that the current superpatch fails to cover. For example, in Fig. 5, the first patches that Superpatch 0 fails to cover on the x-axis is A-1 and B-2, whose leftmost x-coordinate is 18, which is that of Superpatch 1. Go-right moves continue till the last possible superpatch in the current row, namely Superpatch 2.

For the go-right move, we can use a simple *incremental data transfer* (Fig. 6) strategy to negate the need for repeated global memory access. Specifically, we store the data points in each superpatch in the shared memory such that the y index changes faster than the x index. This means that when we perform a go-right move, if the next and the current superpatches have any overlap, that is, the first few columns of data points in the next superpatch are also the last few columns in the current one, these data points already reside in a contiguous sub-chunk of shared memory. Hence, we do not need to reload these points from global memory. Rather, we can simply move the starting index of the next superpatch to where these data points currently reside in the shared memory. For example, in Fig. 6, as the first 3 columns of Superpatch 0 in Fig. 5 are also the last 3 columns of Superpatch 1, and is already in a sub-chunk of shared memory with the starting index $18 \times 21 = 378$. we simply move the starting position of Superpatch 0 to 378 without reloading these 3 columns. For the rest of the new superpatch, we append these data points to the existing ones. If we hit the end of the chunk of shared memory for superpatches, we simply re-use the indices from the beginning of the chunk. For example, for Superpatch 1, the columns other than the first 3 ones are stored in the positions with indices 0 to $18 \times 21 - 1 = 377$. Also, if the last superpatch of the current row is smaller than the rest, such as the case with Superpatch 2, we can simply keep part of the shared memory vacant.

The second way of moving is the *go-up* move (lines 21-22 in Algorithm 2). When the go-right moves hit the end of the current row, we re-initiate the superpatch at the leftmost position of the next row. The y-coordinate of the points at the bottom of the current row is the minimum y-coordinate of the points in the patches that have not been covered by previous rows. For example, in Fig. 5, when moving from Superpatch 2 to 3, this minimum y-coordinate is 18, which is that of the points at the bottom of Patches A-2, A-3, B-10, B-11, B-12, B-13 and B-14. Thus, Superpatch 3 starts at 18 on the y-axis.

After the go-up move, we can again perform the go-right move till the end of this row, conducting feature extraction for all patches in this row along the way. Then we can again perform the go-up move. This iterative process stops when it is no longer possible to perform either the go-right or go-up moves (lines 23-24 in Algorithm 2), at which point all patches in the example have been processed.

Looking back on the entire Pseudo-One-Pass Sweep, we reach a point where we can explain its namesake. By *one-pass*, we mean that unlike the naïve approach where we scan the global memory once for each feature scale, we can now *logically* perform only one scan using the superpatches to achieve multi-scale feature extraction; by *pseudo*, we mean while it is *logically* possible to scan only once, in practice we need repeated global memory access for overlapping data points in superpatches from different rows (the purple regions in Fig. 5). However, the number of such points is usually very limited compared to the total number of points in the example. Also, it is worth noting that our incremental data transfer strategy ensures that *no repeated global memory access is required for a single row*, except for those data points that have appeared in a previous row. This can greatly reduce the total number of repeated global memory accesses.

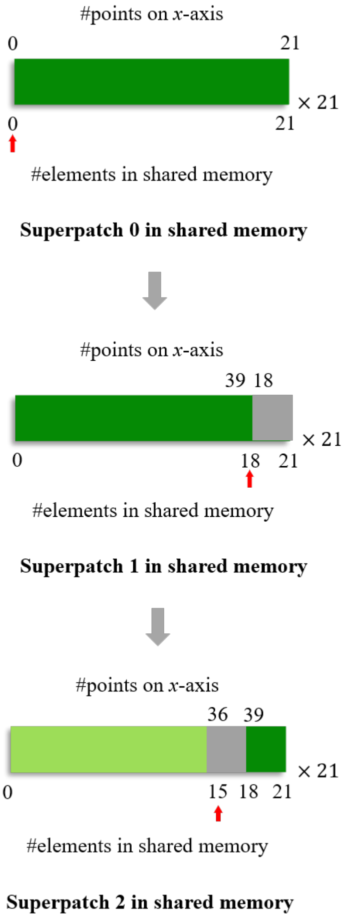

#points on x-axis
0
21
× 21
0
21
#elements in shared memory
Superpatch 0 in shared memory
#points on x-axis
39
18
× 21
0
18
21
#elements in shared memory
Superpatch 1 in shared memory
#points on x-axis
36
39
× 21
0
15
18
21
#elements in shared memory
Superpatch 2 in shared memory

Fig. 6. An illustration of shared memory usage for Superpatches 0-2 in Fig. 5 under incremental data transfer. Green indicates new data points that require global memory access. Grey indicates data that is already in shared memory. Light green indicates unused shared memory space. Red arrow indicates where the first point in the superpatch is stored.

Before moving on, we note 2 points: First, we provide advice on how to set the superpatch size parameters $spScaleX$ and $spScaleY$. Obviously, both of these parameters should be larger than the largest feature scale to accommodate at least one patch with the largest size. Moreover, we would like the superpatch to be as large as possible given a fixed number of bytes available in the shared memory. This can be achieved by making $spScaleX$ and $spScaleY$ as large as possible while keeping their difference as small as possible.

Second, we look into whether the choice of core methods is significantly limited by the relatively scarce memory resource on the GPU. Fortunately, the answer is no. Specifically, in Algorithm 2, we let each thread execute the core method for a single patch at a time. The patch itself, which holds the raw raster data, already resides in shared memory. Therefore, the only remaining concern is assigning the extra workspace memory needed by the core method to process the raw data. Given the highly limited size of per-thread registers and per-block shared memory, one simple solution is to relegate the workspace to global memory as needed. Considering the fact that modern NVIDIA GPUs have (tens of) gigabytes of global memory, this solution will suffice for the vast majority of (if not all) core methods that can be run on the CPU. However, as was mentioned in Section IV-A, global memory is slow to access and using it can limit the overall efficiency of raw feature extraction. Rather, we prefer to assign the workspace memory within the per-thread registers, or use both the registers and a small proportion of the shared memory. This would normally require the workspace to be of $O(1)$ size (namely constant size), regardless of the number of points in the raw data. As it turns out, in many cases, this is achievable. To illustrate, we look into the two core methods mentioned in Section III-C, and see how they can be implemented with constant workspace and relatively low time complexity.

- PCA: Applying PCA to a patch of data points entails 3 steps:
  1) Obtain the covariance matrix $\mathbf{C}$ of the patch. Suppose the patch contains n data points, each being a 3D coordinate, then the patch can be represented as an $n \times 3$ matrix $\mathbf{M} = [\mathbf{m}_1^{\mathrm{T}}, \mathbf{m}_2^{\mathrm{T}}, \mathbf{m}_3^{\mathrm{T}}]$ where $\mathbf{m}_i^{\mathrm{T}} = [m_{i,1}, m_{i,2}, \ldots, m_{i,n}]^{\mathrm{T}}$ $(i = 1, 2, 3)$ are the columns of $\mathbf{M}$, and $\mathbf{C}$ is a 3×3 matrix where the element at position (i, j) is
  $$c_{i,j} = \frac{\sum_{k=1}^{n} m_{i,k} m_{j,k}}{n-1} \tag{1}$$
  Note that $c_{i,j}$ can be obtained in $O(n)$ time with $O(1)$ workspace as we only need to keep one $m_{i,k}$ and one $m_{j,k}$ in the workspace at a time. As there are $3 \times 3 = O(1)$ elements in $\mathbf{C}$, it takes $O(n)$ time and $O(1)$ space to obtain $\mathbf{C}$.
  2) Obtain the eigenvalues λ_1, λ_2 and λ_3 of $\mathbf{C}$ and the explained variance ratios $\frac{\lambda_1}{\lambda_1+\lambda_2+\lambda_3}$ and $\frac{\lambda_2}{\lambda_1+\lambda_2+\lambda_3}$, which takes $O(1)$ time and workspace as the size of $\mathbf{C}$ is constant.
  3) Transform the explained variance ratios to their coordinates in a ternary graph, which also takes $O(1)$ time and workspace as there are only $2 = O(1)$ values to transform.

  All in all, the entire process takes $O(1)$ workspace and $O(n)$ time.
- $STAT$: Applying $STAT$ to a patch of n data points entails calculating their std, as well as min, Q1, median, Q3 and max which are order statistics. Among them, std (denoted as σ) can be calculated as
  $$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(z_i - \mu)^2} = \sqrt{\frac{\sum_{i=1}^{n} z_i^2}{n} - \mu^2} \tag{2}$$
  where z_i is the elevation value of the i-th point in the patch and μ is the mean of elevation values of all points in the patch, which can be calculated as
  $$\mu = \frac{\sum_{i=1}^{n} z_i}{n} \tag{3}$$
  It is easy to see that we simply need to obtain the sums $\sum_i z_i^p (p = 1, 2)$ to further calculate std in constant workspace and time, and we can obtain them in $O(n)$ time and $O(1)$ workspace, as we only need to keep the sum so far and the next z_i value in the workspace at a time. Thus, it takes $O(1)$ workspace and $O(n)$ time to obtain std.
  For the order statistics, we can obtain them using some selection algorithm which usually comes with a space-time trade-off. In our implementation, we use the *basic algorithm* proposed in [34], which uses constant workspace and has a time complexity of $O(nlgn)$, which is usually fast enough as it is unlikely that the patches are excessively large. In the rare cases that this happens, Munro and Raman [34] also provided methods with lower time complexity with slightly higher demand for workspace memory. We omit discussion of these methods for brevity.

In conclusion, by implementing the core method in a way that only requires $O(1)$ workspace, which is achievable in many cases, or by simply relegating the workspace to global memory, we can run the vast majority of (if not all) core methods on the GPU.

D. GPU-based feature aggregation

We now discuss GPU acceleration for feature aggregation. From Section III-D (in particular lines 20-23 in Algorithm 1), we can conclude that the aggregated features for an ROI can be divided into segments with an ID of $(ex, fScale, i)$, in which ex is the ID of an example in the ROI, $fScale$ is one of the feature scales, and i is an index of the output vector of the core method. Each segment consists of the 7 statistics mentioned in Section III-D. Among them, mean and std can be obtained in constant time if we pre-calculate the two sums mentioned in the discussion on $STAT$ in the previous section (although here the values to sum up are raw feature values, rather than elevation values). These sums can be obtained by CUDA-based parallel reduction [35]. For the order statistics, we can first sort the raw features that correspond to the current segment, which can be performed on GPU using existing software tools such as Thrust[1] [36] and CuPy[2] [37], and then trivially obtain them in constant time.

E. GPU-accelerated Cross Validation

For Cross Validation (CV), we mainly apply GPU acceleration to the classifiers. Here we have a wide range of open-source GPU-based implementations for a variety of classifiers. For example, the CuML[3] [38] library includes GPU-accelerated implementations of multiple commonly-used

---

[1] https://developer.nvidia.com/thrust

[2] https://cupy.dev/

[3] https://github.com/rapidsai/cuml

classification models; the ThunderSVM[1] [39] library provides GPU-based implementations of support vector machines (SVMs); the XGBoost[2] [22] library has built-in support for GPU acceleration; k-nearest neighbor is implemented in CUDA-based packages such as KNN_CUDA[3]; deep learning frameworks such as PyTorch[4] [40] and Tensorflow[5] [41] can be used to implement logistic regression and artificial neural networks executable on GPUs.

