Portland State University

Transportation Research and Education Center

Summer 2025 Graduate Research Assistantship

# Measuring Distributional Similarity via Maximum Mean Discrepancy

*Author:*
Elijah Whitham-Powell,
M.S. Computer Science
M.S. Applied Statistics Candidate
Portland State University

*Supervisor:*
Tammy Lee, Ph.D.

09/15/2025


# Contents

# 1 Executive Summary

Understanding the key differences between data collection approaches and their resulting datasets is crucial for transportation planning and traffic management. This project aims to compare travel-time data from two prominent sources, PORTAL and INRIX, using the Maximum Mean Discrepancy (MMD) statistic. Each dataset records travel-time on highway segments. However, they differ in approaches to collecting those recordings, where fixed point highway sensors are used in one dataset (PORTAL) and OEM probe data from moving vehicles are used in the other (INRIX). By analyzing the travel-time distributions from these datasets, we can identify significant differences and similarities that may impact their use in various applications.

**The Data:** Data was collected from two sources: PORTAL–public data managed by the Transportation Research and Education Center (TREC) at Portland State University (PSU)–and INRIX–a commercial provider. The PORTAL data is aggregated from sensors maintained by Oregon Department of Transportation (ODOT) and Washington State Department of Transportation (WSDOT). The INRIX data is collected from GPS-enabled vehicles, mobile devices, and other third-party sensors.

**The Analysis:** An unbiased estimator of MMD with the Radial Basis Function (RBF) kernel was applied to various views of the travel-time data in order to ask and attempt to answer several questions about how the distributions may or may not differ. The focus of analysis was constrained to a subset of 15-minute interval travel-time readings from 2019 through 2024 on I-5, I-205, and SR-14 in the Portland, Oregon - Vancouver, Washington Metropolitan Area.

**Missing Intervals:** The datasets differed substantially in data completeness, with PORTAL containing notable gaps at several stations and INRIX having comparatively few missing intervals. To ensure that these differences did not disproportionately influence the distributional comparisons, three complementary strategies were used to handle missing readings: standardization with zero-filling, masking during computation, and a combined standardized-masking approach. These methods allowed the analysis to separate effects due to missingness from genuine distributional differences.

**Key Findings:** Results indicate that the travel-time readings from PORTAL and INRIX can be considered to come from a different distribution as measured by the MMD statistic. This suggests that the two data sources may capture different aspects of traffic conditions–potentially due to differences in data collection methods and sensor types. However, the experiments also revealed a trend towards increasing similarity between the datasets over time, hinting at possible improvements in data collection, sensor coverage, or processing techniques.

**Code for this project can be found in the following GitHub repo[8]:**

- `https://github.com/whitham-powell/TREC-PORTALvsINRIX-MMD`

## 2 Introduction

We aim to measure similarity or dissimilarity between the two data collection approaches and the underlying distributions generated by each. This can be difficult without a parametric assumption about the distributions. Kernel methods and the Maximum Mean Discrepancy (MMD) statistic provide a non-parametric approach to measuring distributional similarity based on samples drawn from each distribution. Scaling that MMD computation to number of standard deviations from the expected MMD results in an apples to apples comparison across different segments, time frames, or groupings of the data.

### 2.1 Previous Work with PORTAL and INRIX

Previous work was done by the author in collaboration with Transportation Research and Education Center (TREC) at PSU as part of a capstone course [9][7]. The previous project focused on processing and spatially joining the two datasets so that meaningful comparisons could be made. This project improved upon the previous spatial joining and continued with the analysis of travel-time data using a statistical approach based on kernel methods.

- **PORTAL**–maintained by state transportation agencies–derives its travel-time metrics primarily from roadside sensors (e.g., loop detectors) that record vehicle counts and speeds as cars pass fixed locations.
- **INRIX**–a commercial provider–collects data from GPS-enabled vehicles, mobile devices, and other third-party sensors. By tapping into moving "probe" data rather than fixed detectors, INRIX can often capture a broader picture of travel conditions.

### 2.2 Objectives

The primary goal was to implement and compute an unbiased $\widehat{\text{MMD}}^2$ estimator, using the Radial Basis Function (RBF) kernel with a computed median heuristic hyperparameter $\gamma$. This was done for an entire year's worth 15-minute interval travel-time readings and used in permutations tests to determine if the smallness or largeness of the computed $\widehat{\text{MMD}}^2$ estimator between two distributions is statistically significant. This pattern was repeated for several views of the same raw data.

### 2.3 Tools

Python 3.12 and the following libraries:

- `PyTorch`: For tensor computations and GPU acceleration.
- `PyKeOps`[1]: For efficient computation of large kernel matrices without materializing them in memory.
- `Pandas`: For data manipulation and analysis.
- `GeoPandas`: For spatial data joining and manipulation.
- `NumPy`: For numerical computations on arrays and matrices not handled by Torch.
- `Shapely`: To convert the Well Known Binary (WKB) format to a more usable format for spatial joins with the INRIX shape data.

# 3 Background

**Why MMD?** Maximum Mean Discrepancy (MMD) gives a non-parametric, kernel-based measure of distributional difference that avoids density estimation and works well in high dimensions [2, 5]. With a *characteristic* kernel (e.g., RBF), MMD = 0 if and only if the distributions are identical [6]. MMD has also been applied in related domains. For example, Li et al. demonstrated the use of MMD as a distribution-alignment loss for transfer learning in short-term traffic prediction tasks [4].

## 3.1 Kernel Mean Embedding (KME)

Given a kernel function $k$, defined on a set of data $\mathcal{X}$ with a corresponding feature space–Reproducing Kernel Hilbert Space (RKHS)–$\mathcal{H}$, the mean embedding of a distribution $\mathbb{P}$ on $\mathcal{X}$ is the function in $\mathcal{H}$ defined as,

$$\mu_{\mathbb{P}}(y) = \mathbb{E}_{X\sim\mathbb{P}}\left[k(X, y)\right], \quad y \in \mathcal{X}$$

which can then be estimated using the empirical mean of $N$ samples from $\mathbb{P}$,

$$\hat{\mu}_{\mathbb{P}}(y) = \frac{1}{N}\sum_{i=1}^{N} k(X_i, y), \quad X_i \overset{iid}{\sim} \mathbb{P}.$$

This allows us to map a distribution $\mathbb{P}$ to a single point $\mu_{\mathbb{P}}$ in the RKHS $\mathcal{H}$ as the KME that captures the essential features or "fingerprint" of the distribution $\mathbb{P}$.

## 3.2 Maximum Mean Discrepancy (MMD)

MMD is a statistical measure used to quantify the difference between two probability distributions $\mathbb{P}$ and $\mathbb{Q}$. It leverages the concept of KMEs to represent each distribution as a point in an RKHS. The MMD is defined as the distance between these two points in the RKHS, given by:

$$\begin{aligned}\mathrm{MMD}^2 &= \|\mu_{\mathbb{P}} - \mu_{\mathbb{Q}}\|^2_{\mathcal{H}} \\ &= \underbrace{\mathbb{E}_{X,X'\sim\mathbb{P}}\left[k(X, X')\right]}_{(i)} + \underbrace{\mathbb{E}_{Y,Y'\sim\mathbb{Q}}\left[k(Y, Y')\right]}_{(i)} - \underbrace{2\mathbb{E}_{X\sim\mathbb{P},Y\sim\mathbb{Q}}\left[k(X, Y)\right]}_{(ii)}\end{aligned}$$

where (i) are the expectations of the kernel function over pairs of samples from the same distribution, and (ii) is the expectation over pairs of samples from different distributions.

**Given samples, $\{X_1, \ldots, X_n\} \sim \mathbb{P}$ and $\{Y_1, \ldots, Y_m\} \sim \mathbb{Q}$ and a kernel function $k(\cdot, \cdot)$, the unbiased empirical MMD is computed as:**

$$\widehat{\mathrm{MMD}^2}(\mathbb{P}, \mathbb{Q}) = \frac{1}{n(n-1)}\sum_{i\neq j} k(X_i, X_j) + \frac{1}{m(m-1)}\sum_{i\neq j} k(Y_i, Y_j) - \frac{2}{nm}\sum_{i,j} k(X_i, Y_j)$$

Using the unbiased estimator, we can compute the MMD between two distributions finding the distance between their KME fingerprints. Generally speaking, the smaller the MMD, the greater the similarity between the two distributions, the larger, the more dissimilar. The further away from zero, the more evidence there is that the two distributions are different or come from different random processes.

### 3.3 Kernel Choice

The Radial Basis Function (RBF) kernel, also known as the Gaussian kernel, was chosen due to it being a *characteristic kernel*[1], similar in structure to commonly used distance metrics, and its minimal hyperparameter tuning requirements. MMD is not limited to this kernel. Any kernel could be used as long as it is symmetric and positive-semi-definite[2].

**The RBF kernel has two equivalent forms:**

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right) \equiv \exp\left(-\gamma\|x - y\|^2\right), \quad \gamma = \frac{1}{2\sigma^2}$$

where $\sigma$ or $\gamma$ is a parameter that controls the width of the kernel. The $\gamma$ specified version is used in the implementation of the RBF kernel in the `scikit-learn` Python library and this project. This was done in this project to ensure expected computations early in development matched those of using `scikit-learn`.

## 4 Methodology

### 4.1 Overview of Approach

We compare PORTAL vs. INRIX using $\widehat{\text{MMD}}^2$ with an RBF kernel across three complementary data views (Sec. 6): (1) whole-system 1D readings, (2) entity-centric full-year vectors, and (3) time-centric system snapshots. Each view is evaluated under a consistent preprocessing scheme (Sec. 5.3) and missing-data handling strategy[3], followed by a permutation test to assess significance.

### 4.2 Choosing the Kernel Parameter $\gamma$

The choice of the kernel parameter $\gamma$ is crucial as it influences the sensitivity of the RBF kernel to differences between data points. A small $\gamma$ value leads to a wide kernel, making the MMD less sensitive to small differences between distributions, while a large $\gamma$ results in a narrow kernel, increasing sensitivity but potentially leading to overfitting to noise in the data.

