National AI Student Challenge 2026

Technical Report

Presented by

Iizen Sng Lik Tiah

Joe Ong Teng Kiat

Travis Ang Zhi Yan

Pan Yu Hung Shaun

Brennen Goy Tang Jung

April 2026

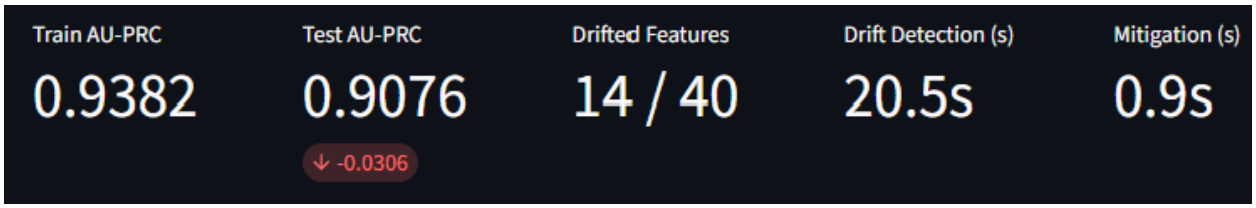
1.0 Introduction

Standard telecom models fail when deployed against non-stationary data. This hackathon forces exactly that challenge: we train on 70,430 labelled rows spanning January to October 2025 but are evaluated against a hidden test set of up to 10 million rows, with column names that may differ and distributional characteristics that are known to shift.

This phenomenon, also known as **Data Drift**, occurs when the statistical properties of inference-time data diverge from those of training data, silently collapsing model performance. Drift manifests as:

1. **Covariate Drift** (a shift in P(X)), or/and
2. **Concept Drift** (a shift in P(Y|X)), or/and
3. **Label Drift** (a shift in P(Y))

Each of these drifts require a different corrective strategy.



(Fig. 1: Dashboard Header Metrics Bar)

Our solution treats drift as a first-class design constraint, organised into five tightly-coupled stages:

This detailed and intrinsic report covers drift detection and mitigation, preprocessing and feature engineering, model training, and a monitoring dashboard (bonus).

The result is a data drift pipeline which achieves a 0.939 AU-PRC on the training set and ~0.908 on the test set, with a ~32-second drift-handling runtime well within the challenge's time budget.

2.0 Data Understanding & Exploratory Analysis

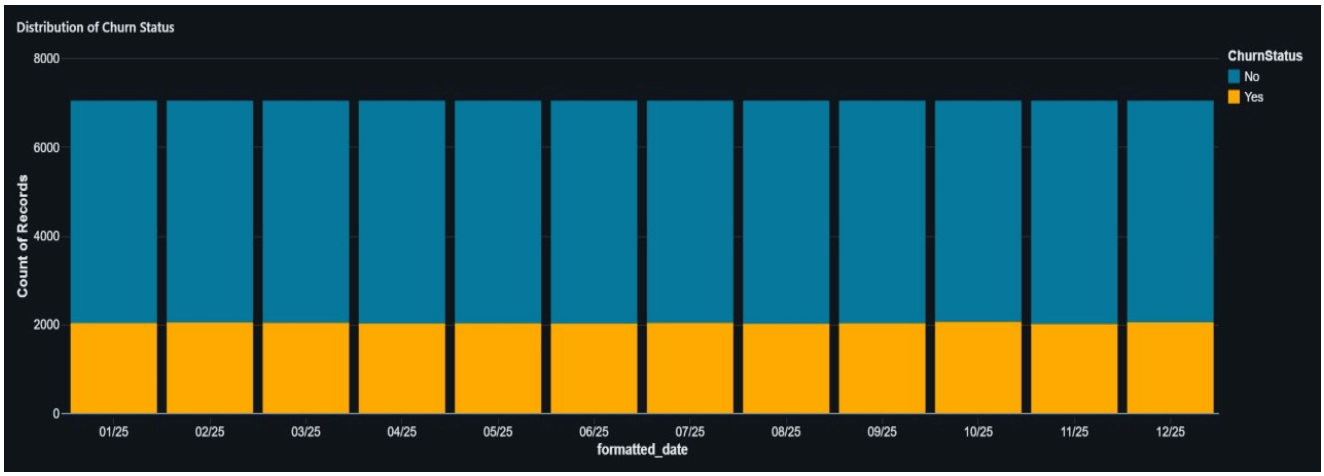
The public training dataset contains approximately 70,430 customer records spanning ten months across January 2025 to October 2025 of Singtel’s subscriber base.

Each row represents a single customer snapshot with 40 features covering demographics (age, gender, marital status, number of dependents), geographic information (city, latitude, longitude,

area code), account attributes (tenure, contract type, payment method, paperless billing), service subscriptions (phone, internet, streaming, security, device protection), and financial metrics (monthly charges, total charges, total revenue, refunds, extra data charges, long-distance charges).

The binary target variable indicates whether the customer churned during the observation period. The test set contains 14,086 rows spanning November - December 2025 and is unlabelled. The challenge specification notes that the hidden evaluation dataset (up to 10 million rows) may exhibit different feature names and distributional characteristics from the public training data, making robustness to data drift a first-order design concern.

2.1 Class Distribution



(Fig. 2: Stacked Bar Chart of monthly distribution of churn status across both train and test datasets)

The stacked bar chart in fig. 2 exhibits a moderate class imbalance: approximately 71.2% of training customers are classified as non-churners and 28.8% as churners (overall churn rate of 0.2879).

This imbalance motivates two design decisions for our evaluation:

1. The use of LightGBM's is_unbalance: True parameter to automatically reweight the loss function
2. The adoption of **Area Under the Precision-Recall Curve** (AU-PRC) as the primary evaluation metric rather than AUC-ROC.