V. EXPERIMENTS

For empirical study, we conduct 2 sets of experiments: First, we present a demo of our FARMYARD pipeline to showcase its utility in a real-world geoscientific research project. Second, we conduct experiments on both real and simulated large planetary LiDAR datasets to demonstrate the efficiency of our Pseudo-One-Pass Sweep (POPS) based raw feature extraction framework. All experiments were run on a Ubuntu 18.04 server with 2 NVIDIA Quadro RTX 6000 GPUs (only one is used at a time), 1 AMD EPYC 7402P 24-core CPU @2.8GHz, and 125GB memory. All CPU-based code is implemented in Python 3.8, while all GPU-based code is implemented in CUDA-C [14] and linked with the Python code using the PyCUDA[6] [42] package. All results are averaged over 10 runs.

A. PARKER: A Demo of the Utility of FARMYARD

In this section, we showcase the utility of our FARMYARD pipeline in an ongoing geoscientific research project called Planetary lidAR seeKing for lifE signatuRe (PARKER), where we are interested in discovering topographical features related to terrestrial life activities. To do so, we attempt to find features that can distinguish between the Earth and other planets with no known life signs (currently we focus on Mars) by analyzing planetary LiDAR data. Specifically, we follow our FARMYARD pipeline step by step, with notable settings for each step listed as follows.

1) *Data collection:* We collect raw points of various regions scattered across the globe on the Earth and Mars (Fig. 7). The reason why these regions have been chosen is that they encompass 4 landscapes that are present on both the Earth and Mars, which are volcanoes, drainage systems, landslides (or rock avalanches) and impact craters. For the Earth, we use the GEDI [4] repository (since 4/2019); for Mars, we use the MOLA [1] repository (1997-2006). See Section II for a more detailed description of these repositories.
2) *Preprocessing:*
    - *ROI selection:* From each of the aforementioned regions, we manually select one or more non-rectangular ROIs, each bounding a sub-region with only one dominant landscape.
    - *CRS transformation:* As mentioned in Section III-B, we apply the Lambert Conformal Conic Alternative (LCCA) projection to the selected ROIs.
    - *Raster interpolation:* For each transformed ROI, we interpolate them to raster data with 60m resolution with the IDW [11] algorithm. IDW has two parameters: search radius r and power parameter α. We fix the former to 10,000m, while setting the latter from one of 1 : 5.
3) *Feature Definition:* For the feature template, we consider 2 core methods, 2 dimensionalities, and 3 feature scales.
    - Core method $core$: We consider PCA and $STAT$, which have been introduced in Section III-C.
    - Dimensionality dim: We consider both 2D and 3D.
    - Feature scales $fScales$: We consider 3 options, each of which is a group of 3 feature scales: (300m, 400m, 500m) which we designate as the $small$ group, (700m, 800m, 900m) which we designate as the $medium$ group, and (1100m, 1200m, 1300m) which we designate as the $large$ group.

    Combining these options leads to $2\times2\times3=12$ candidate feature templates. For the rest of this section, we refer to each template in the format of $\{core\}-\{dim\}-\{fScales\}$. For example, $PCA-2D-small$ indicates the feature template with the core method PCA, the dimensionality 2D, and the feature scales $small$.
4) *Feature extraction:*
    - *Raw feature extraction:* We set both the example scale $exScale$ and example step $exStep$ to 2000m. We fix the feature stride-to-scale ratio $stsRatio$ to 0.5 for all 9 feature scales that are included in the aforementioned options for $fScales$. Also, we implement the raw feature extraction in such a way that it is executed for all 9 feature scales in a single run on GPU (rather than one separate run for each option of feature scale). This can maximally leverage the ability of our POPS framework to handle multiple scales.
    - *Feature aggregation:* We use the CuPy [37] package to sort raw features in each segment (see Section IV-D) before we pass them on to our own CUDA-C code.
5) *Feature evaluation:*
    - *Data partition:* For our Domain-Shifted Partition (DSP), since we are trying to find features related to inter-planet differences, we set the Target Category (TC) to $planets$ with the Target Labels (TLs) being Earth and Mars. For Non-Target Categories (NTCs), we consider $landscapes$ and $geolocations$. The Non-Target Labels (NTLs) for the former are volcano, drainage, landslide and crater, and those for the latter are Etna, Emi_Koussi, Vesuvius, Elysium, Olympus, Pavonis, Borku, Ngazima, Sycamore, Durius, Elaver, Warego, Blackhawk, Socompa, Coprates, Ganges, Ius and Gosses_Bluff. The number of folds K is set to 3, as larger settings can not lead to partitioned datasets that meet the criteria of DSP. The numbers of examples

---

[1] https://github.com/Xtra-Computing/thundersvm

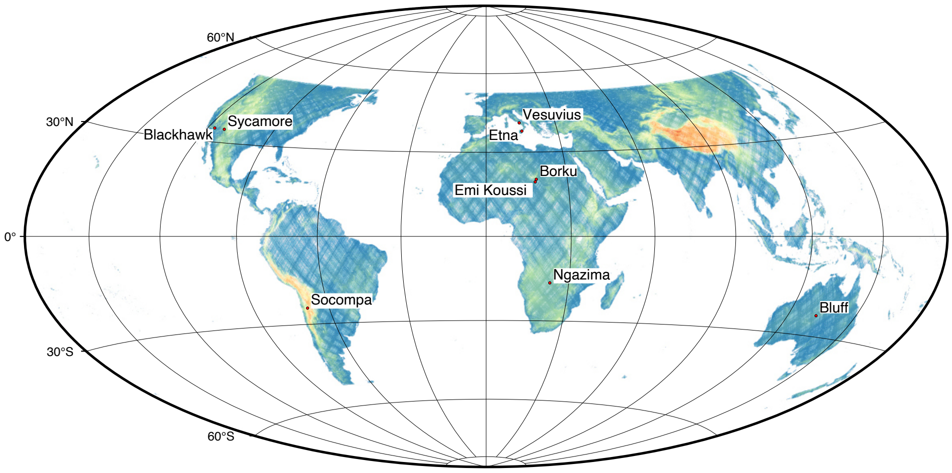
[2] https://xgboost.readthedocs.io/en/stable/index.html

[3] https://github.com/unlimblue/KNN_CUDA

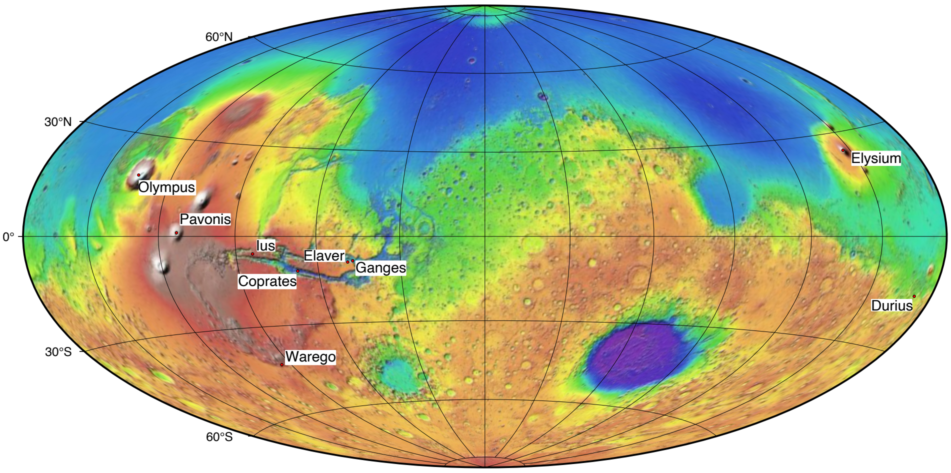
[4] https://pytorch.org/

[5] https://tensorflow.org/

[6] https://pypi.org/project/pycuda/



(a) Earth



(b) Mars

Fig. 7. Geolocations of the regions on the Earth (a) and Mars (b) currently studied in the PARKER project.

in all folds are shown in Table I[1].

- *Cross Validation (CV):* For data balancing, we consider 5 balancing methods: no balancing, 2 oversampling methods which are SMOTE [23] and ADASYN [24], as well as 2 undersampling methods which are random undersampling (RUS) and undersampling with instance hardness threshold (IHT) [25]. For the latter four, we use the implementation provided by the Imbalanced-learn [26] library. For the choice of classifiers, we apply 8 classifiers, including 4 SVM classifiers with different kernels (linear, RBF, polynomial, and Sigmoid) which are implemented in the ThunderSVM [39] package, XGBoost [22] and 3 k-NN classifiers (the number of neighbors k are set to 1, 5 and 10) which are implemented using the KNN_CUDA package (see Section IV-E).
- *Feature visualization:* For feature visualization, we use the t-SNE algorithm [27], which is a dimensionality reduction algorithm that can facilitate the visualization of high-dimensional datasets in 2D space. In our case, we visualize both the raw and aggregated features extracted using each feature template for all examples. The raw feature vector for each example is obtained by concatenating the features of all patches in it. t-SNE has an expected density parameter $perplexity$ that can affect the visualization result by controlling the number of points used as nearest neighbors (see [27] for details). Here we consider 3 values for $perplexity$: 10, 30, 50. For visual clearance, we keep the number of examples for each target label under 500 within the visualization by random sampling.

Note that in some of the aforementioned steps, we consider multiple FARMYARD configurations, as was recommended in Section III-F. Specifically, for raster interpolation, we consider 5 configurations as dictated by the 5 different IDW $alpha$ settings; for data balancing, we consider 5 configurations as dictated by the 5 balancing methods; for CV, we consider 8 configurations as dictated by the 8 classifiers; for visualization,

---

[1] Note that the data is highly imbalanced. We will explain why it was hard for us to obtain a balanced dataset in Section V-B.

TABLE I
NUMBER OF EXAMPLES FOR CROSS VALIDATION

Fold	**Target Labels (TC: planets)**	**Non-Target Labels**		**Number of examples**		
		NTC: landscapes	**NTC: geolocations**			
1	Earth	volcano	Etna	124	766	83249
			Emi_Koussi	640		
			Vesuvius	2		
	Mars	volcano	Elysium	5631	82533	
			Olympus	59985		
			Pavonis	16917		
2	Earth	drainage	Borku	206	5224	10762
			Ngazima	4998		
			Sycomore	16		
		crater	Gosses_Bluff	4		
	Mars	drainage	Durius	1318	5538	
			Elaver	1810		
			Warego	1643		
		crater	Durius	457		
			Warego	310		
3	Earth	landslide	Blackhawk	3	64	2198
			Socompa	61		
	Mars	landslide	Coprates	551	2134	
			Ganges	1077		
			Ius	506		

we consider 3 configurations as dictated by the 3 perplexity values. This leads to $5 \times 5 \times 8 = 200$ configurations for CV, and $5 \times 3 = 15$ configurations for feature visualization.

Next, we show how we can carry out the 3 further analysis tasks mentioned in Section III-F, based on the CV and visualization results.