**Median Heuristic for $\gamma$:** This heuristic sets $\gamma$ based on the median of the pairwise squared distances between random samples from both distributions, providing a balance between sensitivity and robustness. Specifically, $\gamma$ is set as follows:

$$\gamma = \frac{1}{2 \cdot \underbrace{\text{median}(\{\|X_i - X_j\|^2, \|X_i - Y_j\|^2, \|Y_i - Y_j\|^2\})}_{\sigma^2}}$$

where $X_i, X_j$ are samples from distribution $\mathbb{P}$ and $Y_i, Y_j$ are samples from distribution $\mathbb{Q}$. The median of these squared distances becomes the $\sigma^2$ used in $\gamma = \frac{1}{2\sigma^2}$. This approach helps to ensure that the

[1] Which guarantees that the MMD equals zero if and only if the two distributions are identical[6]

[2] Common choices include the linear kernel, polynomial kernels, Laplace kernel, and Matérn kernels.

[3] The strategies were only applied to views 2 and 3. View 2 required a strategy to ensure consistent and equal length time features. View 3 required the missing-data strategy to make comparison with view 2 viable.

kernel captures the relevant scale of the data differences without being overly sensitive to outliers or noise[4].

### 4.3 Computing the Unbiased MMD Estimator

The estimator was implemented using PyTorch and PyKeOps[1] with torch bindings and can be found in the Appendix (PyKeOps version in Appendix C.2). The implementation was designed to be efficient and scalable, leveraging the power of PyTorch and PyKeOps to handle large datasets and high-dimensional data on both CPU and GPUs without memory overflows.

### 4.4 Permutation Tests for Statistical Significance

To determine if the computed $\widehat{\text{MMD}^2}$ values were statistically significant, permutation tests were used to create a null distribution of $\widehat{\text{MMD}^2}$ values. The null hypothesis for each permutation assumes the two distributions are the same is created as follows:

1. Combine the samples from both distributions into a single dataset.
2. Randomly shuffle the combined dataset and split it into two new datasets of the same sizes as the original datasets.
3. Compute the unbiased $\widehat{\text{MMD}^2}$ for the two new datasets.
4. Repeat Step 2 and Step 3 for 500 permutations[5] to create a distribution of $\widehat{\text{MMD}^2}$ values under the null hypothesis.

P-values are omitted due to every resulting p-value being equal and effectively zero ($\frac{1}{500}$) for each model. The z-scores are reported instead (Figure 4) to show the difference in number of standard deviations the $\widehat{\text{MMD}^2}$ value is from the mean of the null distribution.

### 4.5 Handling Missing Intervals (applies to Views 2 and 3)

Given that there were missing intervals, 3 different strategies were employed to handle the missing data. The goal of these strategies was to isolate missingness and reduce its impact as the only difference between the two distributions[6].

1. **"Z-scaled and 0-fill"**: Each column was standardized to have a mean of 0 and a standard deviation of 1. The remaining missing values were then filled with 0s. This makes the assumption that the missing values are at the mean of that time interval. This removes variance-shape differences between stations and TMCs but introduces bias by filling missing values with 0s.

[4]Given its random nature and relative low computational cost, the median heuristic was run 100 times and the median of those results were used as the final $\gamma$ for that particular experiment. The code for the median heuristic followed by a usage example can be found in Appendix C.1.

[5]This number was chosen to have reasonable runtimes for the large datasets. A single MMD computation could take as long as 10 minutes on an NVIDIA RTX 3090 GPU. This was reduced to 20 seconds per MMD using a cloud provided NVIDIA H200 GPU. With higher end GPUs and more compute resources, the number of permutations could be increased.

[6]It is worth noting that PORTAL had the most missing data with several stations having less than 90% coverage for the year with one having less than 30% coverage. The INRIX data appeared to have at most 4 missing intervals for a given year.

2. **"Masking"**: A masked version of both the gamma heuristic and MMD computation were used to ignore missing values or values that did not overlap with the other dataset. This remains faithful to the observed data but can be dominated by high-variance time intervals in which noise differences can inflate the MMD value[7].

3. **"Z-scaled and Masking"**: A combination of the two previous strategies where the data was standardized and instead of filling missing values with 0s, they were kept unfilled (NaNs) and masked during the gamma heuristic and MMD computation. This is the strictest approach that isolates mean and shape differences from missingness and variance magnitude.

# 5 Data Description

The data used in this project was collected from two sources: PORTAL and INRIX. The data sources have travel-time recordings for all the highways within ODOT Region 1, some outside the region and SW Washington. The data collected for this project was a subset of available data for the Portland, OR - Vancouver, WA Metro area and spanned the years 2019 to 2024 focusing on SR-14 (WA), I-5 (OR and WA), and I-205 (OR and WA).

## 5.1 PORTAL

PORTAL is a public transportation data platform that provides access to a variety of transportation data sources. PORTAL data includes real-time transit data, historical transit data, and transit schedules. This data is used by transportation agencies, businesses, and researchers to monitor transit operations, plan transit projects, and analyze transit patterns[9].

- **'stations' table**: Contains the unique identifiers for each station, the highway (by unique ID) the station is on, and the Well Known Binary (WKB) representation of the station's coverage geometry.
- **'highways' table**: Contains the unique identifiers for each highway, a highway name and the direction or bound of travel for the highway (N, S, E, W).
- **'travel-times' table**: Contains the 15-minute interval travel-time readings for each station in the PORTAL dataset. Each reading includes the station ID, the timestamp of the reading, and the travel-time in seconds.

## 5.2 INRIX

INRIX is a private company that collects and analyzes traffic data. They provide a variety of data products, including real-time traffic data, historical traffic data, and traffic forecasts. INRIX data is collected from a variety of sources, including GPS data from vehicles, mobile devices, and road sensors. INRIX data is used by transportation agencies, businesses, and researchers to monitor traffic conditions, plan transportation projects, and analyze traffic patterns[9].

[7] This increases the memory complexity and forces the materialization of the kernel matrix that PyKeOps was designed to avoid. However, in this view the data was considerably wider than it was tall (35,040 features vs ∼100 samples) and the kernel matrix was manageable on even a consumer grade GPU with pure PyTorch. Memory complexity did prevent consumer-grade GPUs from running the time-centric view when the number of rows jumped to 35,040

- **TMC shape files**: Contains the unique `tmc_code` for each TMC, highway name, direction of travel, and the covered segment's geometry in Well Known Text (WKT) format. The Shapefiles were limited to 2019-2023, even though 2024 travel-times were available. As such the spatial feature view was not computed for 2024.
- **TMC travel-times**: Contains the 15-minute interval travel-time readings for each TMC in the INRIX dataset. Each reading includes the TMC code, the timestamp of the reading, and the travel-time in minutes.

### 5.3 Preprocessing and Alignment

The data was preprocessed to ensure consistency in format, units, and time zones. The geometries of each dataset were converted to EPSG:3857 (Web Mercator) for consistency and to allow for easier spatial joining and determining closeness using meters as the unit of measurement instead of degrees. The travel-times were converted to seconds to ensure both datasets were in the same units.

- **PORTAL:** This data required converting the WKB geometries to WKT format in order to determine spatial closeness to the INRIX geometries[8].
- **INRIX:** The shape files were converted to parquet format for easier handling and the travel-times recorded in minutes were converted to seconds to match the PORTAL data. The opportunity was also taken to group Shapefiles by year rather than as provided–by region[9] to account for segment geometry changes over time.

## 6 Data Orientation

The raw data consisted of 15-minute interval travel-time readings from the PORTAL and INRIX datasets for the Portland Metro area. The data spanned the years 2019 to 2024 and included several highways and corridors. The data was preprocessed to ensure consistency in format, units, and time zones. The data was reshaped into three different views to answer different questions about the distributions of travel-time readings between the two datasets.

### 6.1 View 1 (1D Readings Row-by-Row): *Are the overall distributions of travel-time readings from PORTAL and INRIX different?*

The first view of the data was to take the single (1D) feature of 15-minute interval travel-time readings for every station in the PORTAL dataset and every TMC in the INRIX dataset. This ignores spatial pairings and time alignment and cannot detect correlation or joint structure differences. This is also the most computationally intensive view of the data and required the use of PyKeOps.

[8]The geometries available significantly reduced the number of PORTAL stations that could be matched to INRIX TMCs. The geometries were also generated at the time the data was collected and may not necessarily represent the exact location of the road segment a station covers at a given time. This is a limitation of the data and not the preprocessing.

[9]The INRIX shape files were provided by year and regions–Oregon and Clark County, Washington.

Table 1: Unfiltered samples and cross-dataset kernel Sizes by year.

| Year | PORTAL | | INRIX | | Total Samples | Kernel Size (P×I) |
|---|---|---|---|---|---|---|
| | Stations | Samples | TMCs | Samples | | |
| 2019 | 107 | 3,163,158 | 139 | 4,856,660 | 8,019,818 | $1.54 \times 10^{13}$ |
| 2020 | 115 | 3,688,298 | 139 | 4,841,300 | 8,529,598 | $1.79 \times 10^{13}$ |
| 2021 | 119 | 3,780,171 | 139 | 4,865,012 | 8,646,183 | $1.84 \times 10^{13}$ |
| 2022 | 114 | 3,793,532 | 139 | 4,870,004 | 8,663,536 | $1.85 \times 10^{13}$ |
| 2023 | 116 | 3,927,778 | 139 | 4,870,004 | 8,797,782 | $1.91 \times 10^{13}$ |
| 2024 | 134 | 4,352,721 | 139 | 4,883,348 | 9,236,069 | $2.13 \times 10^{13}$ |

### 6.2 View 2 (Entity-centric Full Year Vectors): *Do stations and TMCs differ in their overall temporal patterns?*

The second view of the data was to pivot and reshape the data such that each station or TMC had a single vector of 15-minute interval travel-time readings for the entire year. The data was still not filtered or matched spatially to each other. The time intervals were aligned to a 1-year grid of 15-minute intervals where some stations might not have a reading for the specific timestamp but, it ensured the features were aligned in time and of the same length ($\sim$ 35,040 intervals).