Despite this, the churn rate is remarkably stable across training months, ranging from 0.2852 (August) to 0.2918 (October), with a maximum deviation of only 0.39 percentage points from the global mean, which indicates no target drift within the training period.

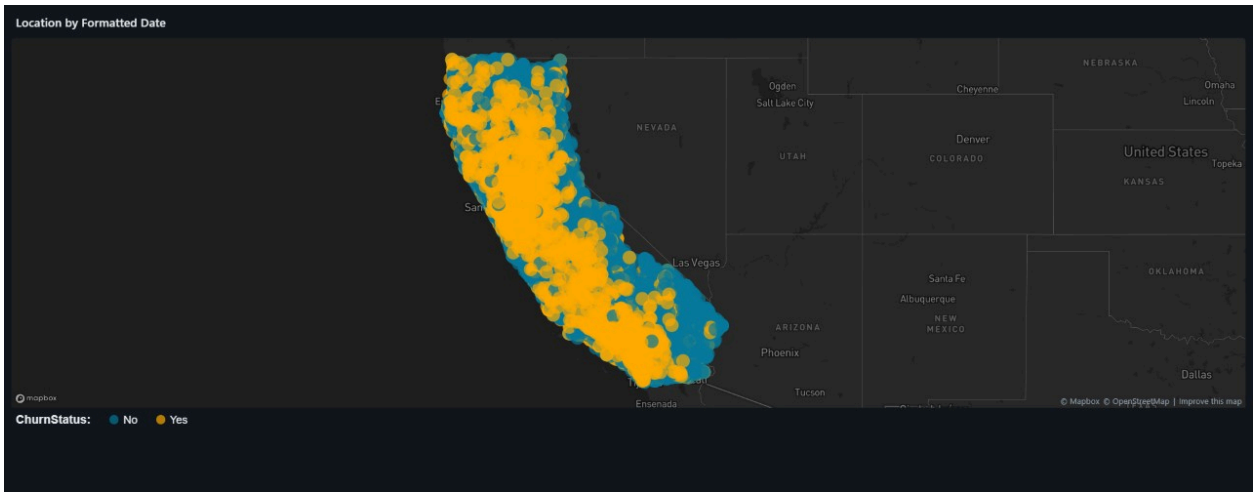
2.2 Feature Types and Quality

Of the 40 features, 23 are categorical (including contract type, payment method, internet type, and binary service subscription flags) and 17 are numerical (including tenure, monthly charges, total charges, and geographic coordinates). Initial inspection revealed several data quality issues. Constant and near-zero-variance features. Two columns, **Country** and **State**, contain a single unique value across the entire training set and are automatically detected and dropped. VoiceService was identified as near-zero variance (97% of customers share the same value) and is likewise removed. Semantically meaningful nulls. Standard imputation would fill missing values with the column mode. However, a null in Offer does not mean the customer probably had Offer A. Instead, it means the customer received no promotional offer, a meaningfully different state that correlates with higher churn risk. These are filled with **explicit semantic labels** (further explained in Section 5.5) before any standard imputation runs.

This highlights missing-pattern drift. As shown, two features exhibit significant null rate shifts:

1. **PrioritySupport**'s null rate increases from a training average of 17.6% to 29.3% in the test set (delta = 11.7%).
2. **DigitalInvoicing** moves in the **opposite** direction, from 17.3% null in training to 0% null in the test set. Binary missing-value indicator flags are added for both features.

2.3 High-Cardinality and Geographic Features



(Fig. 3: Geographic Scatter Map of Customer Locations coloured by Churn Status)

LocationCity contains over 6,000 unique values and is handled through frequency encoding to avoid the memory bloat of one-hot encoding. Meanwhile, raw continuous coordinates (Latitude, Longitude) and **AreaCode** are retained as numerical inputs. Rather than dropping them, they are passed through the pipeline's non-linear QuantileTransformer, compressing their heavy-tailed distributions and allowing LightGBM to safely learn broad regional clusters without hardcoding specific geographical strings.

2.4 Temporal Structure

The training data spans ten months with 7,043 records each (perfectly balanced volume). Standard label encoding would imply a spurious ordinal relationship and encourage memorisation of city-specific patterns.

Exploratory analysis of per-month feature distributions revealed that several key features drift substantially across months. **AvgMonthlyLongDistanceCharges** exhibits a cyclical pattern with PSI exceeding 0.68 in alternating months (Feb, May, Jun, Nov, Dec), whilst **MonthlyCharge** shows a step shift beginning in July (PSI > 0.30 from July onward).

Contract type proportions shift from August onward. These temporal patterns directly motivate the drift detection framework described in the following sections.