1) Feature ranking: We would like to first see which ones among the 12 candidate feature templates can best distinguish between Earth and Mars, which may be indicative of their relevance to terrestrial life activities. To this end, we look into the CV results shown in Fig. 8, from which it is very hard to decide a clear winner. Therefore, as was mentioned in Section III-F, we perform two statistical tests over the κ scores of the 12 feature templates across all 200 CV configurations, obtaining the Critical Difference Diagram (CDD) [30] shown in Fig. 9. In the CDD, each feature template has a *mean rank* (the values over the lines connected to templates) obtained from the Friedman test, and the smaller the rank value (that is, the more to the right in the CDD), the better CV results the template has yielded. As is shown in Fig. 9, $PCA-2D-small$ has led to the best overall results.

Next, we examine whether there are statistical differences between one feature template to another. In other words, if one template is better than another, we wish to know if it is *significantly* better. In the CDD, feature templates are grouped into *cliques* as indicated by the **bold** bars. Those in the same clique, such as $STAT-2D-medium$ and $STAT-3D-medium$, as well as $PCA-3D-large$ and $PCA-2D-large$ in Fig. 9, have no statistically significant differences, while the rest do. In particular, $PCA-2D-small$ is significantly better than all the other feature templates, which makes it the sole winner.

Finally, from Fig. 9 we can make 2 interesting observations besides deciding the winning feature template: First, PCA generally outperforms $STAT$ for the same group of feature scales (except $large$). Second, $small$ generally outperforms $medium$, and $medium$ generally outperforms $large$, no matter which core method is used.

2) Feature qualification: While we already know that $PCA-2D-large$ significantly outperforms the remaining 11 feature templates, we still need to examine whether its performance is good enough to effectively distinguish between the Earth and Mars. To do so, we look into its κ scores across all 200 configurations, whose mean, standard deviation, minimum and maximum values are $0.139, 0.114, 0.478, -0.240$, respectively. Unfortunately, while there is no universally accepted way to interpret the "goodness" of κ scores, the aforementioned values are most certainly not good.

This assessment is corroborated by the F1-scores on the two target labels and the feature visualization results: For the F1-scores, the mean, standard deviation, minimum and maximum values for the F1-scores on Mars are $0.793, 0.143, 0.953, 0.168$, respectively, while those on Earth are $0.183, 0.113, 0.496, 0.003$. The F1-scores on Mars seem reasonably good, but those on Earth are poor. For the feature visualizations, the results are shown in Fig. 10. For brevity, we only present the results with IDW $\alpha = 1$, relegating the remaining results to Fig. 18 and 19 in the Appendix. As is shown, in neither the visualizations of raw and aggregated features are the two target labels well-separated. Moreover, in most cases, the intra-class distribution is highly heterogeneous, which indicates that $PCA-2D-small$ can not lead to low-variance representations for either the target labels. These observations also hold true for cases where IDW $\alpha \neq 1$.

In conclusion, $PCA-2D-small$, and thus all 12 feature templates, have so far failed to produce features that meet our

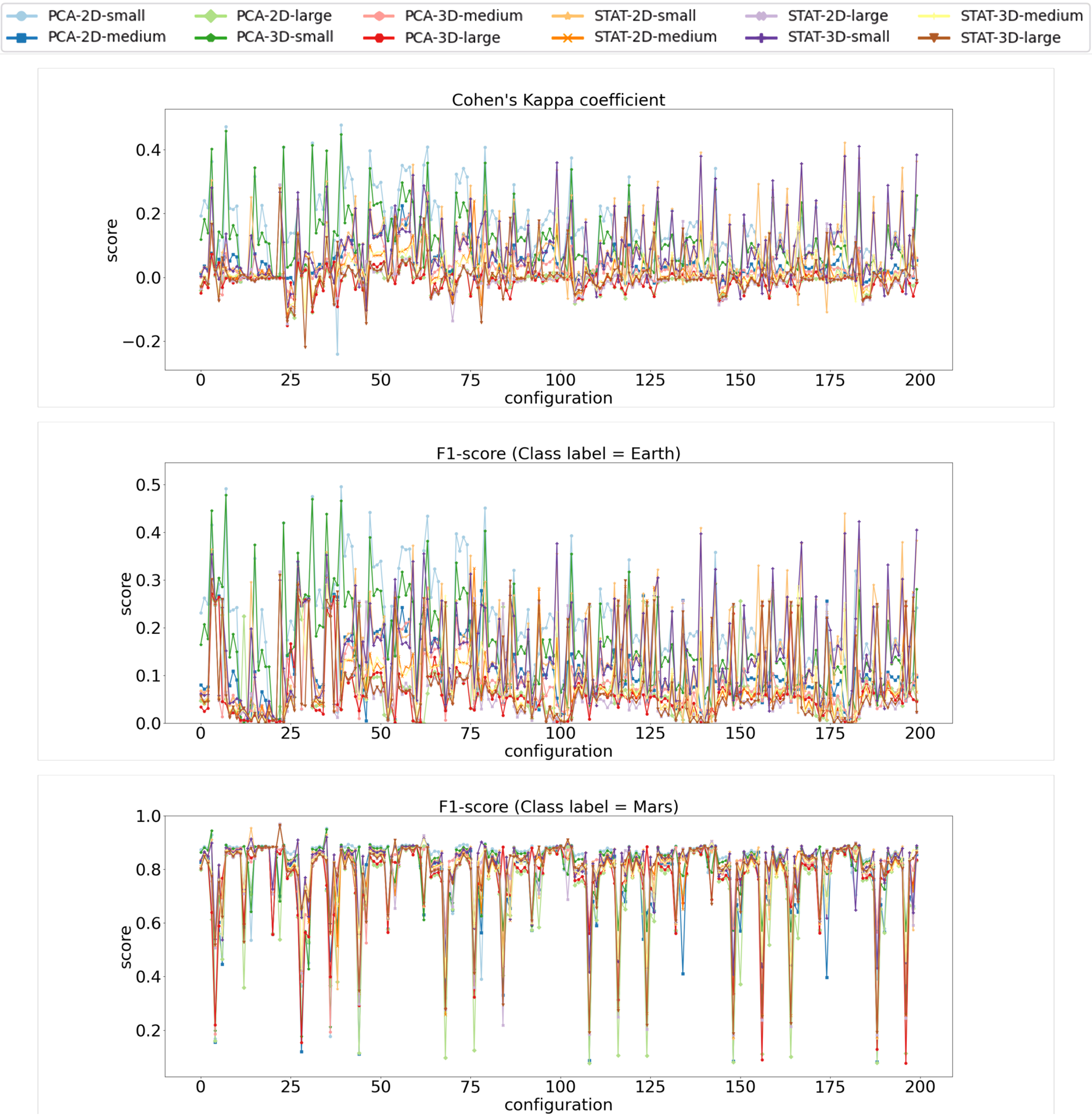


Fig. 8. Raw cross validation results of the 12 feature templates across 200 FARMYARD configurations. The name of the configurations has been omitted for visual clearance.

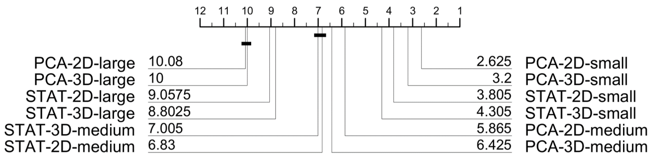


Fig. 9. Critical difference diagram of the cross validation κ scores of the 12 feature templates across 200 FARMYARD configurations. The more to the right, the better results a feature template has yielded. Those in the same clique (indicated by **bold** bars) have no statistically significant differences.

criterion when it comes to distinguishing between the Earth and Mars. Again we note that none of the 12 feature templates in question should be considered contributions of our paper. Rather, our main contribution is a novel FARMYARD pipeline that has worked as expected, helping us discover that none of the 12 templates we have currently considered are likely to lead us to further insights into the inter-planetary topographical differences related to life activities. That being said, we are encouraged by the superior performance of PCA to $STAT$. As will be discussed in the next section, we now have a preliminary hypothesis on why PCA has led to poor results and are working to improve it accordingly.

3) Bias analysis: While $PCA-2D-small$ has generally led to underwhelming results, we are curious to see whether these have to do with its bias towards one of the two target labels. As was previously mentioned, the average F1-scores for Earth and Mars are 0.183 and 0.793. This strongly suggests

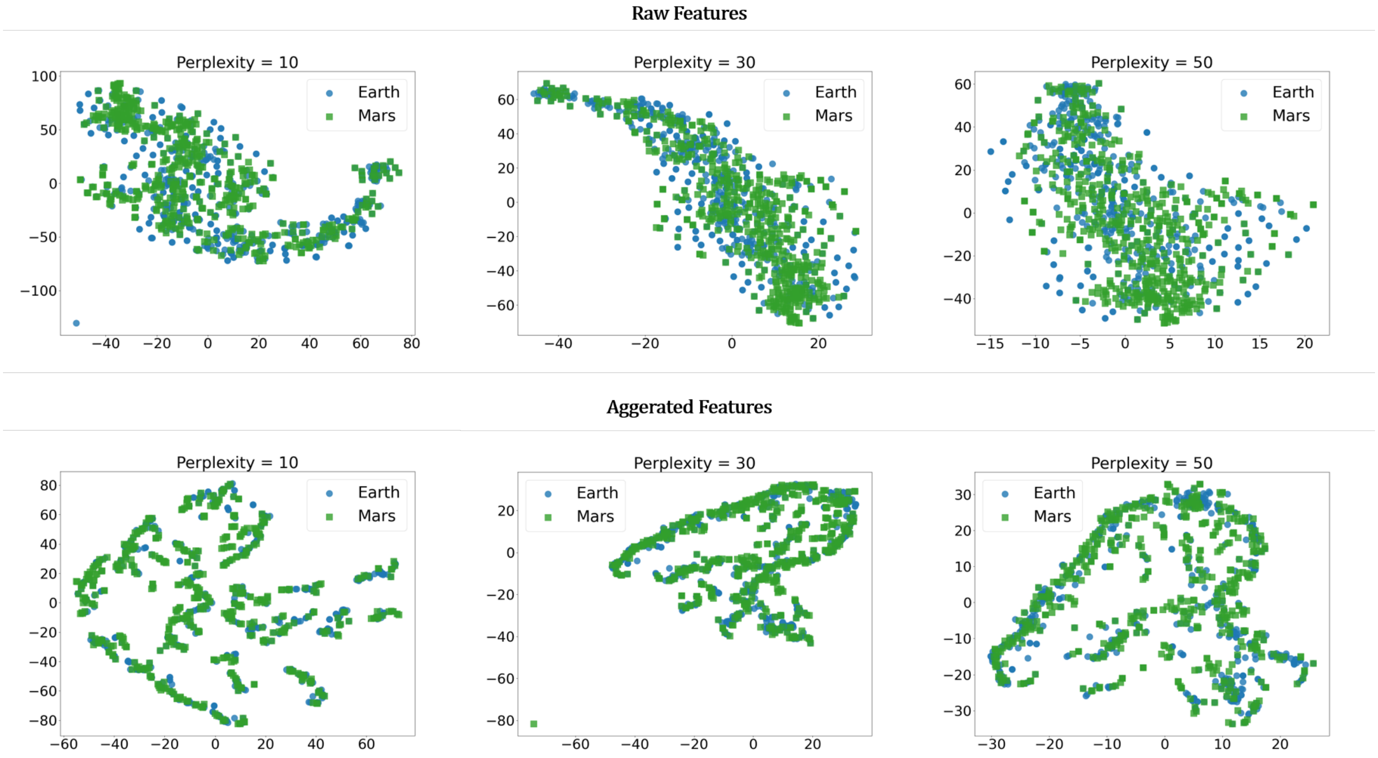


Fig. 10. Feature visualization for the feature template $PCA-2D-small$ with IDW $\alpha=1$. See Fig. 18 and 19 in the Appendix for cases where α is set to 2, 3, 4, 5.

that there is a bias towards Mars. In fact, this observation is not only true to $PCA-2D-small$, but also true to the other feature templates. The reason behind this is likely to be the fact that we have far more training examples for Mars than Earth, and the data balancing techniques have failed to compensate for this.