Table 2: Unfiltered samples and features by year when reshaped to full year vectors.

| Year | **PORTAL** (samples × features) | **INRIX** (samples × features) |
|---|---|---|
| 2019 | 107 × 35,040 | 139 × 35,040 |
| 2020 | 115 × 35,136 | 139 × 35,136 |
| 2021 | 119 × 35,040 | 139 × 35,040 |
| 2022 | 114 × 35,040 | 139 × 35,040 |
| 2023 | 116 × 35,040 | 139 × 35,040 |
| 2024 | 134 × 35,040 | 139 × 35,040 |

### 6.3 View 3 (Time-centric System Snapshot Vectors): *Do stations and TMCs differ in their spatial patterns at each moment?*

The third view of the data was to pivot and reshape the data such that each 15-minute interval had a single vector of travel-time readings for all stations or TMCs at that time. For this to work and ensure equal length feature vectors, the data was spatially joined using an improved version of the spatial join from a previous project[7]. The improved version[10] enforces a 1-to-1 mapping via the Hungarian algorithm[3][11] used in conjunction with segment overlap to break ties. Secondly, it ensures that the direction of travel was the same for each station and TMC pair.

[10]The improved version can be found on GitHub: `https://github.com/whitham-powell/TREC-PORTALvsINRIX-MMD/blob/master/create_portal_inrix_sjoin.py`
The original approach: `https://github.com/whitham-powell/cades-traveltime-compare/blob/main/demo_sjoin_portal_inrix_meta.py`

[11]A basic description and example implementation of the Hungarian algorithm: `https://en.wikipedia.org/wiki/Hungarian_algorithm`

Table 3: Filtered samples and features by year when reshaped to time-centric system snapshot vectors.

| **Year** | **PORTAL** (samples × features) | **INRIX** (samples × features) |
|---|---|---|
| 2019 | 35,040 × 36 | 35,040 × 36 |
| 2020 | 35,136 × 36 | 35,136 × 36 |
| 2021 | 35,040 × 63 | 35,040 × 63 |
| 2022 | 35,040 × 63 | 35,040 × 63 |
| 2023 | 35,040 × 63 | 35,040 × 63 |

### 6.4 Additional Missing Data Handling Strategy Dependent Questions (Views 2 and 3)

Depending on the missing data handling strategy, additional questions are raised for the entity-centric full year vector and time-centric (views 2 and 3 respectively).

1. **"Z-scaled and 0-fill"**: Are the distributions different if every time slot is put on the same scale and missing slots are treated as the slot's average (neutral)?
2. **"Masking"**: Are the distributions different when we only compare slots both observed, letting natural slot variance weight the distance?
3. **"Z-scaled and Masking"**: Are the distributions different on overlapping slots, with each slot equally weighted by scale?

## 7 Results

### 7.1 Overall trends in the results

Looking at the results in Figure 1, we can see that the $\widehat{\mathrm{MMD}^2}$ values as measured in standard deviations from the null distribution are decreasing over time in a general sense for each view. The clearest trend can be seen with the linear scale plots located in the Appendix (B, Figure 2).

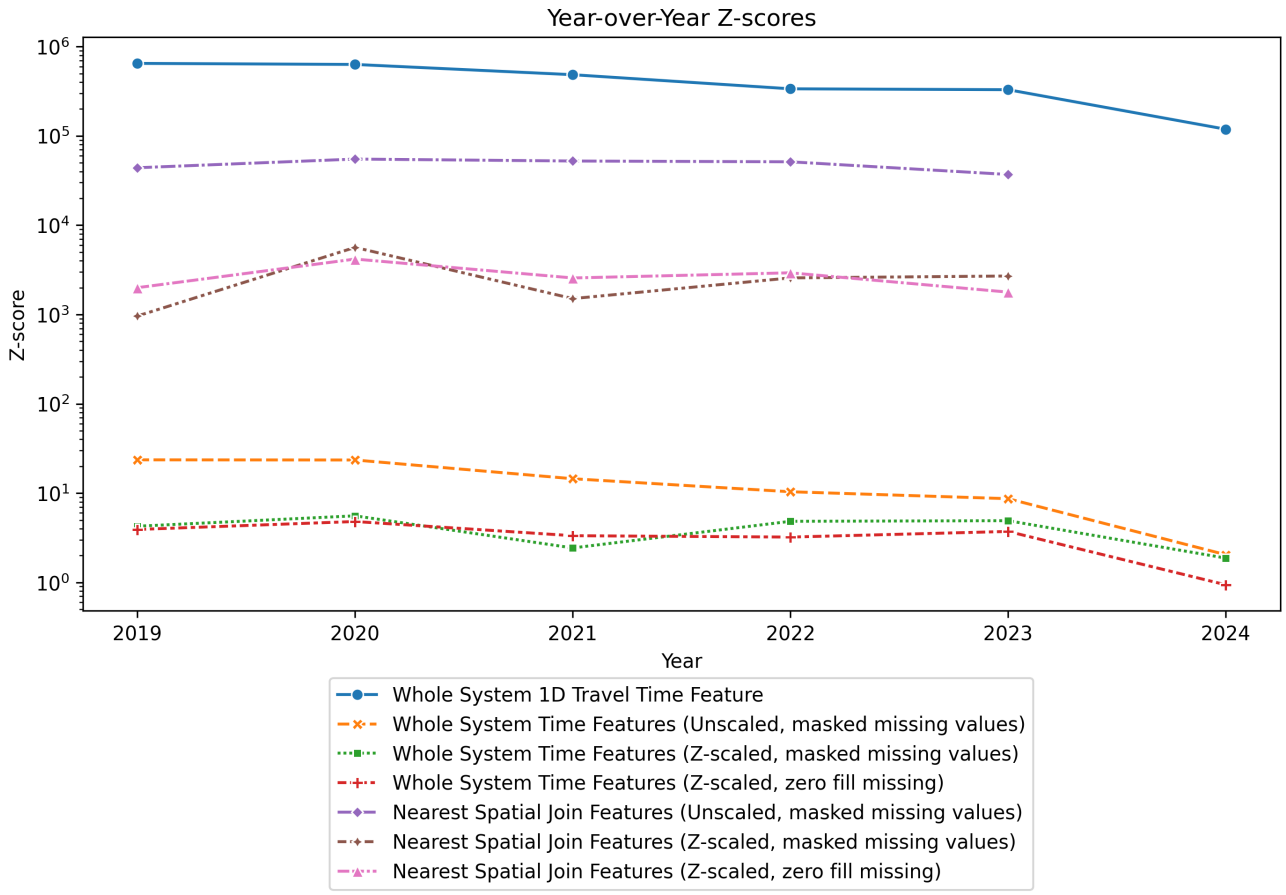



Figure 1: Z-scored $\widehat{\text{MMD}}^2$ results for all models and years with logarithmic y-axis. The y-axis is logarithmic to better show the differences between the models and account for the large range of values.

## 7.2 Comparing the views

The whole system 1D view shows the largest distances between the two datasets with the highest z-scores. The entity-centric full year vectors show the smallest distances between the two datasets with the lowest z-scores as a group and the time-centric system snapshot vectors fall in between the two other views.

- **View 1 (1D Readings Row-by-Row)**–***Are the overall distributions of travel-time readings from PORTAL and INRIX different?***: This view shows the largest differences between the two datasets with z-scores ranging from roughly 118-thousand to 648-thousand standard deviations away from the null distribution (See Table 4). This indicates that the overall distributions of travel-time readings from PORTAL and INRIX are significantly different to an extreme degree. This view is the most sensitive to differences in the distributions as it does not account for spatial or temporal correlations.

- **View 2 (Entity-centric Full Year Vectors)**–***Do stations and TMCs differ in their overall temporal patterns?***: This view shows the smallest differences between the two datasets with z-scores ranging from 0.94 to 23 standard deviations away from the null distribution (See Table 4).

This indicates that while there are still significant differences between the two datasets, they are less pronounced when looking at the overall temporal patterns of stations and TMCs. This view accounts for temporal correlations but does not account for spatial correlations.

- **View 3 (Time-centric System Snapshot Vectors)**–***Do stations and TMCs differ in their spatial patterns at each moment?***: This view shows differences between the two datasets with z-scores ranging from just under 18-hundred to just over 55-thousand standard deviations away from the null distribution (See Table 4). This indicates that there are significant differences between the two datasets when looking at the spatial patterns at each moment. This view accounts for spatial correlations but does not account for temporal correlations.

#### 7.2.1 View 2: Entity-centric vs View 3: Time-centric

The entity-centric full year vectors show smaller distances between the two datasets compared to the time-centric system snapshot vectors. This suggests that the temporal patterns of stations and TMCs are more similar between the two datasets than the spatial patterns at each moment. This could be due to the fact that the temporal patterns are more stable over time, while the spatial patterns can be more variable due to factors such as traffic conditions, road closures, and other events. Additionally, the time-centric view has significantly less total sensors in each dataset (36-63) compared to the entity-centric view (107-139). This could lead to a loss of information and potentially increase the differences between the two datasets. This could be improved with better overlapping spatial coverage or being more lenient with what is considered the nearest match in the spatial join.

### 7.3 Comparing the Missing Data Handling Strategies (Views 2 and 3)

With both the entity-centric full year vectors and time-centric system snapshot vectors, the missing data handling strategies indicate when values are scaled and missing values are filled with 0s (neutral), the distances between the two datasets on average are smaller (or more similar) than when masked alone. Unscaled and masked shows the greatest difference between the two datasets in both views 2 and 3. This suggests that the missing data handling strategy can have a significant impact on the measured differences between the two datasets. The neutral strategy assumes that the missing values are at the mean of that time interval, which can reduce the impact of missing data on the MMD computation. The masking strategy, on the other hand, remains faithful to the observed data but can be dominated by high-variance time intervals in which noise differences can inflate the MMD value.

### 7.4 The raw z-scores in Table 4

These values show just how extreme the differences are between the two datasets in the various views. **Higher z-scores indicate that the observed $\widehat{\mathrm{MMD}}^2$ value lies farther from the null mean—meaning the two datasets are more distributionally different—while lower z-scores indicate greater similarity.** These scores are number of deviations away from 0 (the case the distributions are the same with *characteristic kernels*). For example, the 1D view indicates the distributions might as well be on different planets. Similarly, the time-centric view even after handling missing data with the strictest approaches are completely unrelated. The entity-centric view is less extreme but still shows significant differences with improvement overtime.