3.0 Drift Detection Framework

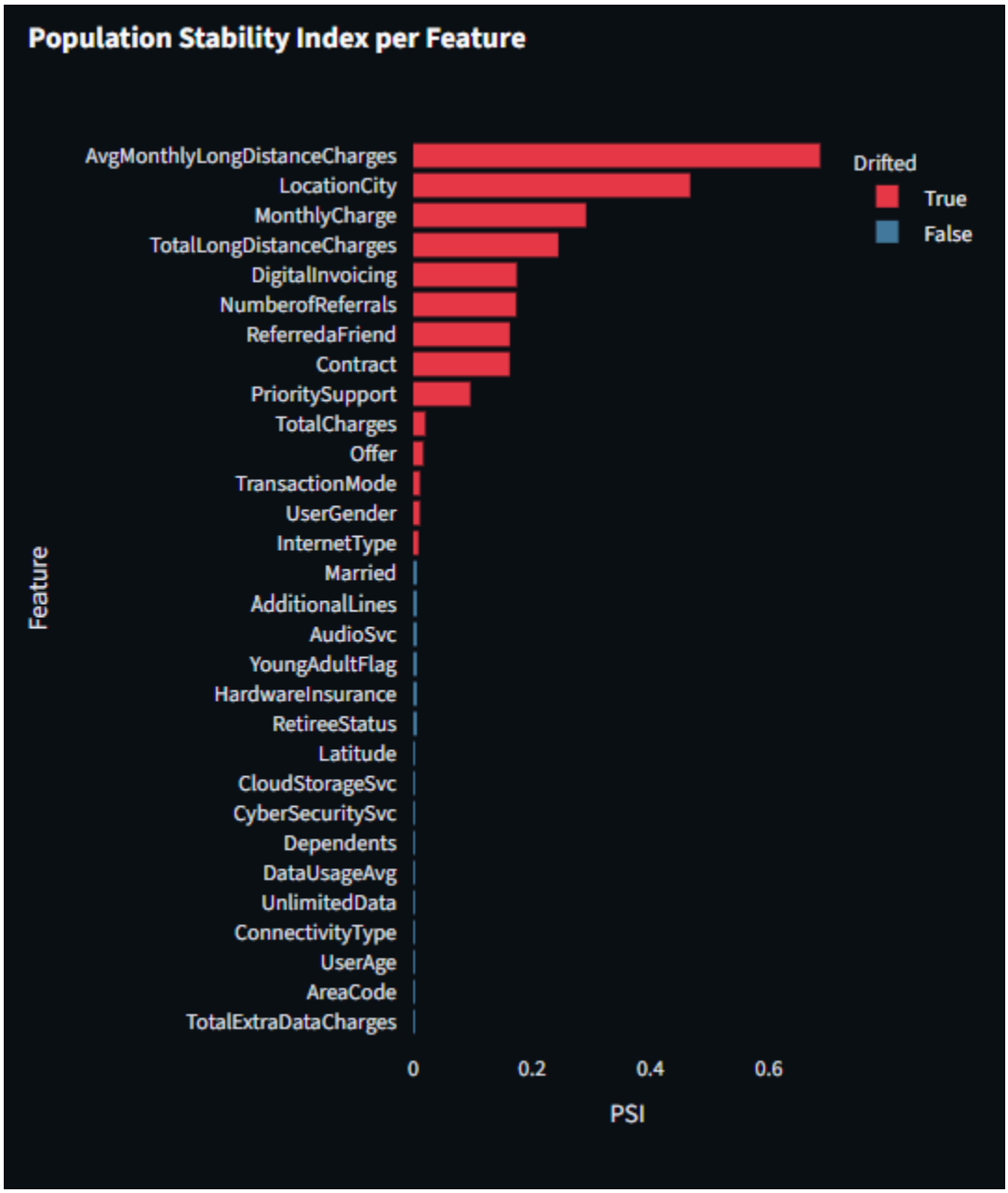
In this challenge, the hidden test set is explicitly stated to differ from the public training set, making drift detection a prerequisite for reliable prediction.

The detection framework operates at two complementary levels:

1. **Univariate Detection:** examines each feature independently, while
2. **Multivariate Detection:** examines the joint feature space.

Together, these produce a comprehensive drift report that directly informs mitigation.

3.1 Univariate Drift Detection



(Fig. 4: Horizontal bar chart of Population Stability Index per Feature)

Univariate drift detection compares each feature's distribution in the training set against the test set using three complementary statistical tests.

At the same time, the detector identified 14 features with statistically significant train-versus-test drift.

3.1.1 Population Stability Index (PSI)

PSI quantifies how much a feature's distribution has shifted between two periods by computing the sum of the weighted log-ratio of bin proportions:

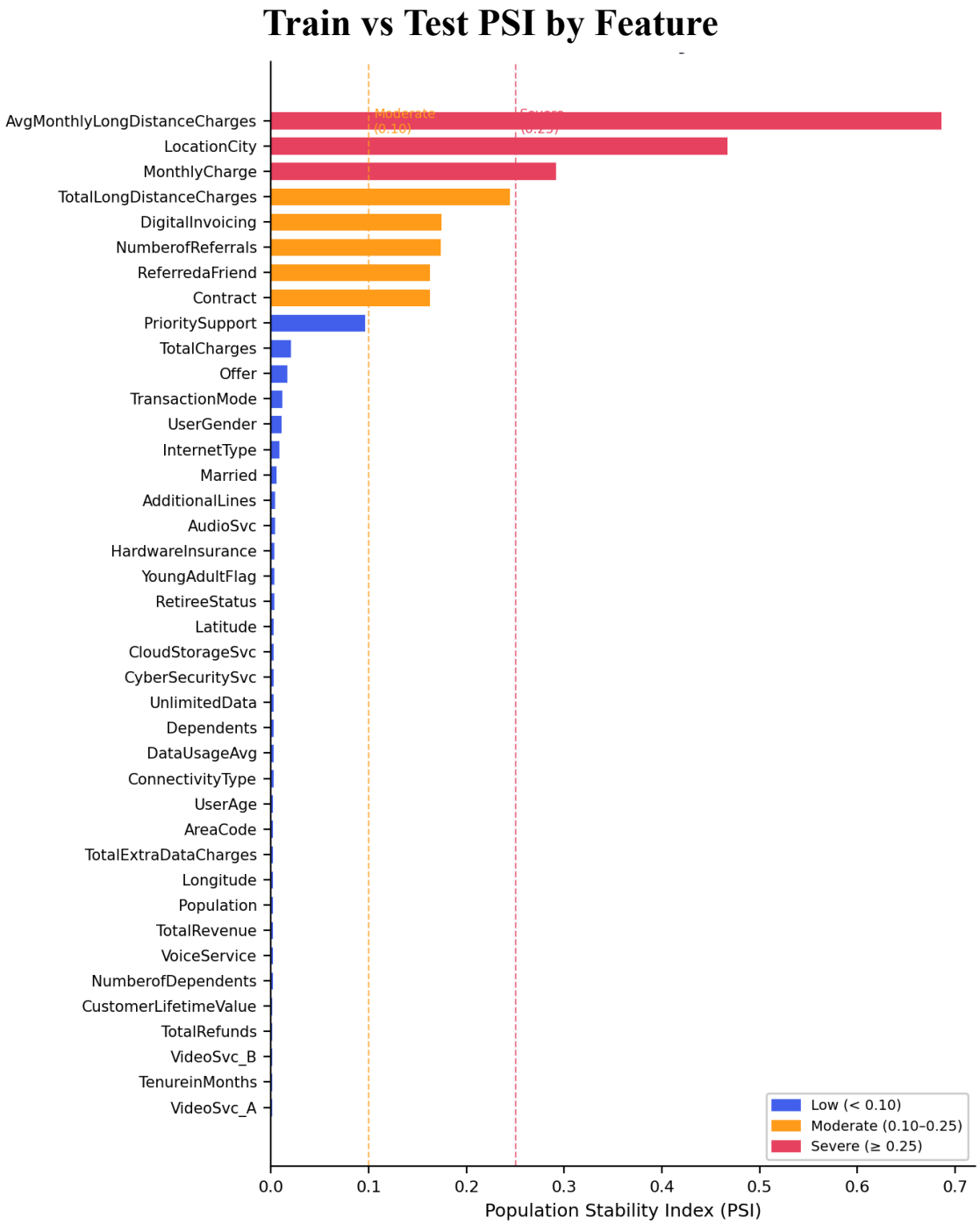
PSI formula:

$$PSI = \sum (P_{test,i} - P_{train,i}) \times In(\frac{P_{test,i}}{P_{train,i}})$$

PSI severity thresholds guide the response: values below 0.10 indicate negligible drift, 0.10 to 0.25 indicate moderate drift, and values above 0.25 indicate severe drift. PSI serves a dual purpose in this pipeline: it flags features for investigation and directly parameterises the proximity reweighting decay rate in the mitigation pipeline (Section 4.1.1).

PSI Range	Severity	Interpretation
< 0.10	Low	Negligible drift; distributions are effectively stable
0.10 - 0.25	Moderate	Noticeable shift; warrants monitoring and investigation
≥ 0.25	Severe	Significant distributional changes; active mitigation required

(Table 5: PSI Severity Thresholds)



(Fig. 6: Train vs Test PSI Chart for all features)

Dashed lines mark the moderate and severe thresholds

3.1.2 Kolmogorov–Smirnov Test (Numerical Features)

For numerical features, the two-sample KS test measures the maximum absolute difference between the empirical cumulative distribution functions:

KS-test formula:

$$D = max | F_{train}(x) - F_{test}(x) |$$

The KS test is distribution-free and captures shifts in location, scale, or shape. It complements PSI by providing a hypothesis test with p-values: features with $p < 0.05$ are flagged as statistically significantly shifted. All numerical features exceeding PSI 0.10 also show KS p-values of 0.0, confirming that the observed shifts are not due to sampling variability.

3.1.3 Chi-Squared / Total Variation Distance (Categorical Features)

For categorical features, the total variation distance (TVD) and chi-squared test of independence determine whether category frequency distributions have shifted. Features where the dominant category flipped between train and test — such as ReferredaFriend (TVD = 0.170) and PrioritySupport (TVD = 0.218, majority class switch) — are particularly impactful because they reverse the direction of the signal the model learned.

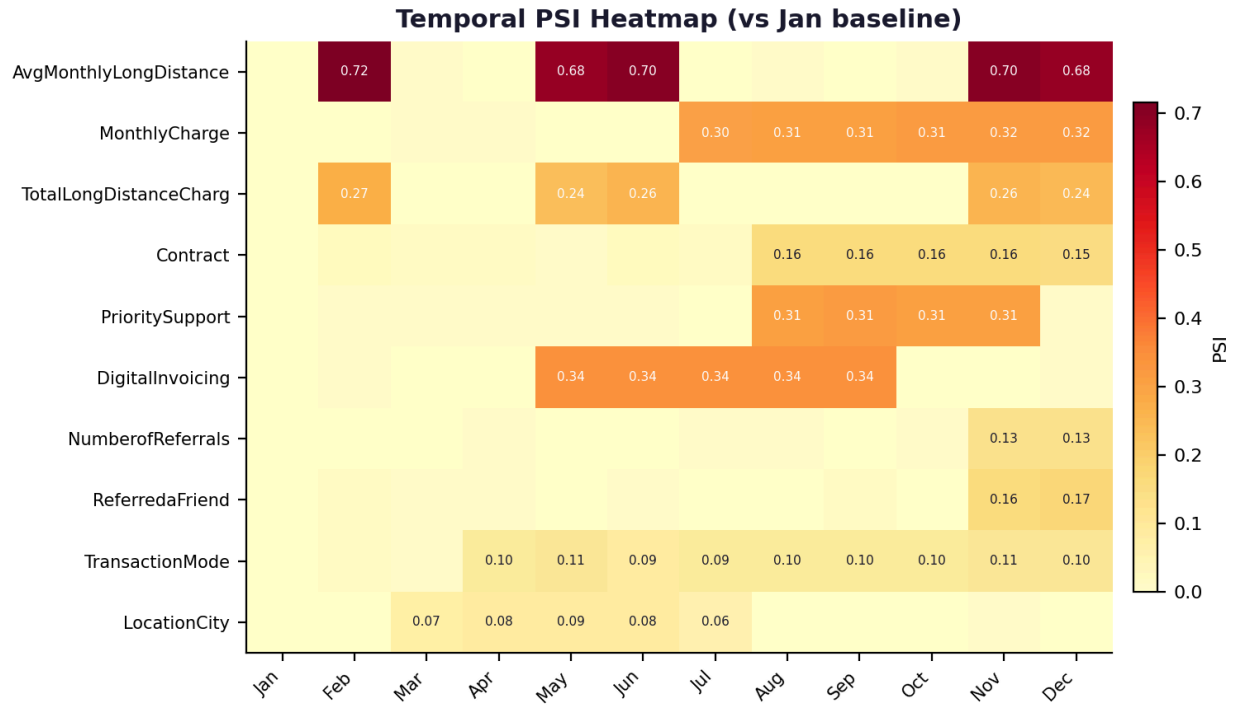
3.1.4 Temporal Pairwise Analysis

Beyond the aggregate train-versus-test comparison, PSI is computed for each feature against the January baseline across all months. Figure 2 shows the temporal PSI heatmap for the ten most volatile features. Several distinct drift patterns emerge.