B. *Discussions*

From the results shown in the last section, we identify the following issues that future researches on planetary LiDAR may address.

1) The need for novel core methods: From the generally poor results yielded by the 12 feature templates we have considered, it is obvious that there is a need for novel core feature extraction methods for planetary LiDAR, at least for discovering the differences between the Earth and Mars. In fact, we are surprised to find that there are very few researches (if not none) for feature extraction from planetary LiDAR. The most closely related work we could find was by Brodu and Lague [12] who proposed the PCA method. However, that work was not originally dedicated to space-borne planetary LiDAR. Rather, it was intended for close-up terrestrial laser scanning (TLS) of natural scenes such as rivers and cliffs. We believe there is an urgent need for novel feature extraction methods that are more adaptive to the characteristics of planetary LiDAR.

In particular, we believe there is great room for improvement for PCA. Specifically, a critical difference between planetary LiDAR and close-up TLS is that for a given pair of (x, y) values, planetary LiDAR only allows for one z value, which is the elevation of the highest point at that coordinate. Close-up TLS, on the other hand, allows for multiple readings for the object at that particular coordinate. For example, suppose there is a tree at the coordinate, we can obtain a series of readings from the top of the tree to its trunk all the way to ground level. The extra readings provide more information than the single elevation reading of planetary LiDAR. We are currently working on improving PCA to adapt to this difference. The general idea is that for each (x, y) coordinate, apart from keeping the original elevation value, we add more z values below the elevation of the highest point, thus mimicking the characteristics of close-up TLS.

2) The need for denser raw data: As was mentioned previously, the results we have obtained are highly biased towards Mars, which is likely caused by far fewer examples from the Earth than Mars. Admittedly, the inclusion of very large Martian geo-objects such as Olympus Mons has played a row in such data imbalance. However, in the meantime, we find it very hard to obtain enough terrestrial data from the GEDI [4] repository due to the sparsity of raw data in many regions. For example, we originally intended to include the terrestrial Ambrym Volcano in our dataset. However, after inspecting its raw 3D point cloud (Fig. 11), we decided that the data was so sparse that it was impossible to conduct raster interpolation to it that was faithful to the groundtruth. We note that this is not an isolated case. Rather, we have faced such difficulties for many regions on the Earth that would have been included were not for data sparsity.

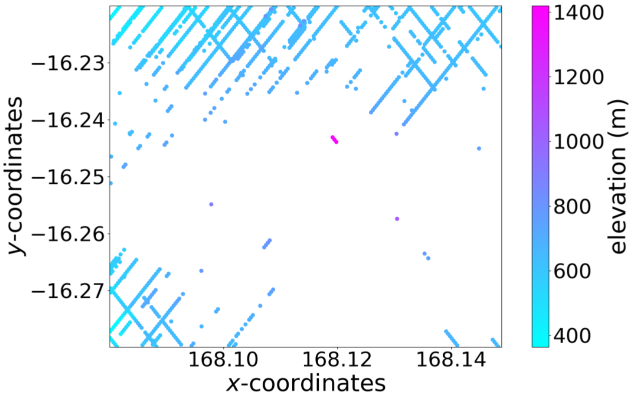


Fig. 11. The raw 3D point cloud of the terrestrial Ambrym Volcano from the GEDI [4] repository, which is so sparse that we were not able to conduct raster interpolation to it that was faithful to the groundtruth.

Moreover, even with the current dataset with relatively dense raw data, raster interpolation still has a significant impact on the CV results. To illustrate, the CDD of the CV scores of $PCA-2D-small$ with different IDW α settings is shown in Fig. 12. As is indicated, different interpolation configurations can lead to significantly different CV scores. This is also true for the majority of the remaining 11 feature templates. To address both this problem and the previously mentioned problem with downright invalid data due to sparsity issues, it is advisable to improve the raw data density in the GEDI [4] repository for certain regions on the Earth, thus making raster interpolation produce more accurate and robust results.

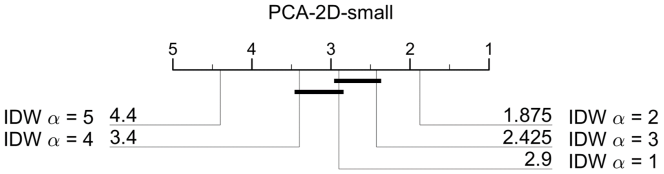


Fig. 12. Critical difference diagram of the cross validation κ scores of the feature template $PCA-2D-small$ with different IDW α settings. See Fig. 20 in the Appendix for diagrams of other feature templates.

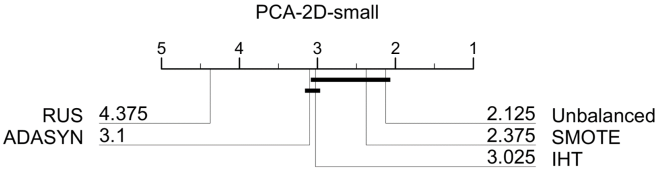


Fig. 13. Critical difference diagram of the cross validation κ scores of the feature template $PCA-2D-small$ with different data balancing methods. See Fig. 21 in the Appendix for diagrams of other feature templates.

3) The need for novel data balancing techniques: While denser raw data can help mitigate data imbalance, it is unlikely to fix the problem entirely. In many cases, data imbalance is almost inevitable, especially when we deal with supersized geo-objects such as Olympus Mons. Therefore, we still need explicit data balancing in CV. However, as was shown in the previous section, the classic data balancing methods we have considered so far have failed to address the aforementioned bias issue. Moreover, different data balancing methods have led to significantly different CV results, as is shown in Fig. 13. Based on these observations, we contend that there is a need for novel data balancing techniques for planetary LiDAR, especially one adaptive to our DSP strategy.

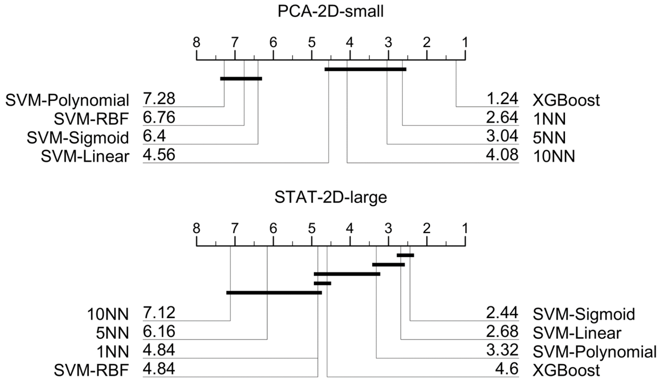


Fig. 14. Critical difference diagrams of the cross validation κ scores of the feature templates $PCA-2D-small$ and $STAT-2D-large$ with different classifiers. See Fig. 22 in the Appendix for diagrams of other feature templates.

4) The need for attention to classification decision boundaries: Last but not least, as is shown in Fig. 14, we have noticed that different classifiers can lead to significantly different CV results for a single feature template. In addition, different feature templates can correspond to different classifiers when the best results are achieved. It is likely that the decision boundaries between the target labels under different feature templates are different, and only the classifiers that can best detect these boundaries can lead to the best results. We believe that the characteristics of the decision boundaries may encode deep geoscientific knowledge that is worth looking into, and it is thus advisable for researchers to pay attention to the decision boundaries which can be obtained by feature visualization (see Section III-E).

C. Efficiency of Raw Feature Selection

We now turn to a key design highlight of our GPU-based FARMYARD pipeline: the Pseudo-One-Pass Sweep (POPS) framework for raw feature extraction, which promises much higher efficiency than its CPU-based counterpart[1]. We now validate this promise using both simulated and real data. For the rest of the section, the number of threads per GPU block is set to 256.

1) Efficiency on simulated data: We begin with experiments on simulated data which is randomly-generated rasters. First, we examine how our POPS framework compares against the following 3 rivals:

---

[1] Note that in for the rest of the section, the CPU-based implementation heavily relies on the NumPy package, whose vectorization capability usually makes CPU-based operations much faster than naïve implementations. This makes it fairer to our CPU-based rival when compared with our POPS framework.

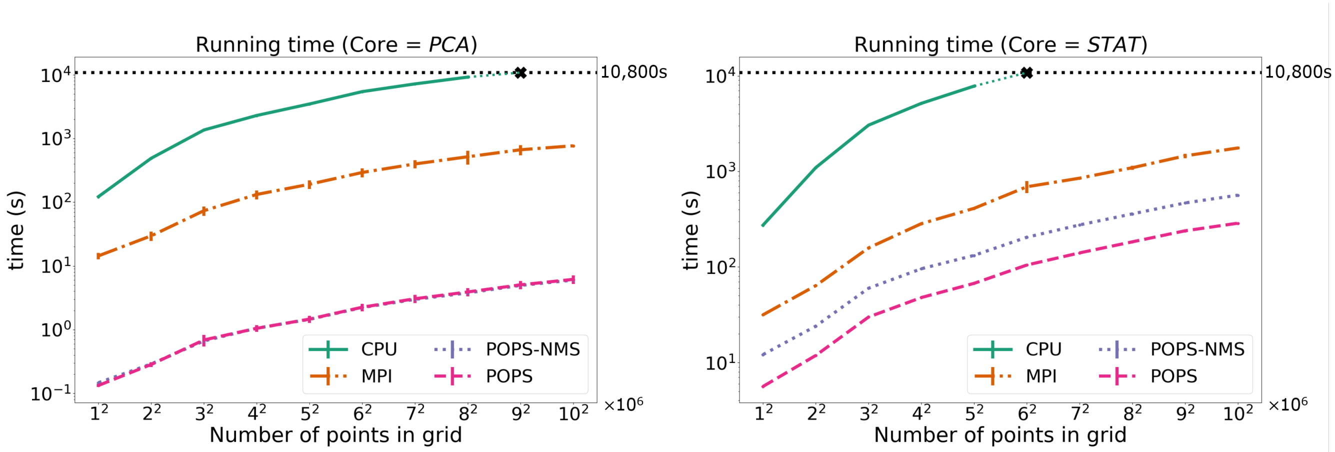


Fig. 15. Raw feature extraction time on simulated rasters with varying numbers of points. Note that the time axis is in log scale.