Table 4: Z-scored $MMD^2$ results for all models and years. The values represent the distance in standard deviations from the null distribution from which the permutation test generated after 500 permutations.

| Representation | Whole System | | | | Nearest Spatial Join | | |
|---|---|---|---|---|---|---|---|
| Feature Type | 1D TT | Time (sens × time) | | | Spatial (time × sens) | | |
| Scaling | Unscaled | Unscaled | Z | | Unscaled | Z | |
| Missing | – | masked | masked | zero fill | masked | masked | zero fill |
| 2019 | 648693.88 | 23.59 | 4.26 | 3.92 | 44011.38 | 962.16 | 1998.15 |
| 2020 | 631333.43 | 23.50 | 5.57 | 4.81 | 55039.83 | 5645.24 | 4177.29 |
| 2021 | 484444.32 | 14.49 | 2.43 | 3.34 | 52340.42 | 1507.19 | 2566.20 |
| 2022 | 336767.42 | 10.36 | 4.84 | 3.22 | 51282.97 | 2570.04 | 2931.23 |
| 2023 | 329135.24 | 8.67 | 4.91 | 3.72 | 36983.19 | 2698.92 | 1781.34 |
| 2024 | 118751.24 | 2.04 | 1.87 | 0.94 | – | – | – |

# 8 Conclusions

Table 5: Questions answered by the core views of the data.

| **View** | **Question** | **Answer** |
|---|---|---|
| **View 1:** Whole System 1D Vectors | Are the overall distributions of travel-time readings from PORTAL and INRIX different? | Yes |
| **View 2:** Entity-centric Full Year Vectors | Do stations and TMCs differ in their overall temporal patterns? | Yes |
| **View 3:** Time-centric System Snapshot | Do stations and TMCs differ in their spatial patterns at each moment? | Yes |

Table 6: Questions answered by the missing data handling strategies.

| **Missing Strategy:** | **Question** | **Answer** |
|---|---|---|
| **Neutral** (*Z-scaled and 0-fill*) | Are the distributions different if every time slot is put on the same scale and missing slots are treated as the slot's average (neutral)? | Yes |
| **Weighted** (*Masking*) | Are the distributions different when we only compare slots both observed, letting natural slot variance weight the distance? | Yes |
| **Equal** (*Z-scaled and masking*) | Are the distributions different on overlapping slots, with each slot equally weighted by scale? | Yes |

Every view and approach to missing data indicated through its z-scored MMD that the distributions of the travel-time readings between the two datasets come from different underlying distributions. This does not provide an insight into which dataset is more accurate, or why the differences exist. What we

can conclude is that there is a pattern of the datasets trending towards more similar than dissimilar over time despite the differences in the underlying data collection methods.

## 9 Future Work

**Drill into more granular views of the data.** These could include time based views such as weekdays vs weekends, or holidays, or time of day. Spatial based views that look at directions of travel such as north vs south or east vs west or even specific highways or corridors. Finally, a more granular spatio-temporal view could be explored such as the joint distribution of a single highway or corridor over the course of a specific day to a specific time of year.

**Additional road systems and highways.** At the system level it could be plausible to include a more comprehensive set of highways and roads in the Portland Metro area for which data is available in both datasets. This would increase the number of samples and potentially the number of features in the higher dimensional views. This could also include other cities or regions where both datasets have coverage.

**Additional features.** Outside the realm of reshaping or filtering travel-times, additional features could be added such as weather, lengths of segments covered, traffic counts, or other relevant data. These additional features could be added to the feature vectors of each view whether it be the 1D readings, entity centric full year vectors, or time-centric system snapshot vectors.

**Monitor MMD estimates over time.** Assuming that a convincing measurable difference exists, then a related avenue of research would be to continue to monitor MMD estimates over time to see if the trend towards more similar continues in future years and eventually converges to a point where the datasets are statistically indistinguishable.

**The MMD and Permutation Test Framework is flexible and extensible.** Essentially, with the MMD and permutation test framework established, it is a matter of reshaping and or filtering the data to answer new questions.

## References


[1] Benjamin Charlier, Jean Feydy, Joan Alexis Glaunès, François-David Collin, and Ghislain Durif. Kernel operations on the gpu, with autodiff, without memory overflows. *Journal of Machine Learning Research*, 22(74):1–6, 2021. URL: http://jmlr.org/papers/v22/20-275.html.

[2] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773, 2012. URL: http://jmlr.org/papers/v13/gretton12a.html.

[3] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. doi:10.1002/nav.3800020109.

[4] Junyi Li, Fangce Guo, Yibing Wang, Lihui Zhang, Xiaoxiang Na, and Simon Hu. Short-term traffic prediction with deep neural networks and adaptive transfer learning. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6, 2020. doi:10.1109/ITSC45102.2020.9294409.

[5] Krikamol Muandet, Kenji Fukumizu, Bharath Sriperumbudur, and Bernhard Schölkopf. Kernel mean embedding of distributions: A review and beyond. *Foundations and Trends® in Machine Learning*, 10(1-2):1–141, 2017. URL: http://dx.doi.org/10.1561/2200000060, doi:10.1561/2200000060.

[6] Bharath K. Sriperumbudur, Kenji Fukumizu, and Gert R.G. Lanckriet. Universality, characteristic kernels and rkhs embedding of measures. *Journal of Machine Learning Research*, 12(70):2389–2410, 2011. URL: http://jmlr.org/papers/v12/sriperumbudur11a.html.

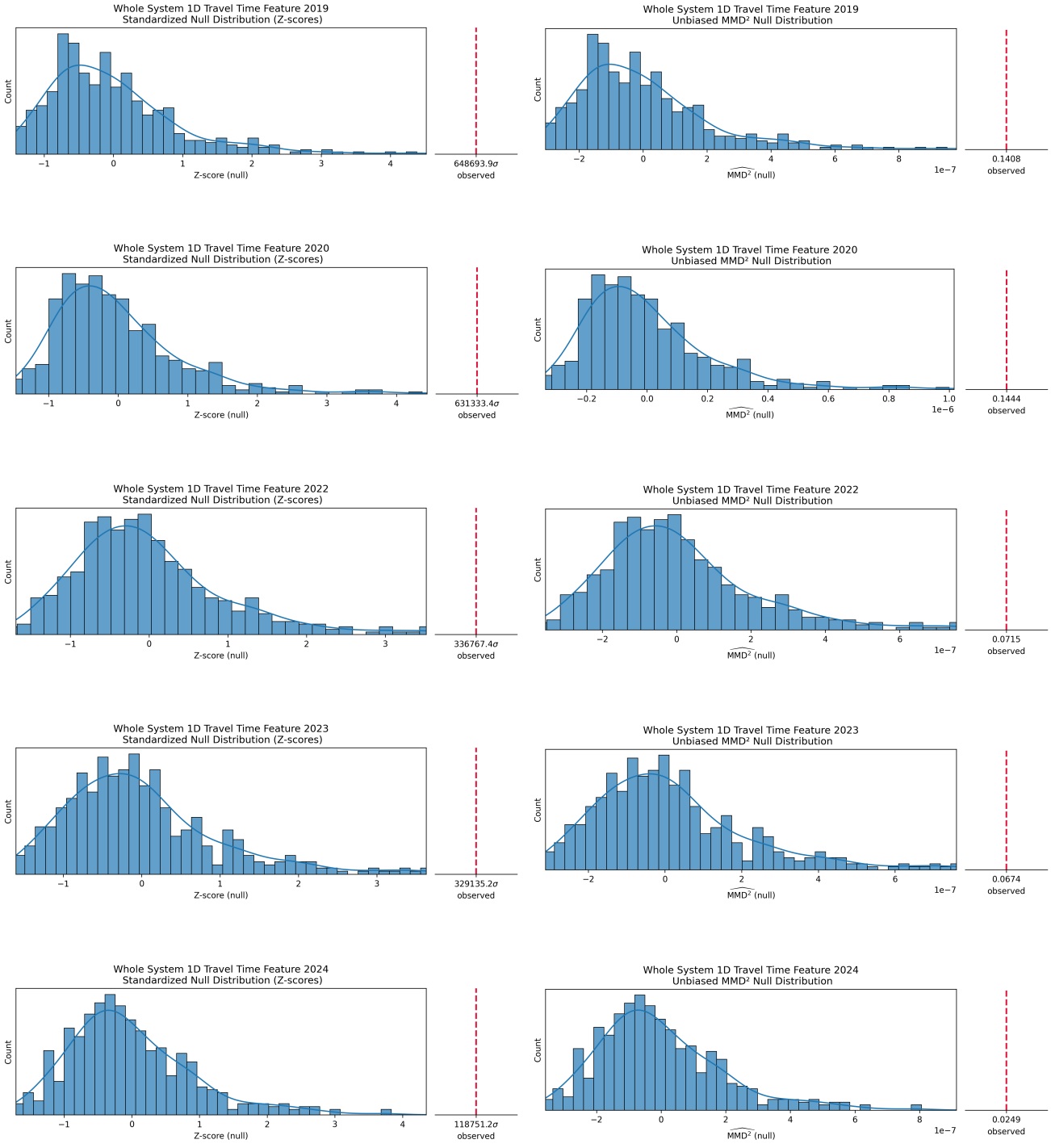
[7] Elijah Whitham-Powell. Computational and data-enabled sciences (CADES) project code. GitHub, 2025. URL: https://github.com/whitham-powell/cades-traveltime-compare.

[8] Elijah Whitham-Powell. TREC: PORTAL vs INRIX MMD analysis. GitHub, 2025. URL: https://github.com/whitham-powell/TREC-PORTALvsINRIX-MMD.

[9] Elijah Whitham-Powell and Meghana Cyanam. Travel time calculation across different data sources. Self-published, 2025. URL: https://whitham-powell.com/papers/CADES_PORTALvsINRIX_report.pdf.