For Cyclical Drift: AvgMonthlyLongDistanceCharges and **TotalLongDistanceCharges** alternate between high-drift and low-drift months (Feb/May/Jun/Nov/Dec are high; Mar/Apr/Jul–Oct are low), suggesting a periodic process such as seasonal pricing changes.

Step Shifts: MonthlyCharge jumps from near-zero PSI to > 0.30 starting July and remains elevated. Contract proportions shift from August onward. **PrioritySupport** undergoes a majority class switch in August–November.

Persistent Drift: TransactionMode drifts consistently from April onward (PSI 0.087–0.110 every month).

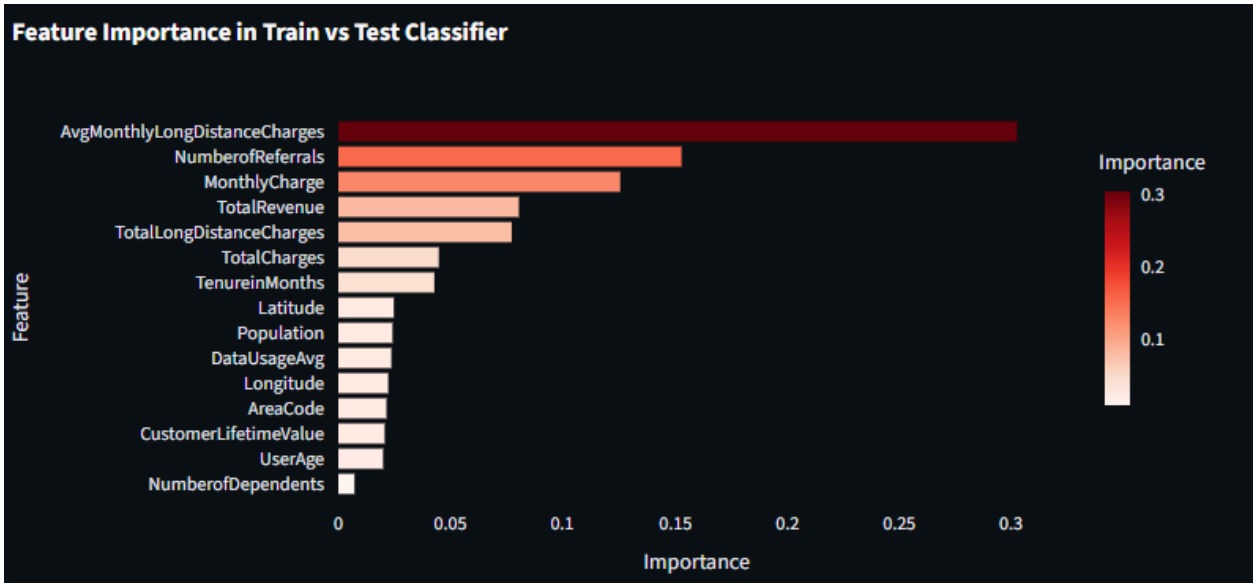


(Fig. 7: Temporal PSI heatmap for the ten most volatile features (vs January baseline)
Darker cells indicate stronger distributional divergence

3.2 Multivariate Drift Detection

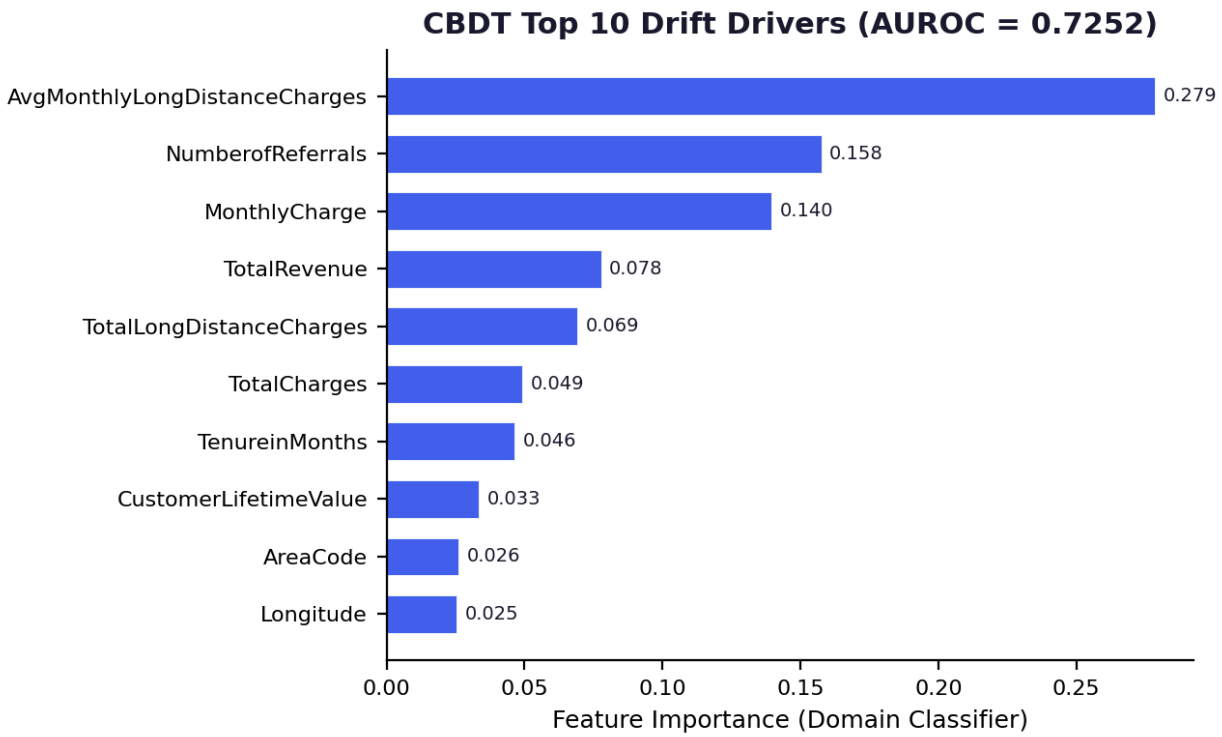
Univariate tests detect shifts in individual features but are structurally blind to changes in the relationships between features. The multivariate detection layer addresses this gap through four complementary techniques. Three of the four tests detect significant drift (vote count: 3/4), confirming that the distributional shift is not limited to individual features.