- **CPU:** the CPU-based (single-core) sequential framework;
- **MPI:** a CPU-based parallel framework with the Message Passing Interface (MPI) [15], which uses 20 cores*We use 20 cores, instead of all 24 cores, to avoid interfering with other users' jobs running on the same server.* with each core processing roughly the same number of examples;
- **POPS-NMS**: a variant of POPS without its ability to simultaneously process multiple feature scales[1],

For POS and the 3 rival frameworks in question, we conduct raw feature extraction using both PCA and $STAT$ on randomly-generated square rasters. These rasters have different numbers of points along each edge ranging from 1000 to 10000 with a step size of 1000[2]. On all rasters, we fix the example scale $exScale$ to 300 data points. As with the feature scales $fScales$, we set it as the union of 3 groups: $small$ where the scales are 5-10 points, $medium$ where the scales are 15-20 points, and $large$ where the scales are 25-30 points[3], thus we have a total of $3 \times 6 = 18$ feature scales. The feature stride-to-scale ratio $stsRatio$ is fixed to 0.5.

The results are shown in Fig. 15. Note that we have set a timeout threshold of 3h (10,800s) to avoid excessively long runs. From the results, we can draw the following conclusions.

- CPU is the slowest of all frameworks, running out of time for both PCA and $STAT$. This comes as no surprise as it exploits no parallelism at all.
- In most cases, MPI is just below 20x as fast as CPU. This is also unsurprising as we have used 20 cores for MPI, and the overhead time of MPI makes its actual acceleration a little short of its theoretical 20x speed boost.
- Both GPU-based frameworks, namely POPS and POPS-NMS, are significantly faster than their CPU-based rivals. For PCA, the GPU-based frameworks are 2-3 orders of magnitude faster than CPU, and 2 orders of magnitude faster than MPI; for $STAT$, the GPU-based frameworks are 1-2 orders of magnitude faster than CPU, and 2-7x as fast as MPI. The reason why the advantage of the GPU-based frameworks is smaller for $STAT$ is that they use a memory-efficient implementation with $O(nlgn)$ time complexity (see Section IV-C) for the core method, while the CPU-based frameworks use one with $O(n)$ complexity. Despite this, overall the GPU-based frameworks remain much faster than their CPU-based rivals. In particular, their advantage over MPI is impressive in that in order for the latter to catch up with the former, we would need to use a cluster of several (even dozens) of servers with the same specifications as the one we used in this work, as opposed to a single GPU for POPS and POPS-NMS. This highlights the efficiency benefit of exploiting the SIMT architecture of the GPU (see Section IV-A) for data-intensive tasks.
- While POPS has nearly identical performance as POPS-NMS on PCA, the former is about 2x as fast as the latter on $STAT$. The reason why there is no significant performance difference on PCA is likely that with an $O(n)$ implementation of the core method, both frameworks are so fast that the actual psuedo-one-pass sweep process shown in Fig. 5 accounts for very limited time compared to overhead operations such as communications between the CPU and the GPU. By contrast, on $STAT$, because the $O(nlgn)$ implementation of the core method makes the overall running time much longer, the performance benefit of the multi-scale processing capability of POPS begins to emerge. Moreover, as we will later show in our experiments on real data, when the volume of data grows larger, this performance benefit will also emerge for PCA. This benefit can be especially valuable for larger rasters and slower core methods such as $STAT$ as the absolute running time in such cases are longer, and a 2x speed boost can save plenty of time.

Our next simulated experiment concerns the impact of different $fScales$ settings on POPS with varying numbers of examples and number of points in each example. Specifically, we have randomly generated a raster with $30,000 \times 30,000 =$

---

[1] We disable simultaneous multi-scale processing for POPS-NMS using the _syncthreads() function (see Section IV-A)

[2] Note that these numbers are NOT the numbers of points in the rasters. In fact, the latter is the square of the former.

[3] Note that the 3 feature scale groups we use for simulated experiments are different from those used for real data in the previous section, despite the fact that they share the same naming.

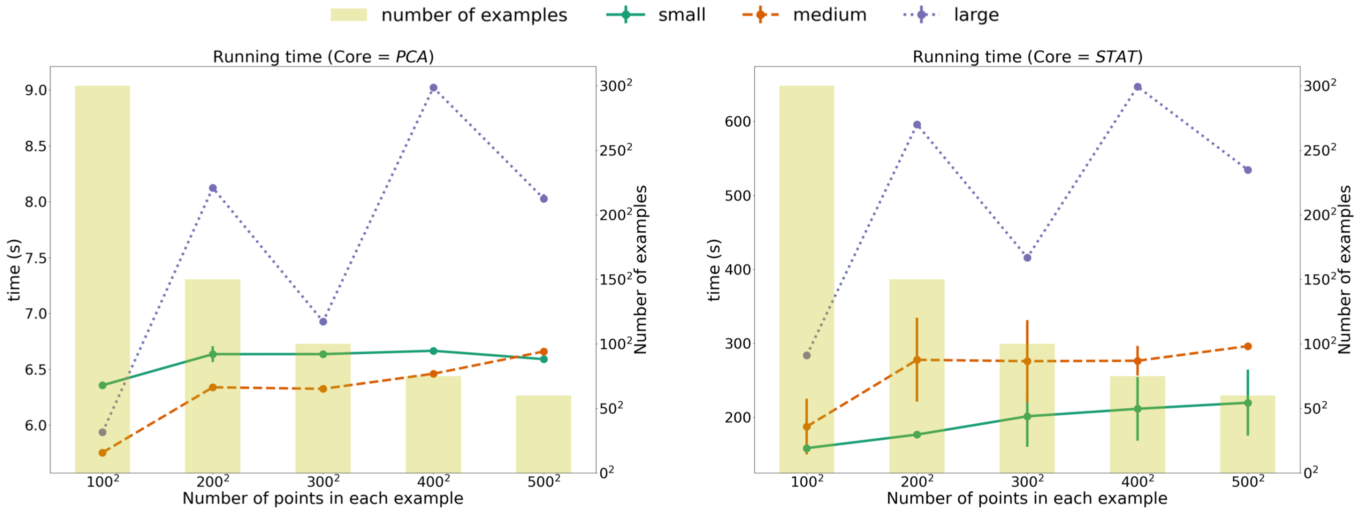


Fig. 16. Raw feature extraction time of POPS on a simulated raster of $30,000 \times 30,000$ with different example and feature scale settings.

9×10^8 points. To put into perspective how large this raster is, suppose it has a high resolution of 30m, then it covers an area of $(30,000 \times 30/1,000)^2 = 810,000\text{km}^2$. This is larger than the entire French territory (excluding Adélie Land in Antarctica), which is $638,474\text{km}^2$ including the overseas territories [43]. We set different values for $exScale$ within a range of $100 : 100 : 500$. Because the size of the raster is fixed, these different settings lead to different numbers of examples in the raster. For $fScales$, we consider the 3 settings of $small$, $medium$ and $large$ in our previous simulated experiment, except that this time we consider them separately, rather than use their union. With $stsRatio$ fixed to 0.5, we run POPS under these 3 settings and record the running times.

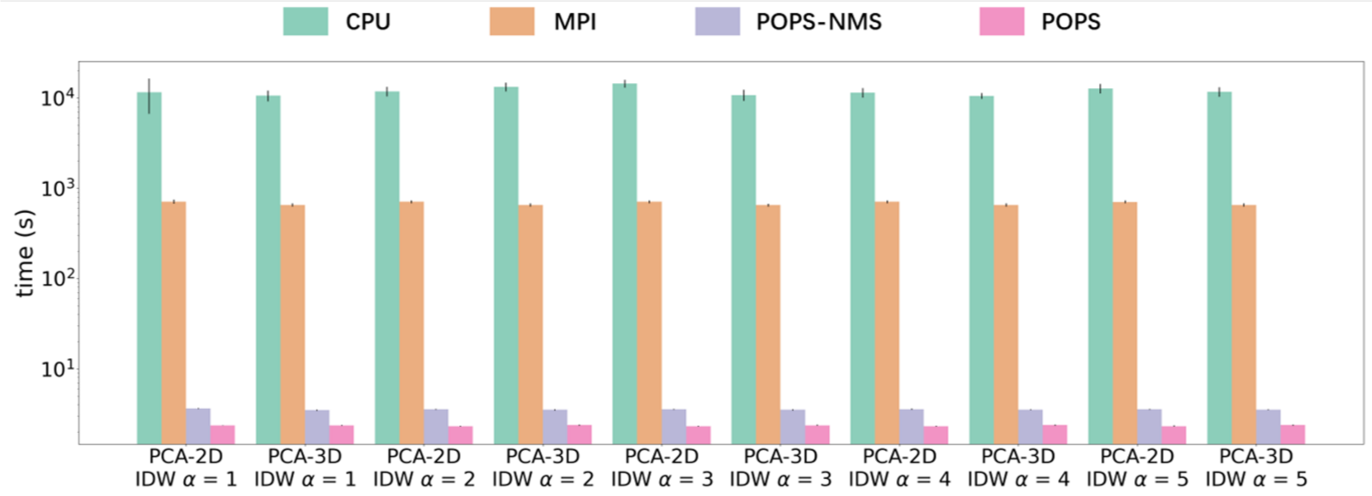
The results are shown in Fig. 16. First of all, POPS runs much slower for $STAT$ than PCA, which is likely due to the higher time complexity of the former. As with the $fScales$ settings, both $small$ and $medium$ leads to relatively short running time, while $large$ has significantly longer time. This reveals a weakness of our POPS framework: it only allows one thread per patch, no matter how large the patch is, thus it is relatively inefficient at processing large feature scales. In particular, despite its optimizations for multi-scale scenarios, this yields limited gains when *all* the scales are large. That being said, the longest running time is a little over 600s (10min), which is still very impressive given the massiveness of the raster. Another interesting phenomenon is the zig-zag pattern of the running time curve for $large$. This is likely to be a combined effect of 2 competing factors: On the one hand, as the number of points in each example increases, so does the number of patches per example and thus the per-example processing time. On the other hand, the number of examples drops as the size of individual examples grows larger, which decreases the overall running time.

All in all, the aforementioned experiments strongly indicate that our POPS framework can easily defeat its CPU-based counterparts and can highly efficiently handle very large rasters. Moreover, its support for simultaneous multi-scale processing is indeed effective in boosting its efficiency, especially in cases where the absolute running time is longer.