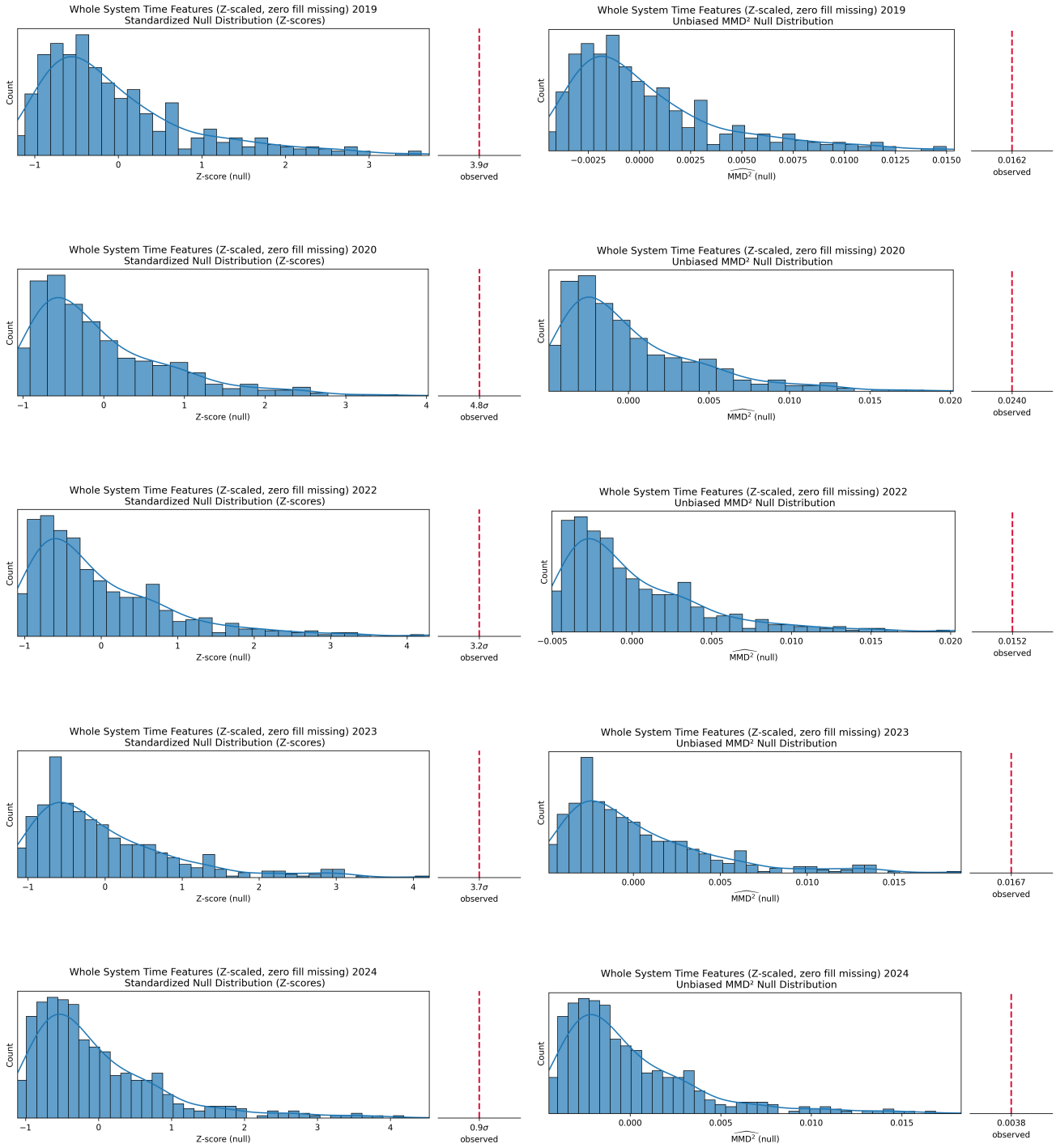
# A All Computed MMD Permutation Test Histograms