3.2.1 Classifier-Based Drift Test (CBDT)



(Fig. 8: CBDT domain classifier feature importance (Horizontal Bar Chart))

A Random Forest is trained on a balanced 50,000-row subsample to distinguish training rows from test rows. The resulting AUROC of 0.7252 confirms moderate but non-trivial distributional divergence (H-divergence = 0.4505). The per-feature importance scores identify the specific features driving the separation.



(Fig. 9: CBDT domain classifier top 10 drift drivers by feature importance. *AUROC = 0.7252*)

From Fig. 9, we can observe that **AvgMonthlyLongDistanceCharges** is the dominating drift driver, followed by **NumberofReferrals** and **MonthlyCharge.** The CBDT serves a dual role: in detection, it quantifies drift severity and identifies drivers; in mitigation, the same architecture estimates per-row density ratios for covariate shift correction, and the top drivers seed the z-score and ratio feature engineering

3.2.2 Maximum Mean Discrepancy (MMD)

MMD measures the distance between the mean embeddings of the train and test distributions under a Gaussian kernel (σ, sigma = 5.674). The test returns a score of 0.00823, which is statistically significant (confirms there is drift), confirming that the two distributions occupy measurably different regions of the feature space.

3.2.3 PCA-Based Distributional Comparison

PCA is applied to the combined train-test feature matrix, and KS tests are run on each principal component's projections. Of 16 components tested, 14 show statistically significant drift,

collectively accounting for 98.7% of total explained variance. This indicates that the distributional shift is pervasive rather than concentrated in one or two directions.

3.2.4 Covariance Structure Analysis

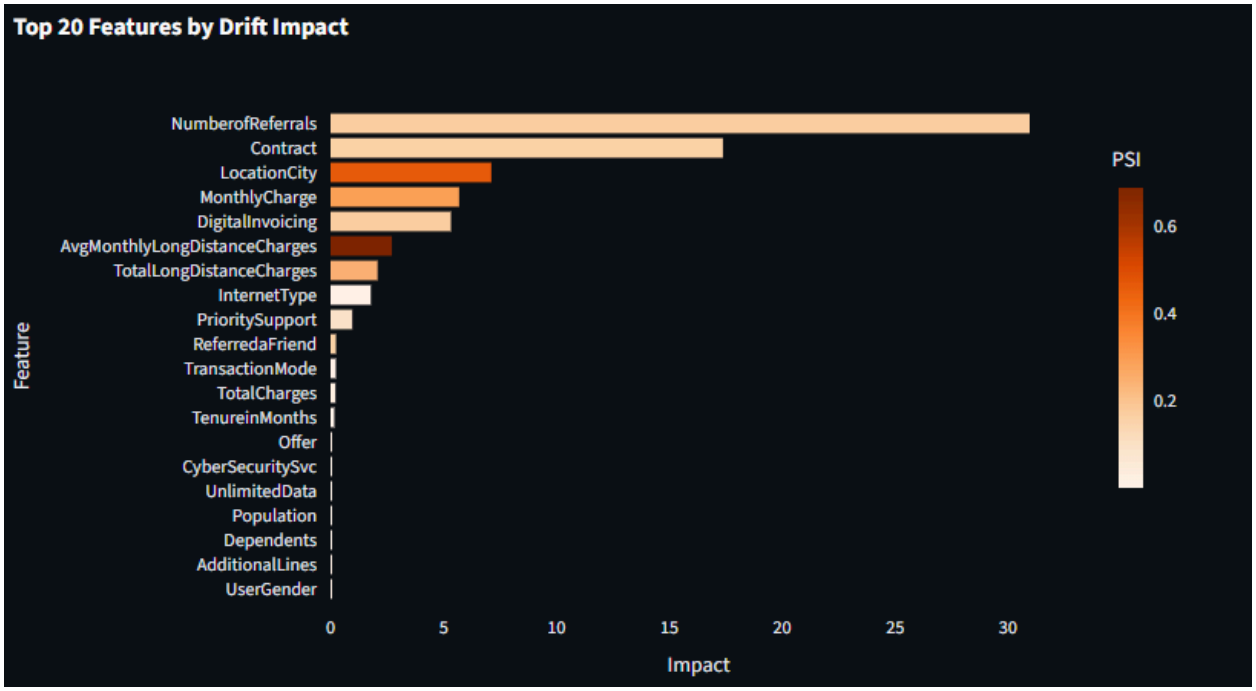
The Frobenius norm of the covariance matrix difference is 0.696 (normalised: 0.041).