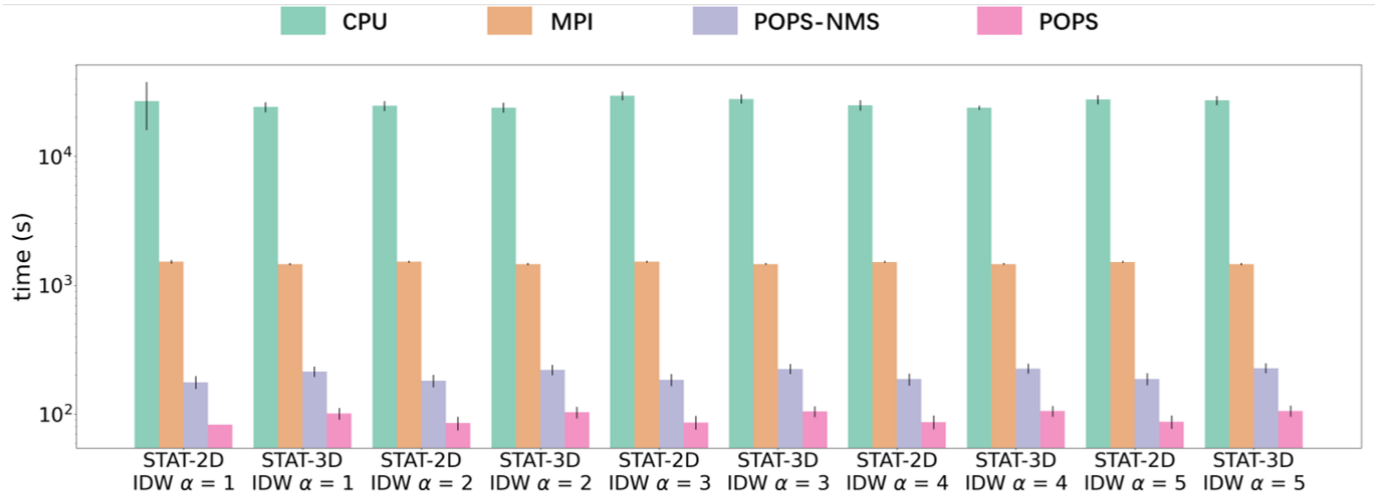
2) Efficiency on real data: We now further validate the efficiency of POPS on real data. Specifically, we report the running time of POPS, POPS-NMS and their CPU-based counterpart when applied to extracting raw features from the dataset used in Section V-A, where we have also specified the parameters for feature extraction. The results are shown in Fig. 17, which are mostly in line with those on simulated data (Fig. 15). Specifically, for PCA, POPS is more than 3 orders of magnitude faster than CPU and 2 orders of magnitude faster than MPI; for $STAT$ where POPS has higher per-patch time complexity, it is still over 2 orders of magnitude faster than CPU and 1 order of magnitude faster than MPI. What is different from the case with the simulated experiment in Fig. 15 is that for real data, not only is POPS more than 2x as fast as POPS-NMS for $STAT$, it is also about 1.5x as fast as POPS-NMS, as opposed to having nearly identical performance as the latter on simulated data. This is likely because the real data volume is larger than that for the simulated experiment, which makes the actual pseudo-one-pass sweep process take most of the running time compared to overhead operations for real data.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a generic and (nearly) entirely GPU-based pipeline for Feature Discovery From Planetary LiDAR Data (FARMYARD) with 3 design highlights: a Pseudo-One-Pass Sweep (POPS) based framework for efficient raw feature extraction of massive raster data, a two-level ROI division scheme for multi-scale feature extraction of local regions, and a Domain-Shifted Partition (DSP) strategy for robust feature evaluation in the presence of interfering factors. We validated the utility, effectiveness and efficiency of our contributions through extensive experiments. Based on our results on real-world data, we have also made suggestions to future researches on planetary LiDAR such as developing new core methods for feature extraction (especially improving upon



(a) PCA



(b) STAT

Fig. 17. Raw feature extraction time on the real dataset used in Section V-A. Note that the time axis is in log scale.

PCA), enhancing raw data density, improving data balance, and paying attention to classification decision boundaries for different feature extraction methods.

For future work, we mainly focus on the following 3 aspects.

1) Improving efficiency on large patches: As was demonstrated in Fig. 16, the efficiency gain of our POPS-based feature extraction method decreases for larger patches, due to its limitation that a patch can only be processed by a single thread. In response, we plan to extend our current method to allow for parallel processing of a patch using multiple threads, thus boosting our efficiency on larger patches. This may only be possible for some core methods. If so, we plan to propose generic guidelines that can help the user determine whether their core method can be executed using multiple threads, and if this is the case, the optimal number of threads per patch to use for maximum efficiency.

2) Incorporating deep learning (DL): Throughout this paper, we have limited our discussions to traditional hand-crafted features, rather than deep features learned by neural networks. This is primarily out of concerns for feature interpretability, as the black-box nature of data-driven neural networks can prohibit domain experts from interpreting learned deep features and thus uncovering equally deep knowledge. However, in view of the effectiveness of DL, we plan to incorporate it into our current pipeline, so that the interpretability of handcrafted features and the efficacy of DL can complement each other. One possible way of doing this is to first discover potentially interesting handcrafted features using our current pipeline, and then apply a neural network to fine-tune these features, e.g. weighting them using the attention mechanism [44]. Moreover, recent advances in explainable DL [45] may also shed light on humanly interpretable knowledge embedded in deep models.

3) Extending to new topics on planetary topography: The ultimate goal of this work is to provide a new methodology for comparative studies of planetary topography that can lead to future geoscientific discoveries. Thanks to the generic and GPU-based nature of our FARMYARD pipeline, we can now compare multiple geo-objects with easiness, accuracy and efficiency never possible before, which can potentially usher in a new era of the study of the "text" provided by topography. Apart from continuing with our PARKER project (see Section V-A) which seeks to find topographical life signatures on the Earth, with FARMYARD, we can extend our research to other interesting topics. For example, we can study the influence of life on planetary geomorphology by comparing biotic and abiotic controlled landscapes (on the same planet); for similar landscapes at different geolocations, we can study the influence of the different forming mechanisms of their topography. We can even ask questions as yet unthought-of, because once new ways of seeing become available, new questions emerge.

Appendix: Additional Experimental Results

We provide additional experimental results that were previously omitted for brevity, which are shown in Fig. 18-22.

Acknowledgment

This work is supported by the Data Intelligence Institute of Paris (diiP), and IdEx Université Paris Cité (ANR-18-IDEX-0001). Numerical computations were partly performed on the S-CAPAD/DANTE platform, IPGP, France. We gratefully acknowledge the support of NVIDIA Corporation with the donation of GPU used for this research.

References

[1] D. E. Smith, M. T. Zuber, S. C. Solomon, R. J. Phillips, J. W. Head, J. B. Garvin, W. B. Banerdt, D. O. Muhleman, G. H. Pettengill, G. A. Neumann *et al.*, "The global topography of mars and implications for surface evolution," *Science*, vol. 284, no. 5419, pp. 1495–1503, 1999.

[2] D. E. Smith, M. T. Zuber, G. A. Neumann, F. G. Lemoine, E. Mazarico, M. H. Torrence, J. F. McGarry, D. D. Rowlands, J. W. Head, T. H. Duxbury *et al.*, "Initial observations from the lunar orbiter laser altimeter (lola)," *Geophysical Research Letters*, vol. 37, no. 18, 2010.

[3] B. W. Stiles, S. Hensley, Y. Gim, D. M. Bates, R. L. Kirk, A. Hayes, J. Radebaugh, R. D. Lorenz, K. L. Mitchell, P. S. Callahan *et al.*, "Determining titan surface topography from cassini sar data," *Icarus*, vol. 202, no. 2, pp. 584–598, 2009.

[4] R. Dubayah, M. Hofton, J. Blair, J. Armston, H. Tang, and S. Luthcke, "Gedi l2a elevation and height metrics data global footprint level v002," *NASA EOSDIS Land Processes DAAC*, 2021.

[5] W. E. Dietrich and J. T. Perron, "The search for a topographic signature of life," *Nature*, vol. 439, no. 7075, pp. 411–418, 2006.

[6] H. J. Melosh, *Planetary Surface Processes*, ser. Cambridge Planetary Science. Cambridge University Press, 2011.

[7] M. A. Wieczorek, "Gravity and topography of the terrestrial planets," *Treatise on Geophysics*, vol. 10, pp. 165–206, 2007.

[8] R. D. Lorenz, B. W. Stiles, O. Aharonson, A. Lucas, A. G. Hayes, R. L. Kirk, H. A. Zebker, E. P. Turtle, C. D. Neish, E. R. Stofan *et al.*, "A global topographic map of titan," *Icarus*, vol. 225, no. 1, pp. 367–377, 2013.

[9] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[10] D. G. Krige, "A statistical approach to some basic mine valuation problems on the witwatersrand," *Journal of the Southern African Institute of Mining and Metallurgy*, vol. 52, no. 6, pp. 119–139, 1951.

[11] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in *Proceedings of the 1968 23rd ACM National Conference*, 1968, pp. 517–524.

[12] N. Brodu and D. Lague, "3d terrestrial lidar data classification of complex natural scenes using a multi-scale dimensionality criterion: Applications in geomorphology," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 68, pp. 121–134, 2012.

[13] E. Yudkowsky *et al.*, "Artificial intelligence as a positive and negative factor in global risk," *Global catastrophic risks*, vol. 1, no. 303, p. 184, 2008.

[14] NVIDIA Corporation, "Cuda c++ programming guide," 2021.

[15] W. Gropp, W. D. Gropp, E. Lusk, A. Skjellum, and A. D. F. E. E. Lusk, *Using MPI: portable parallel programming with the message-passing interface*. MIT press, 1999, vol. 1.

[16] C. S. Balzotti and G. P. Asner, "Episodic canopy structural transformations and biological invasion in a hawaiian forest," *Frontiers in plant science*, vol. 8, p. 1256, 2017.

[17] C. Mallet and F. Bretar, "Full-waveform topographic lidar: State-of-the-art," *ISPRS Journal of photogrammetry and remote sensing*, vol. 64, no. 1, pp. 1–16, 2009.

[18] E. Mazarico, D. Rowlands, G. Neumann, D. Smith, M. Torrence, F. Lemoine, and M. Zuber, "Orbit determination of the lunar reconnaissance orbiter," *Journal of Geodesy*, vol. 86, no. 3, pp. 193–207, 2012.

[19] G. Hughes, "On the mean accuracy of statistical pattern recognizers," *IEEE transactions on information theory*, vol. 14, no. 1, pp. 55–63, 1968.

[20] P. De Chazal, M. O'Dwyer, and R. B. Reilly, "Automatic classification of heartbeats using ecg morphology and heartbeat interval features," *IEEE transactions on biomedical engineering*, vol. 51, no. 7, pp. 1196–1206, 2004.

[21] G. Wilson and D. J. Cook, "A survey of unsupervised deep domain adaptation," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 11, no. 5, pp. 1–46, 2020.

[22] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[23] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[24] H. He, Y. Bai, E. A. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*. IEEE, 2008, pp. 1322–1328.

[25] M. R. Smith, T. Martinez, and C. Giraud-Carrier, "An instance level analysis of data complexity," *Machine learning*, vol. 95, no. 2, pp. 225–256, 2014.

[26] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 559–563, 2017.

[27] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[28] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data mining and knowledge discovery*, vol. 31, no. 3, pp. 606–660, 2017.

[29] S. Liang, Y. Zhang, and J. Ma, "Active model selection for positive unlabeled time series classification," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 361–372.

[30] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[31] T. Cheng, "Accelerating universal kriging interpolation algorithm using cuda-enabled gpu," *Computers & geosciences*, vol. 54, pp. 178–183, 2013.

[32] G. Mei, N. Xu, and L. Xu, "Improving gpu-accelerated adaptive idw interpolation algorithm using fast knn search," *SpringerPlus*, vol. 5, no. 1, pp. 1–22, 2016.

[33] Y. Zhang, X. Zheng, Z. Wang, G. Ai, and Q. Huang, "Implementation of a parallel gpu-based space-time kriging framework," *ISPRS International Journal of Geo-Information*, vol. 7, no. 5, p. 193, 2018.

[34] J. I. Munro and V. Raman, "Selection from read-only memory and sorting with minimum data movement," *Theoretical Computer Science*, vol. 165, no. 2, pp. 311–323, 1996.

[35] M. Harris *et al.*, "Optimizing parallel reduction in cuda," *Nvidia developer technology*, vol. 2, no. 4, p. 70, 2007.