## A.1 View 1: 1D Readings Row-by-Row

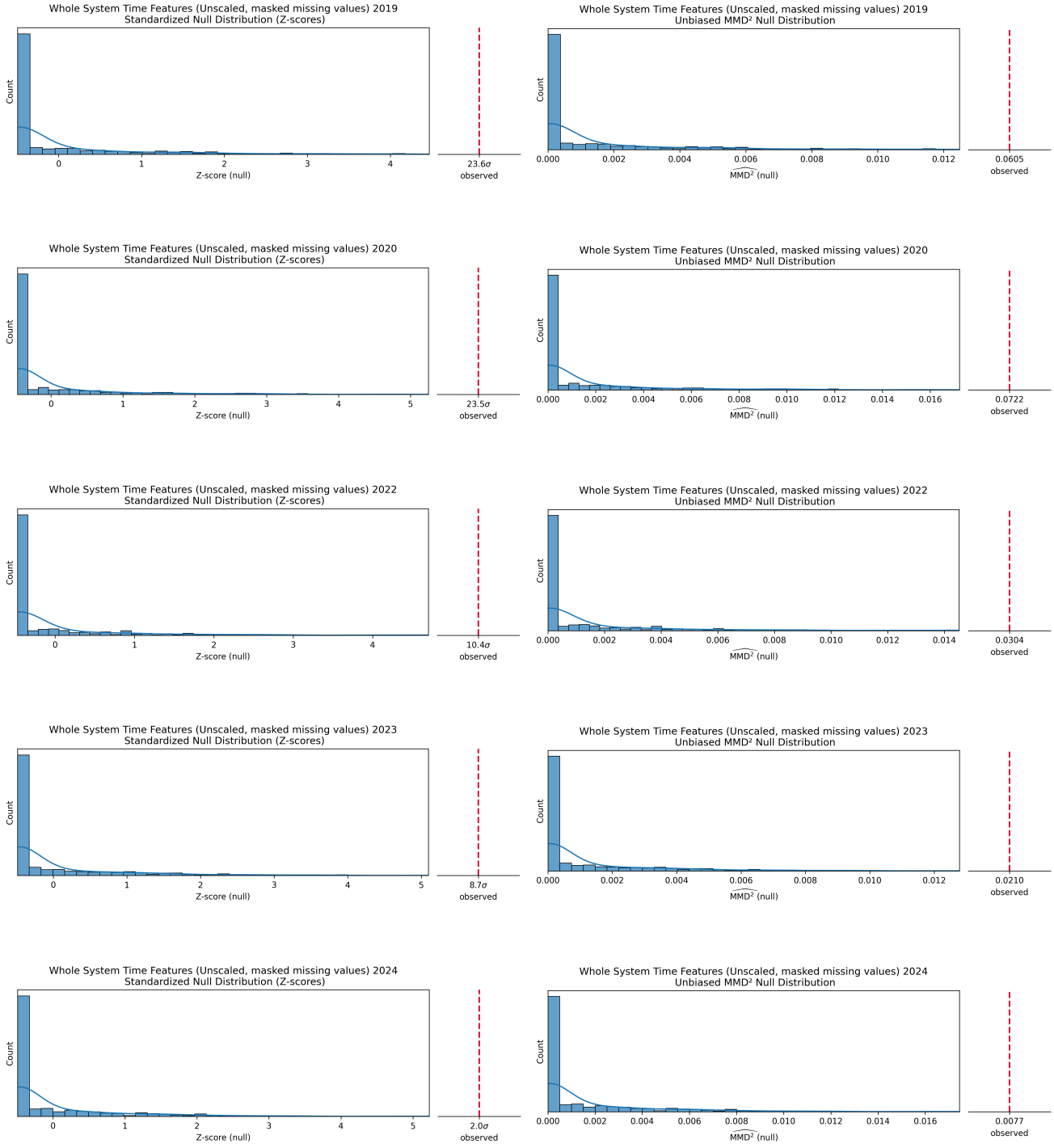


## A.2 View 2: Entity Centric Full Year Vectors

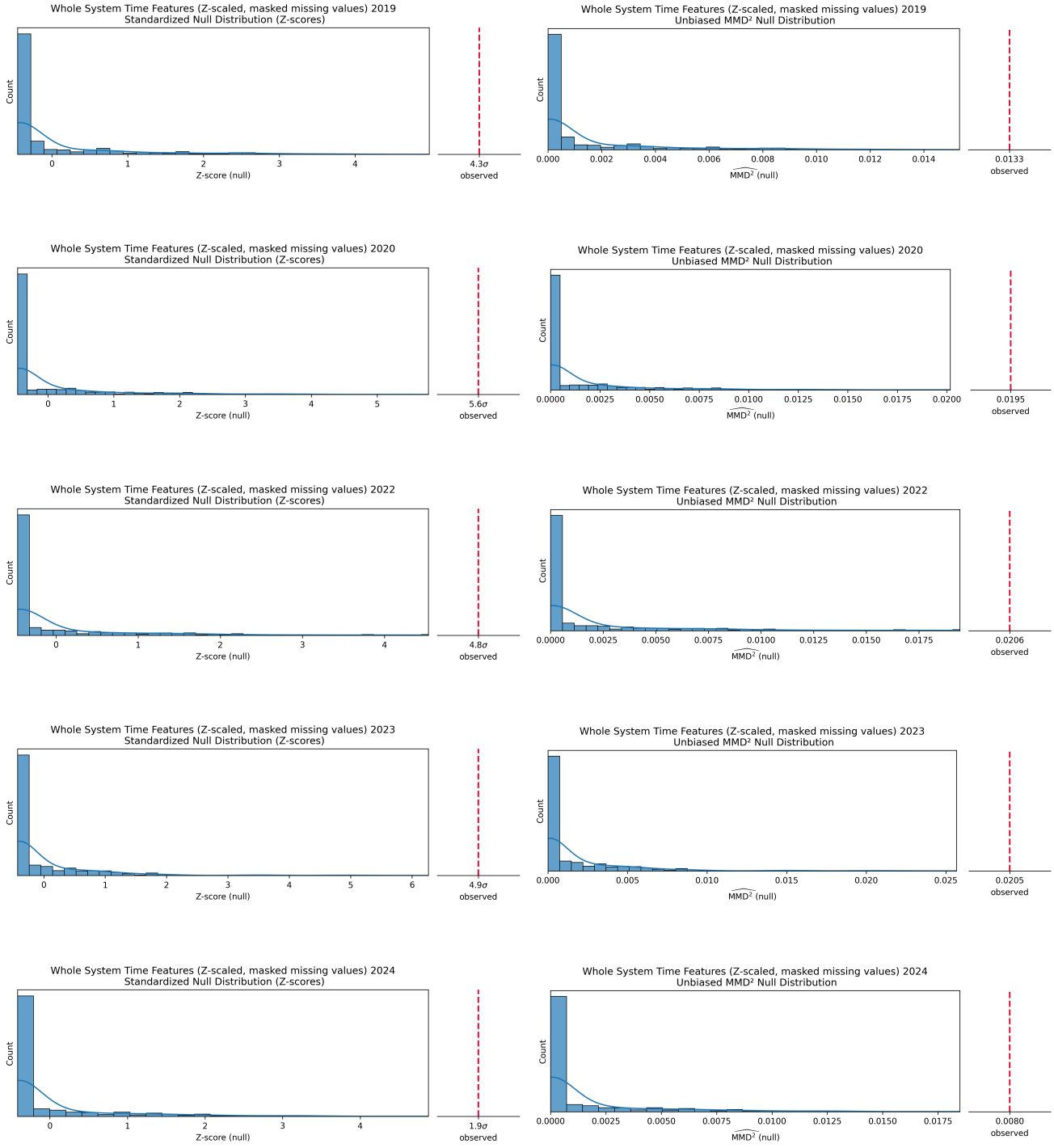
### A.2.1 Z scaled and 0 filled



## A.2.2 Unscaled and Masked

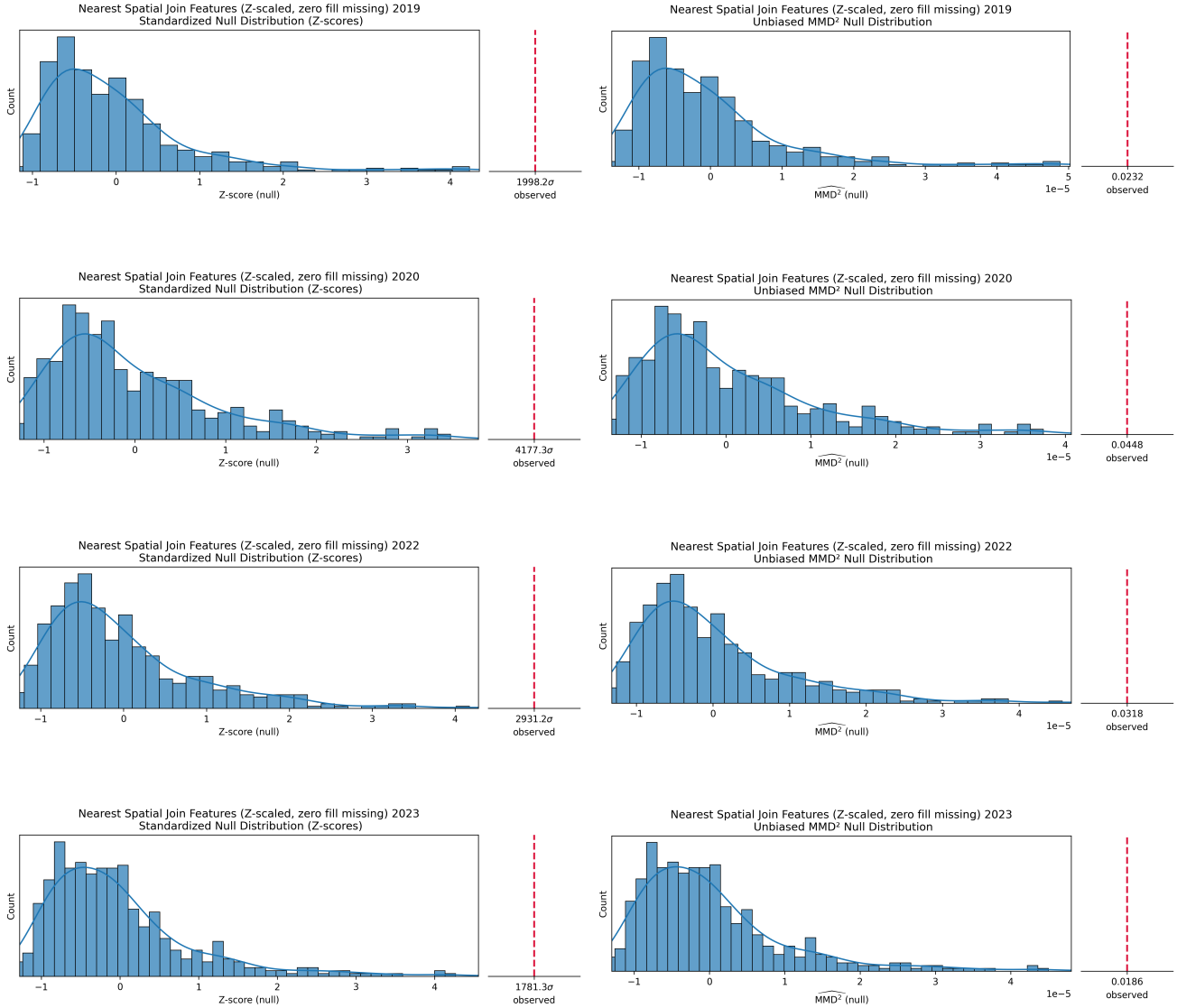


### A.2.3 Z scaled and Masked

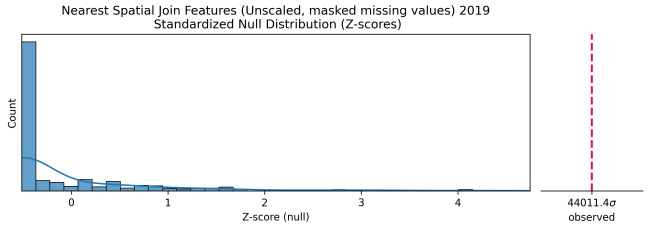


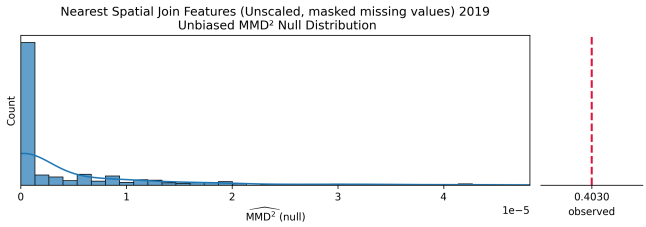
## A.3 View 3: Time-centric System Snapshot Vectors