This is the only multivariate test that does not flag drift, indicating that while marginal distributions and overall joint structure have shifted, the pairwise correlation relationships between features remain largely preserved. This is a favourable result, which means the model's learned feature interactions are likely to transfer, and provided marginal shifts are corrected through reweighting.

3.3 Impact-Weighted Drift Scoring

However, not all drift is equally harmful. The detection framework computes an impact score for each feature by multiplying its PSI by its LightGBM gain importance:

$$Impact\ Score\ =\ PSI\ \times\ Gain\ Importance$$



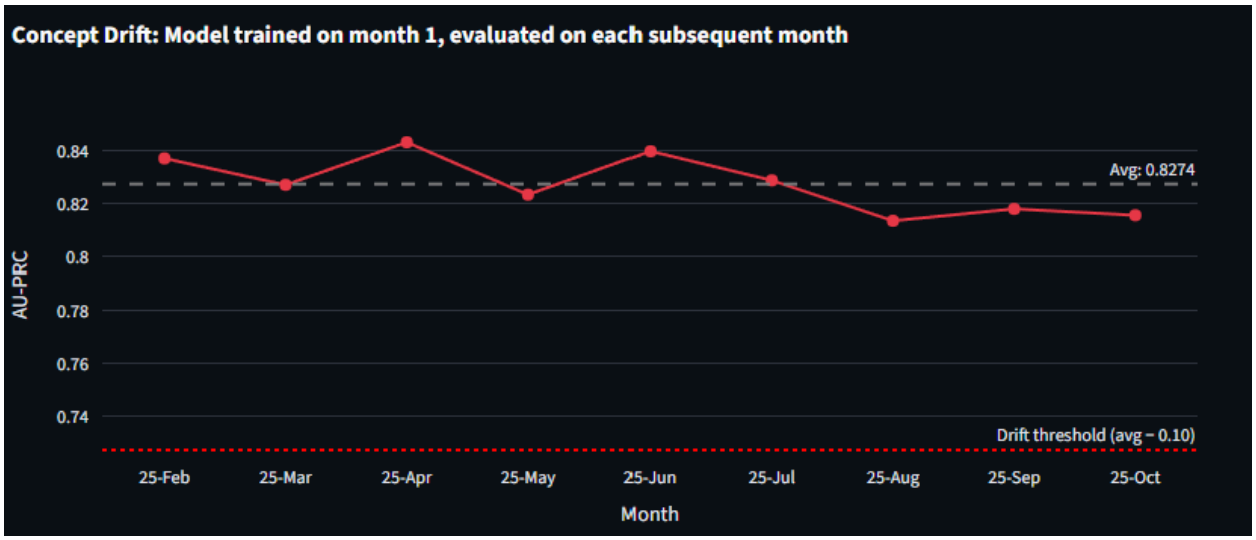
(Fig. 10: Horizontal bar chart showing Drift Impact Scores for top 20 features)

Darker colour shows higher PSI severity

As shown in Fig 7, it reveals the true risk landscape. ***Contract*** type is highlighted as having the highest impact score (42.5) despite only moderate PSI (0.163), because it is the single most important feature in the model (gain importance = 0.261).

Conversely, ***AvgMonthlyLongDistanceCharges*** has the highest PSI (0.686) but relatively low model reliance (gain 0.004), yielding a moderate impact score of 3.0.

3.4 Concept Drift Detection

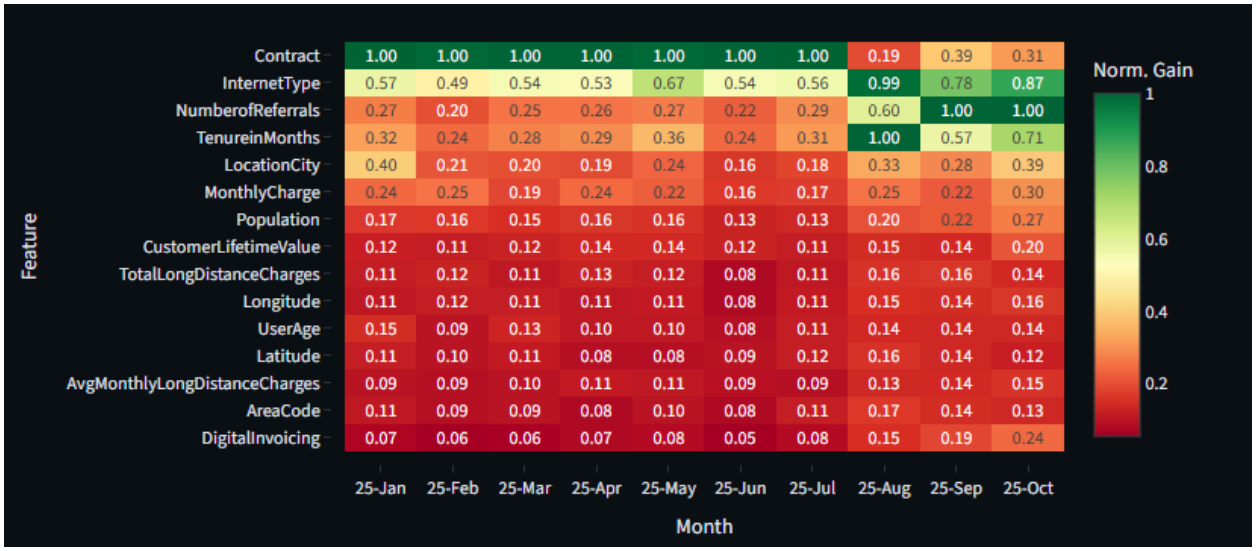


(Fig. 11 : Concept Drift Line Chart across 25-Feb to 25-Oct)