[36] N. Bell and J. Hoberock, "Thrust: A productivity-oriented library for cuda," in *GPU computing gems Jade edition*. Elsevier, 2012, pp. 359–371.

[37] R. Nishino and S. H. C. Loomis, "Cupy: A numpy-compatible library for nvidia gpu calculations," *31st confernce on neural information processing systems*, vol. 151, 2017.

[38] S. Raschka, J. Patterson, and C. Nolet, "Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence," *arXiv preprint arXiv:2002.04803*, 2020.

[39] Z. Wen, J. Shi, Q. Li, B. He, and J. Chen, "Thundersvm: A fast svm library on gpus and cpus," *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 797–801, 2018.

[40] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[41] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[42] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih, "Pycuda and pyopencl: A scripting-based approach to gpu run-time code generation," *Parallel Computing*, vol. 38, no. 3, pp. 157–174, 2012.

[43] Eurostat (European Statistical Office). European Commission. Accessed Nov. 28, 2022. [Online]. Available: https://ec.europa.eu/eurostat/

[44] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[45] E. Tjoa and C. Guan, "A survey on explainable artificial intelligence (xai): Toward medical xai," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 11, pp. 4793–4813, 2020.

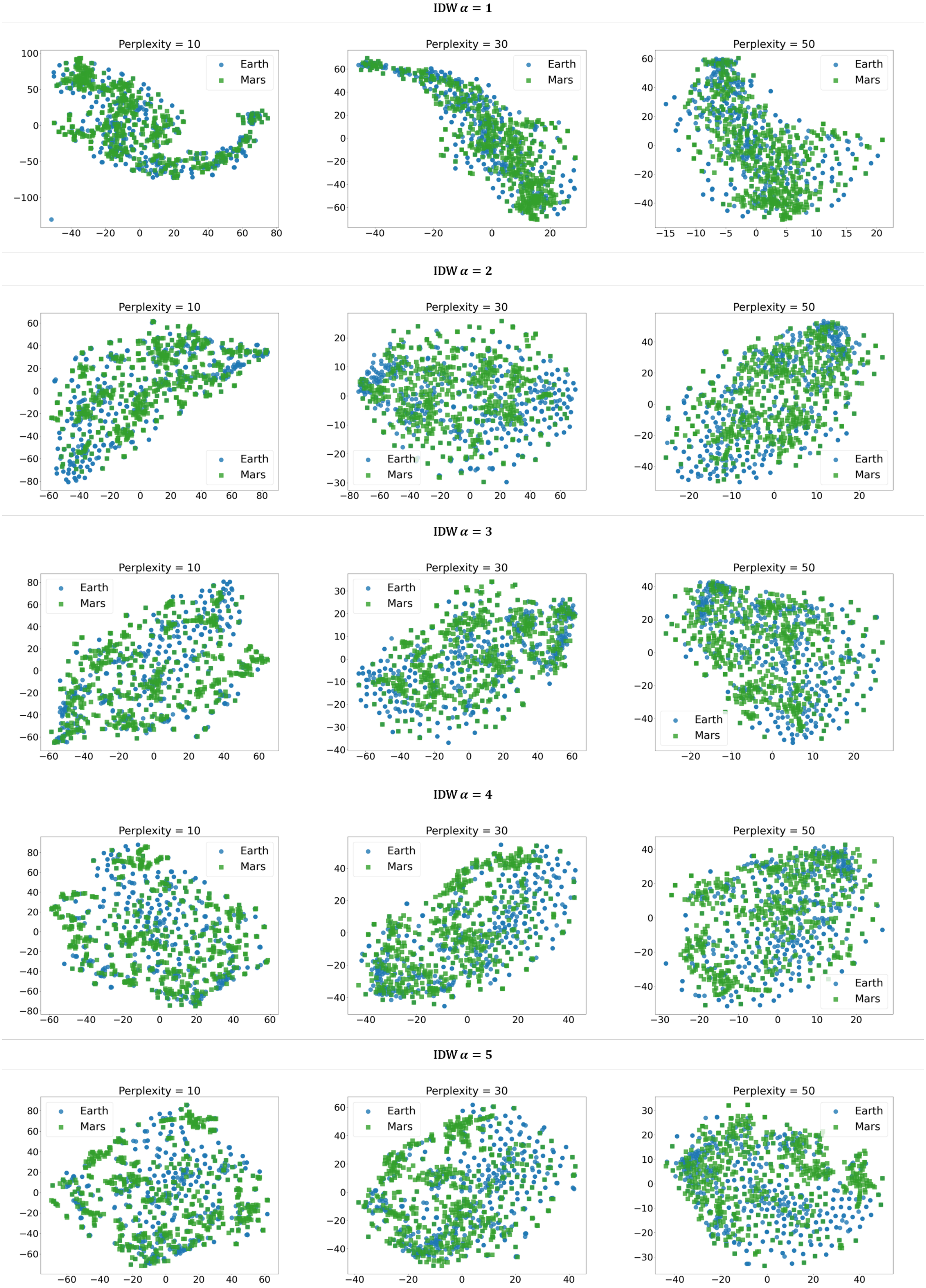
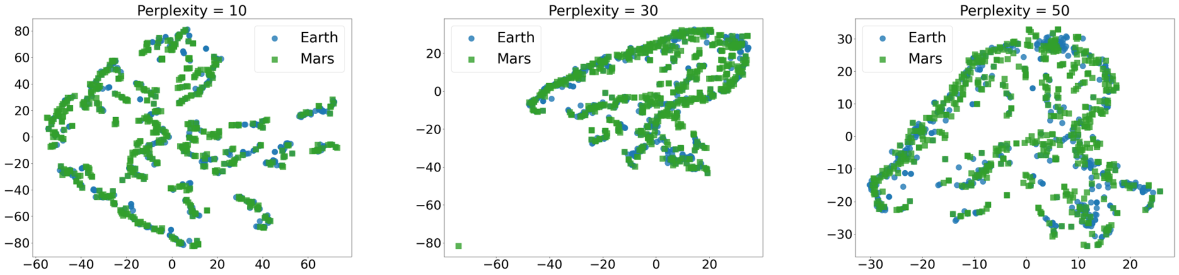
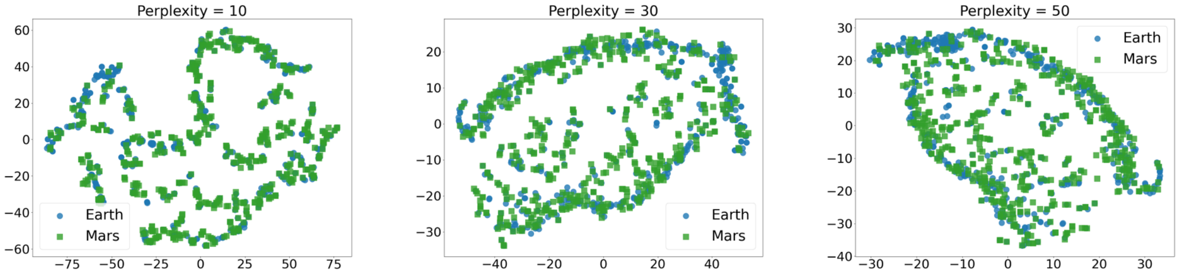


Fig. 18. Visualization of raw features under the feature template $PCA-2D-small$ with IDW α set to 1, 2, 3, 4, 5.

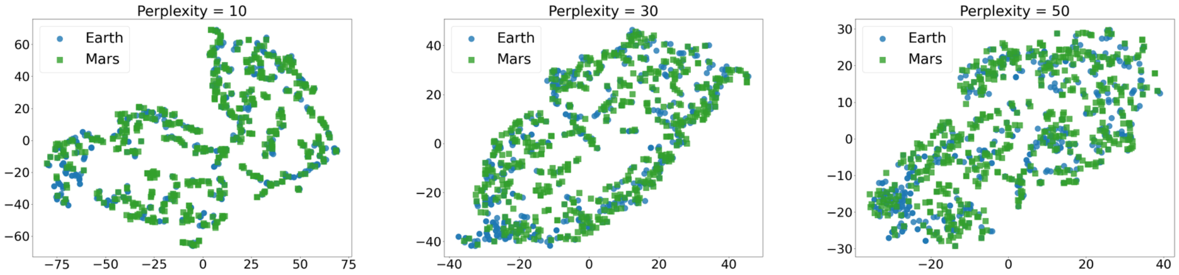
IDW α = 1



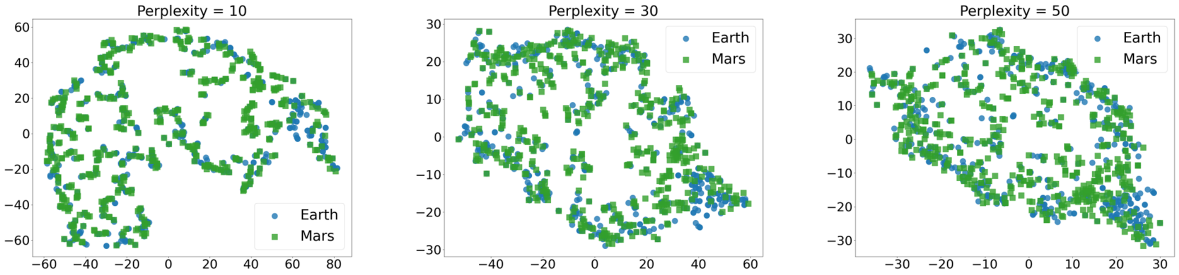
IDW α = 2



IDW α = 3



IDW α = 4



IDW α = 5

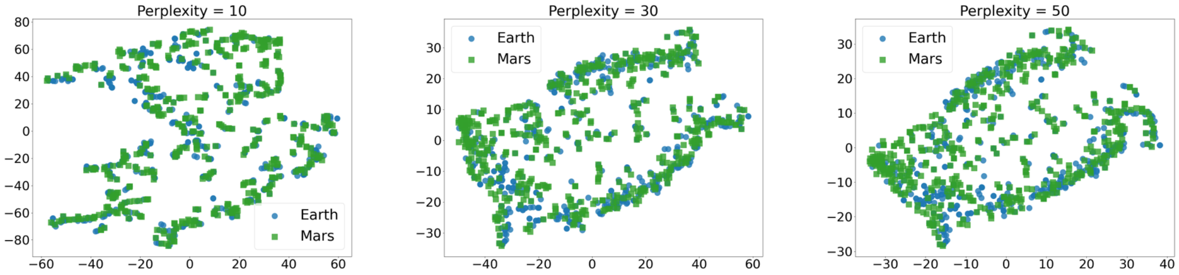


Fig. 19. Visualization of aggregated features under the feature template $PCA-2D-small$ with IDW α set to 1, 2, 3, 4, 5.