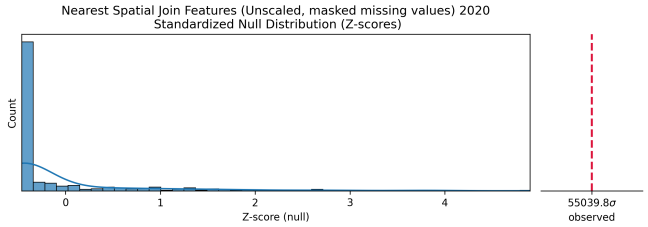
### A.3.1 Z scaled and 0 filled

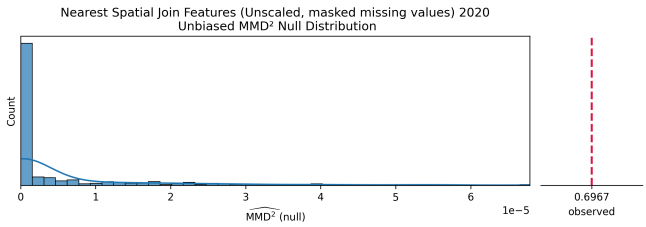


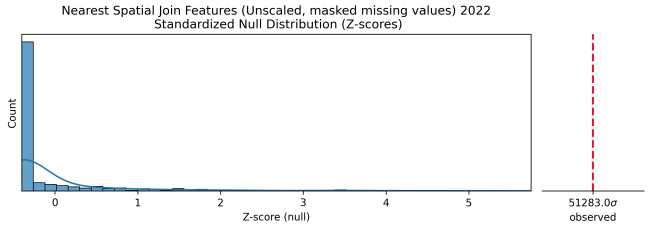
## A.3.2 Unscaled and Masked

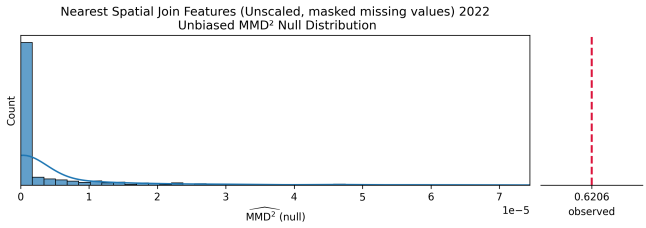


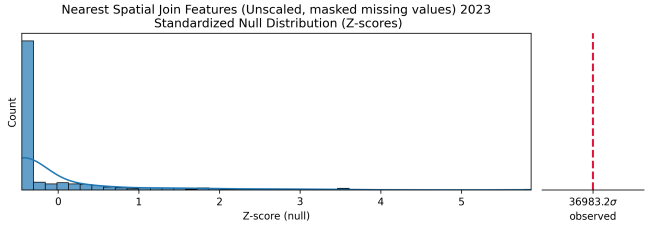



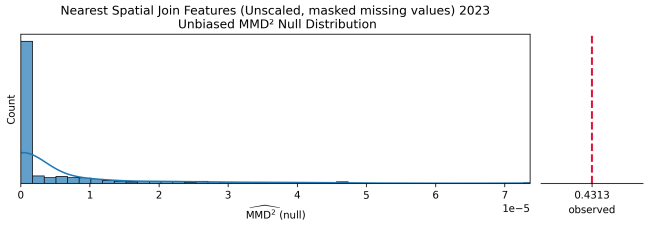



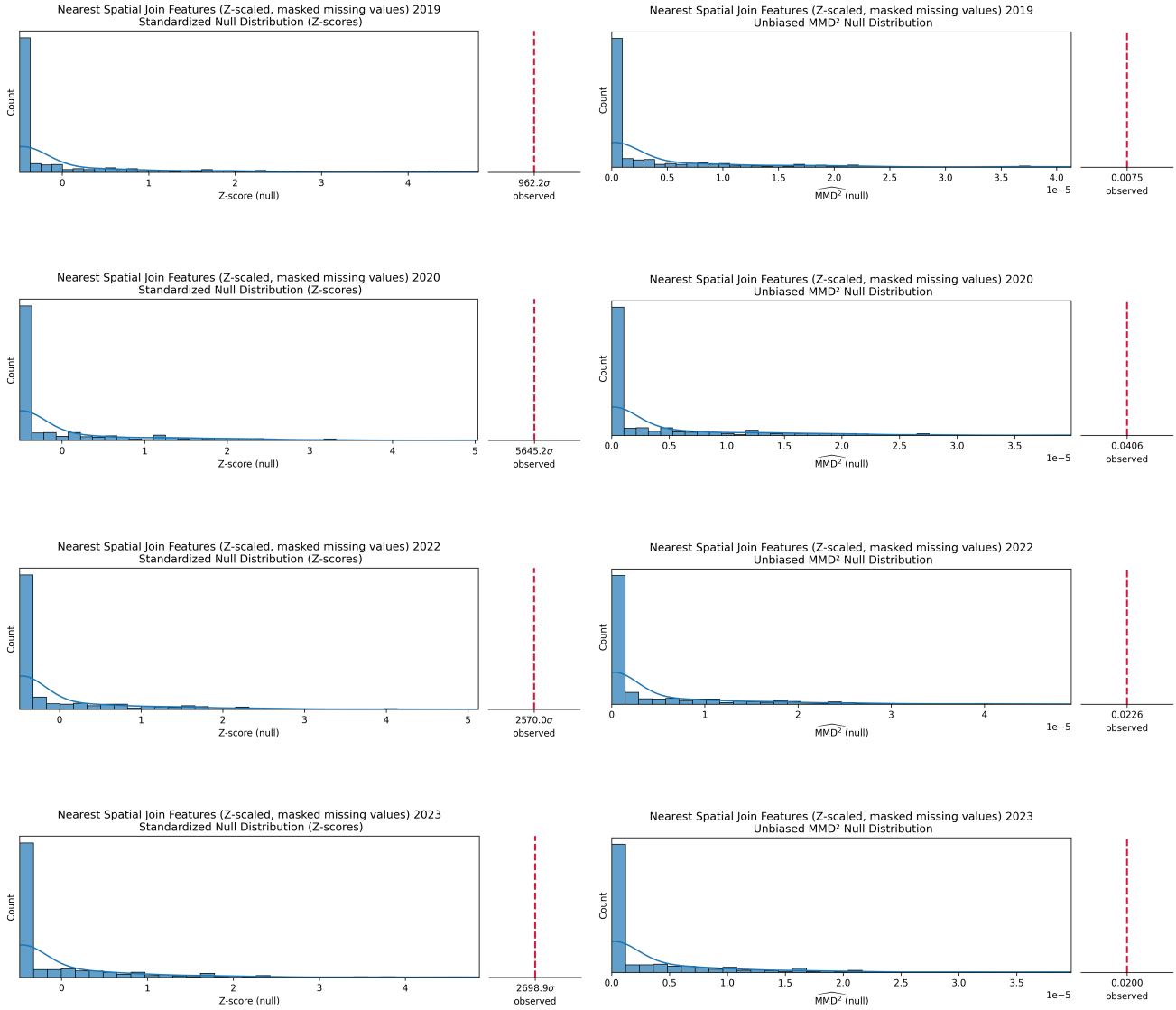



### A.3.3 Z scaled and Masked



# B Views over time plots

Figure 2: 1D Travel Time Feature over time (linear y-axis)

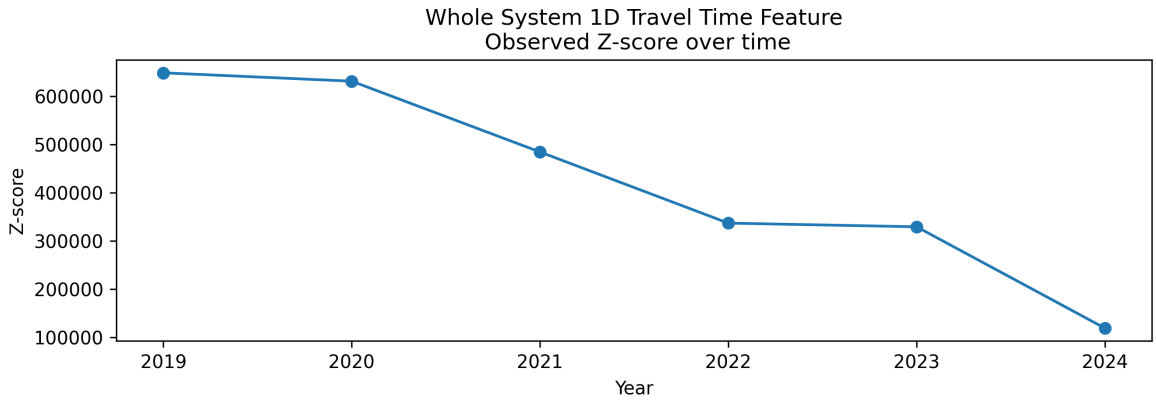
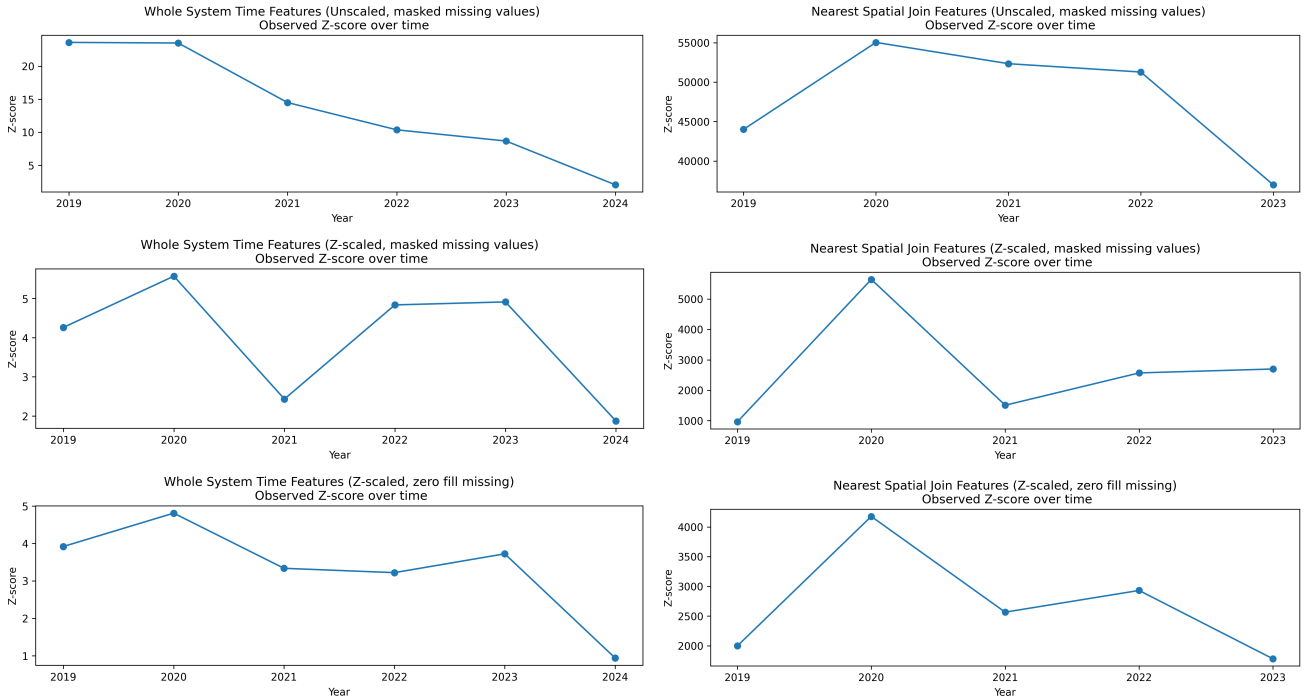



Figure 3: Time and Spatial Features over time (linear y-axis)



## C Code Snippets

The following code snippets were used to compute the MMD estimates and permutation tests. The full complete code can be found on the GitHub repository: `https://github.com/whitham-powell/TREC-PORTALvsINRIX-MMD` and may continue to evolve after the submission of this report.

## C.1 Gamma Median Heuristic

```python
def gamma_from_median_heuristic(
    Z: torch.Tensor,
    max_samples: int = 4000,
    chunk: int = 512,
    g: torch.Generator = None,
) -> float:
    """
    Median heuristic on pooled data Z (n,d).
    Subsamples up to max_samples rows; computes median pairwise distance in chunks.
    Returns gamma = 1 / (2 * sigma^2) with sigma = median distance.
    """
    device = Z.device
    n = Z.shape[0]
    m = min(n, max_samples)

    # sample without replacement (GPU)
    idx = torch.randperm(n, device=device, generator=g)[:m]
    S = Z.index_select(0, idx)  # (m,d)

    meds = []
    for i0 in range(0, m, chunk):
        i1 = min(i0 + chunk, m)
        A = S[i0:i1]  # (a,d)
        for j0 in range(i0, m, chunk):
            j1 = min(j0 + chunk, m)
            B = S[j0:j1]  # (b,d)
            # squared distances (avoid sqrt until the end)
            sq = (A[:, None, :] - B[None, :, :]).pow(2).sum(-1)  # (a,b)
            if i0 == j0:
                tri = torch.triu(sq, diagonal=1)
                vals = tri[tri > 0]
            else:
                vals = sq.reshape(-1)
            if vals.numel():
                meds.append(vals)
    if not meds:
        return 1e-6 # degenerate fallback

    d2 = torch.cat(meds)  # squared distances
    med = d2.sqrt().median().item()  # median distance (not squared)
    sigma2 = med * med
    return float(1.0 / (2.0 * sigma2))
```

---

**Median Heuristic Usage Example:**

```
torch_rng = torch.Generator(device=Z.device).manual_seed(42)
gamma = torch.tensor(
    [
        gamma_from_median_heuristic(
            Z, g=torch_rng
        )
        for _ in range(100)
    ],
    device=Z.device,
    dtype=Z.dtype,
)
gamma = torch.median(gammas)  # median of 100 median heuristic gammas
mmd = mmd_keops(X, Y, gamma=gamma)
```