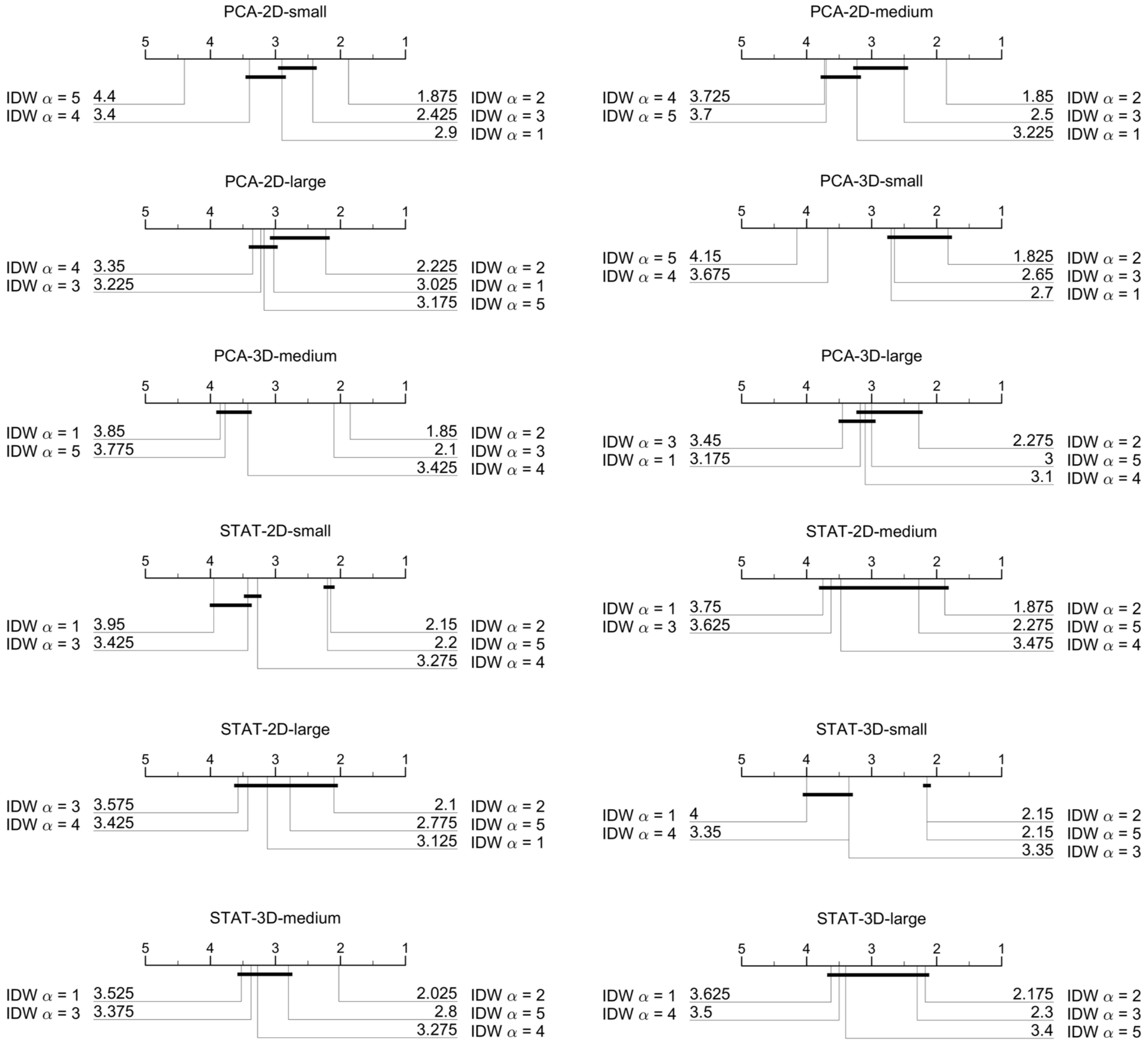


Fig. 20. Critical difference diagrams of the cross validation κ scores of all 12 feature templates with different IDW α settings.

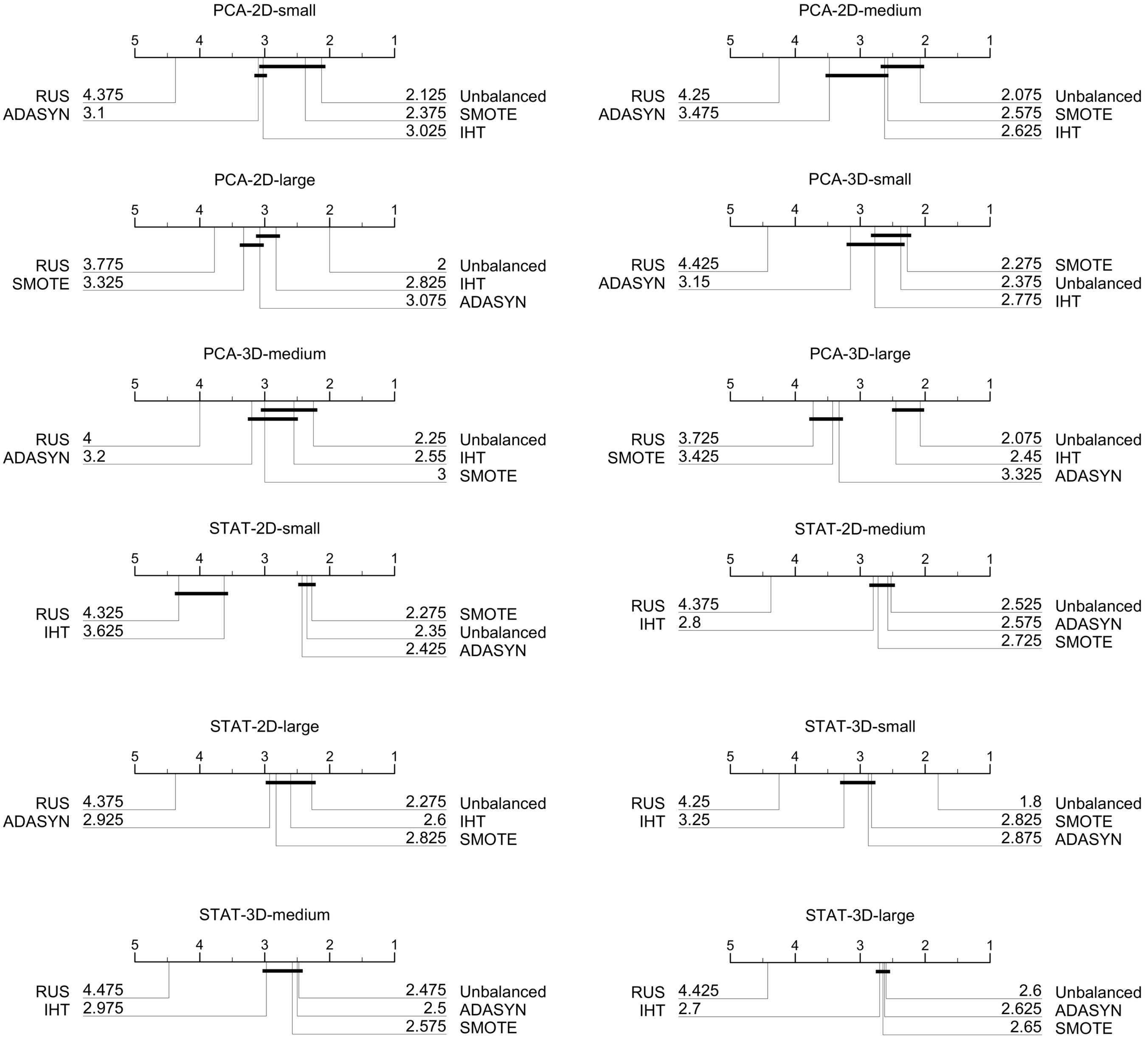


Fig. 21. Critical difference diagrams of the cross validation κ scores of all 12 feature templates with different data balancing methods.

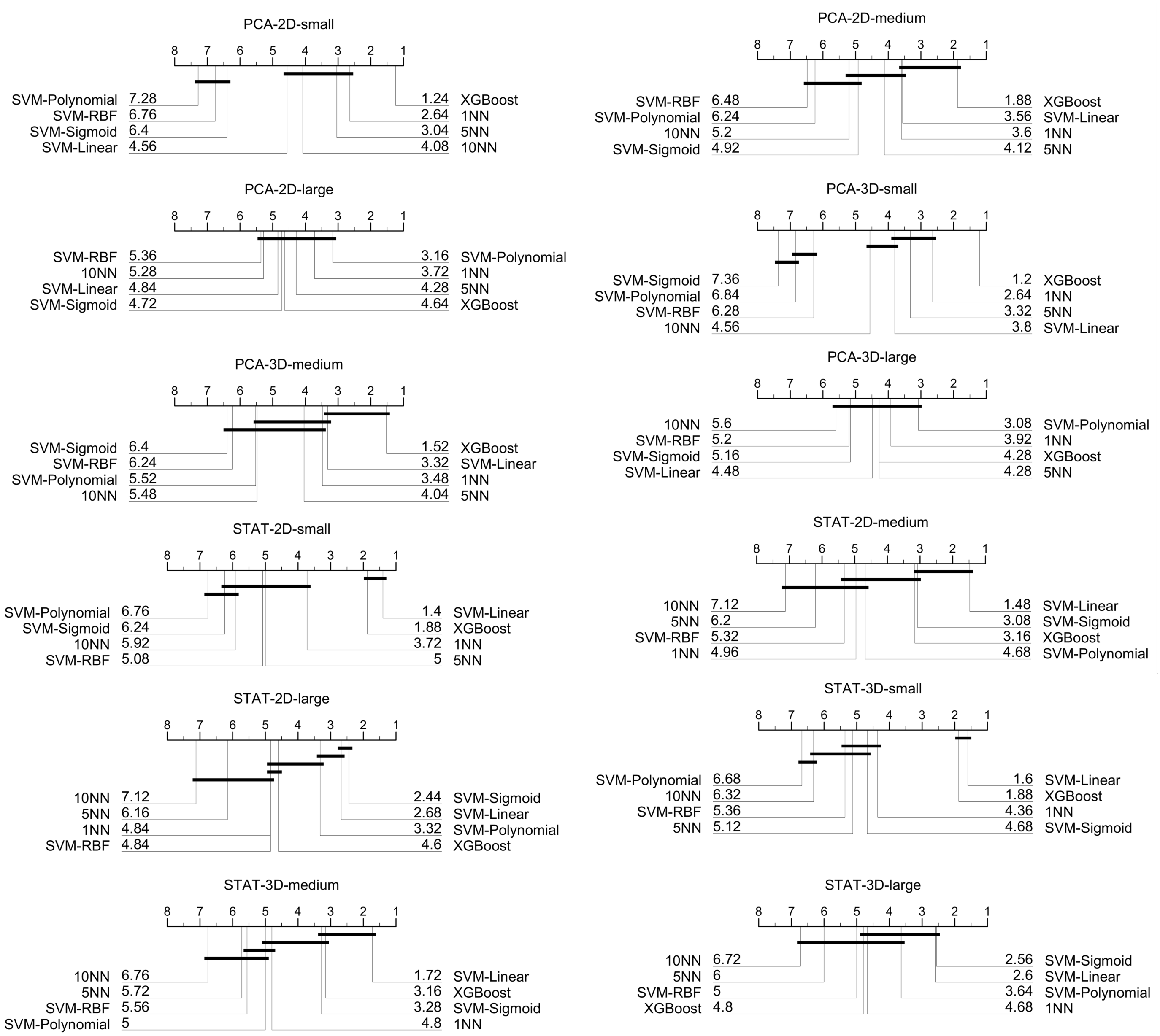


Fig. 22. Critical difference diagrams of the cross validation κ scores of all 12 feature templates with different classifiers.