## C.2 MMD Implementation with PyKeOps

```python
def mmd_keops(X, Y, gamma=None):

    assert X.shape[1] == Y.shape[1], "X and Y must have the same number of features"

    M, N, d = X.shape[0], Y.shape[0], X.shape[1]
    # KeOps symbolic variables
    x_i = LazyTensor(X[:, None, :])  # (M, 1, d)  - indexed by i
    x_j = LazyTensor(X[None, :, :])  # (1, M, d)  - indexed by j
    y_i = LazyTensor(Y[:, None, :])  # (N, 1, d)
    y_j = LazyTensor(Y[None, :, :])  # (1, N, d)

    # RBF kernel with gamma = 1 / d   (matches scikit-learn's default)
    gam = gamma if gamma is not None else 1.0 / d

    K_xx = (-gam * x_i.sqdist(x_j)).exp()  # (M, M) symbolic block
    K_yy = (-gam * y_i.sqdist(y_j)).exp()  # (N, N)
    K_xy = (-gam * x_i.sqdist(y_j)).exp()  # (M, N)

    # Total sums of the three blocks
    S_xx = K_xx.sum(dim=1).sum()  # torch scalar
    S_yy = K_yy.sum(dim=1).sum()
    S_xy = K_xy.sum(dim=1).sum()

    # Diagonal corrections
    # diag_xx_sum = (-gam * x_i.sqdist(x_i)).exp().sum(dim=1).sum()  # = M for RBF
    # diag_yy_sum = (-gam * y_i.sqdist(y_i)).exp().sum(dim=1).sum()  # = N
    diag_xx_sum = torch.tensor(
        M, dtype=torch.float64, device=X.device
    )  # for RBF kernel
    diag_yy_sum = torch.tensor(
        N, dtype=torch.float64, device=Y.device
    )  # for RBF kernel

    # Unbiased MMD  (all plain torch scalars from here on)
    M_f = torch.tensor(float(M), dtype=torch.float64, device=X.device)
    N_f = torch.tensor(float(N), dtype=torch.float64, device=Y.device)

    XX = (S_xx.to(torch.float64) - diag_xx_sum) / (M_f * (M_f - 1.0))
    YY = (S_yy.to(torch.float64) - diag_yy_sum) / (N_f * (N_f - 1.0))
    XY = S_xy.to(torch.float64) / (M_f * N_f)

    mmd2 = XX + YY - 2 * XY  # torch scalar

    return mmd2.item()
```

### C.3 MMD Implementation with Pure PyTorch

```python
def _pairwise_sqdist(X: torch.Tensor, Y: torch.Tensor) -> torch.Tensor:
    """X:(M,d), Y:(N,d) -> D2:(M,N) without (M,N,d) broadcast."""
    X2 = (X * X).sum(dim=1, keepdim=True)  # (M,1)
    Y2 = (Y * Y).sum(dim=1, keepdim=True).T  # (1,N)
    D2 = X2 + Y2 - 2.0 * (X @ Y.T)
    return D2.clamp_min(0)


def mmd_torch(X: torch.Tensor, Y: torch.Tensor, gamma: float | None = None) -> float:
    """Unbiased MMD^2 with RBF kernel (unmasked)."""
    assert X.shape[1] == Y.shape[1], "feature dims must match"
    M, N, d = X.shape[0], Y.shape[0], X.shape[1]
    gam = gamma if gamma is not None else 1.0 / d

    D2_xx = _pairwise_sqdist(X, X)
    D2_yy = _pairwise_sqdist(Y, Y)
    D2_xy = _pairwise_sqdist(X, Y)

    K_xx = torch.exp(-gam * D2_xx)
    K_yy = torch.exp(-gam * D2_yy)
    K_xy = torch.exp(-gam * D2_xy)

    S_xx = K_xx.sum()
    S_yy = K_yy.sum()
    S_xy = K_xy.sum()

    diag_xx = torch.tensor(M, dtype=torch.float64, device=X.device)
    diag_yy = torch.tensor(N, dtype=torch.float64, device=Y.device)

    M_f = torch.tensor(float(M), dtype=torch.float64, device=X.device)
    N_f = torch.tensor(float(N), dtype=torch.float64, device=Y.device)

    XX = (S_xx.to(torch.float64) - diag_xx) / (M_f * (M_f - 1.0))
    YY = (S_yy.to(torch.float64) - diag_yy) / (N_f * (N_f - 1.0))
    XY = S_xy.to(torch.float64) / (M_f * N_f)

    return float((XX + YY - 2.0 * XY).clamp_min(0))
```

## C.4 Masked Variants Implementation with pure PyTorch

### C.4.1 Masked MMD

```python
def mmd_torch_masked(
    X: torch.Tensor, Mx: torch.Tensor,
    Y: torch.Tensor, My: torch.Tensor,
    gamma: float | None = None,
    eps: float = 1e-9,
    rescale_by_d: bool = True,
    use_valid_pair_denominators: bool = True,
) -> float:
    """
    Unbiased masked MMD^2 with RBF kernel (pure Torch).
    - Mask inside the sum (mean over overlaps), no (M,N,d) tensors.
    - Invalid pairs have kernel=0 and are excluded via denominators.
    """
    assert X.shape[1] == Y.shape[1], "feature dims must match"
    assert X.shape == Mx.shape and Y.shape == My.shape, "data/mask shapes must match"
    M, N, d = X.shape[0], Y.shape[0], X.shape[1]
    gam = gamma if gamma is not None else 1.0 / d

    D2_xx, V_xx = _masked_pairwise_sqdist_mean(
        X, Mx, X, Mx, eps=eps, rescale_by_d=rescale_by_d
    )
    D2_yy, V_yy = _masked_pairwise_sqdist_mean(
        Y, My, Y, My, eps=eps, rescale_by_d=rescale_by_d
    )
    D2_xy, V_xy = _masked_pairwise_sqdist_mean(
        X, Mx, Y, My, eps=eps, rescale_by_d=rescale_by_d
    )

    K_xx = torch.exp(-gam * D2_xx) * V_xx
    K_yy = torch.exp(-gam * D2_yy) * V_yy
    K_xy = torch.exp(-gam * D2_xy) * V_xy

    S_xx_all = K_xx.sum()
    S_yy_all = K_yy.sum()
    S_xy = K_xy.sum()

    # Diagonal corrections: rows with >= observed feature have K(ii)=1
    diag_xx = (Mx.sum(dim=1) > 0).to(torch.float64).sum().to(torch.float64)
    diag_yy = (My.sum(dim=1) > 0).to(torch.float64).sum().to(torch.float64)

    if use_valid_pair_denominators:
        Den_xx = (V_xx.sum().to(torch.float64) - diag_xx).clamp_min(1.0)
        Den_yy = (V_yy.sum().to(torch.float64) - diag_yy).clamp_min(1.0)
```

```
        Den_xy = V_xy.sum().to(torch.float64).clamp_min(1.0)
    else:
        Den_xx = torch.tensor(float(M * (M - 1)), dtype=torch.float64, device=X.device)
        Den_yy = torch.tensor(float(N * (N - 1)), dtype=torch.float64, device=Y.device)
        Den_xy = torch.tensor(float(M * N), dtype=torch.float64, device=X.device)

    XX = (S_xx_all.to(torch.float64) - diag_xx) / Den_xx
    YY = (S_yy_all.to(torch.float64) - diag_yy) / Den_yy
    XY = S_xy.to(torch.float64) / Den_xy

    return float((XX + YY - 2.0 * XY).clamp_min(0))

def _masked_pairwise_sqdist_mean(
    X: torch.Tensor, Mx: torch.Tensor,
    Y: torch.Tensor, My: torch.Tensor,
    eps: float = 1e-9,
    rescale_by_d: bool = True,
) -> tuple[torch.Tensor, torch.Tensor]:
    """
    Masked *mean* squared distance per pair using matmuls only.
    Returns:
      D2:(M,N)  masked mean of (x - y)^2 over overlapping features
      V :(M,N)  0/1 validity (1 if any overlap, else 0)
    """
    assert X.shape == Mx.shape and Y.shape == My.shape
    assert X.shape[1] == Y.shape[1], "feature dims must match"
    d = X.shape[1]
    scale = float(d) if rescale_by_d else 1.0

    # Overlap counts k_ij = sum_t m_i,t * m_j,t
    k = Mx @ My.T  # (M,N)

    # Numerator: sum_t m_i m_j (x^2 + y^2 - 2xy) via matmuls
    Sx2 = Mx * (X * X)  # (M,d)
    Sy2 = My * (Y * Y)  # (N,d)
    SX = Mx * X  # (M,d)
    SY = My * Y  # (N,d)

    num = (Sx2 @ My.T) + (Mx @ Sy2.T) - 2.0 * (SX @ SY.T)  # (M,N)
    D2 = (num / (k + eps)).clamp_min(0) * scale  # (M,N)

    V = (k > 0).to(X.dtype)  # (M,N) in {0,1}
    return D2, V
```

### C.4.2 Masked Gamma Median Heuristic

```python
def gamma_from_masked_median_torch(
    Z: torch.Tensor,
    M: torch.Tensor,
    max_samples: int = 4000,
    g: torch.Generator | None = None,
    eps: float = 1e-9,
    rescale_by_d: bool = True,
) -> float:
    """
    Masked median heuristic using the SAME masked distance as mmd_torch_masked.
    """
    assert Z.shape == M.shape
    n, d = Z.shape
    m = min(n, max_samples)
    idx = torch.randperm(n, device=Z.device, generator=g)[:m]
    S, Sm = Z.index_select(0, idx), M.index_select(0, idx)  # (m,d), (m,d)

    D2, V = _masked_pairwise_sqdist_mean(
        S, Sm, S, Sm, eps=eps, rescale_by_d=rescale_by_d
    )

    iu = torch.triu(torch.ones(m, m, dtype=torch.bool, device=Z.device), diagonal=1)
    keep = iu & (V > 0)
    vals = D2[keep]
    vals = vals[vals > 0]
    if vals.numel() == 0:
        return 1e-6
    med = vals.sqrt().median().item()
    sigma2 = med * med if med > 0 else 1e-6
    return float(1.0 / (2.0 * sigma2))
```