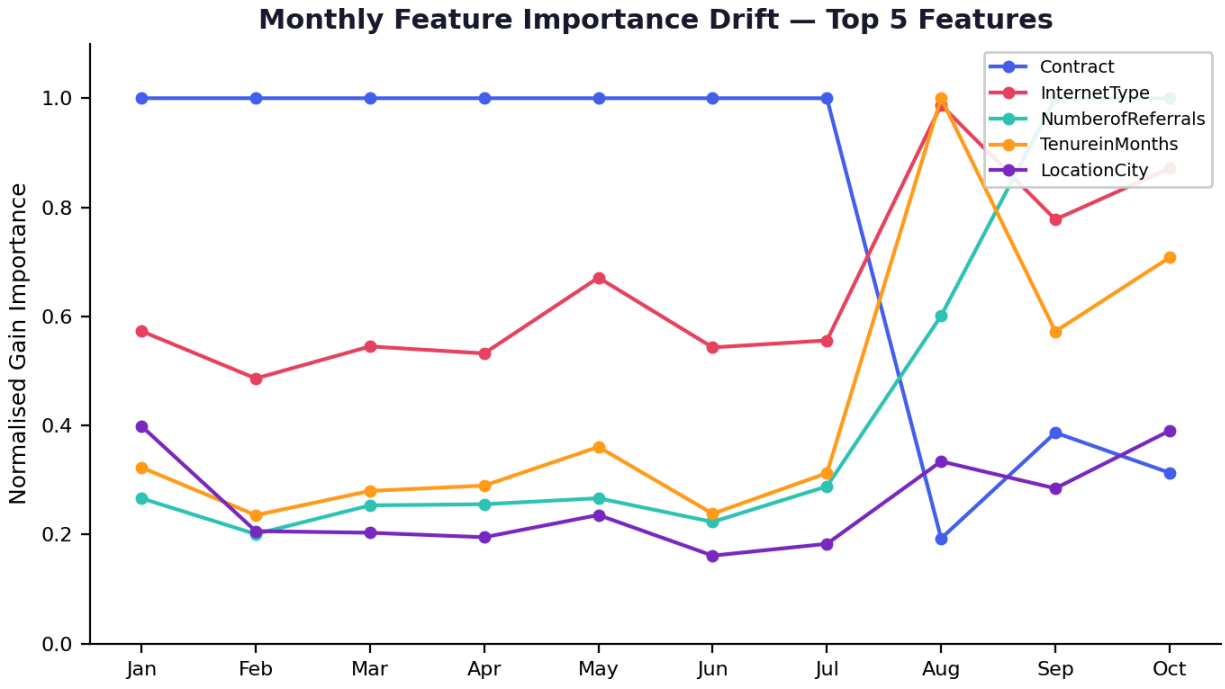
Concept drift refers to a change in P(Y|X). It requires separate detection from covariate drift. Using the leave-one-month-out evaluation trains LightGBM on all months except one and evaluates on the held-out month.

The line chart in Fig. 11 tracks AU-PRC score from a leave-one-month-out evaluation, where the model is trained on all months except one and tested on the held-out month. All months remain well above the drift threshold (0.724), confirming that the relationship between features and churn is stable. It also confirms the drift is **covariate only**. This shows that the relationship between features and churn remains stable. This is a critical finding because it validates that sample reweighting through correction of P(X) is the appropriate mitigation strategy.

3.5 Feature Importance Stability Across Months



(Fig. 12: Monthly Feature Importance Heatmap)



(Fig. 13: Monthly normalised gain importance for the top 5 features)

Lastly, Fig. 13 tracks the normalised gain importance of the top five features across training months. From the chart, we can observe that Contract maintains a normalised gain importance of 1.0 up until the month of August where there is a sharp decline.

This does not indicate concept drift because AU-PRC score remains stable, but rather, it reflects changing marginal distributions where as proportions shift, the model redistributes its reliance to other predictive features. This underscores the importance of correcting for the covariate shift rather than pruning the affected features.

4.0 Drift Mitigation Framework

4.1 Sample Reweighting

To correct for the distributional shift between the training and test sets, sample weights are computed for every training row and passed directly to LightGBM during training. The weighting pipeline combines five independent components.

4.1.1 Proximity & Categorical Reweighting

For each drifted numerical feature, training rows are weighted by their proximity to the test set median, scaled by PSI severity. Proximity is modelled as an exponential decay over the normalised distance from the test median, with the decay rate set to -1.0 for low drift ($PSI < 0.1$), -1.5 for moderate drift ($PSI < 0.25$), and -2.0 for severe drift ($PSI \geq 0.25$). This makes rows closer to the test median receive proportionally higher weights. Categorical features are reweighted by the ratio of test-to-train category frequencies, upweighting categories that appear more frequently in the test set, with ratios clipped to [0.5, 2.0] to prevent extreme adjustments.

4.1.2 Temporal & Concept Drift Weighting

Training data spans multiple months. Months whose feature distributions more closely resemble the test period are upweighted using an exponential decay function: $w_m = \exp(-3 \cdot avg\_PSI\_m)$, where avg_PSI_m is the average PSI between month m and the test set. Separately, the drift detector produces a leave-one-month-out AU-PRC for each training month by training on all other months and evaluating on the held-out month; this provides a per-month measure of concept stability. Each month is weighted by the ratio of its AU-PRC to the overall average, clipped to [0.1, 2.0], reducing the influence of periods where churn behaviour was fundamentally different. Both components are clipped to prevent extreme values.

4.1.3 CBDT Density-Ratio Correction

A Random Forest domain classifier is trained to distinguish training rows from test rows. Random Forest is used for its well-calibrated probability estimates at moderate depth, avoiding the overconfidence that deeper models tend to produce. The predicted probability $\hat{p}(\text{test} \mid x)$ is used to compute the importance weight $w(x) = \hat{p}(\text{test} \mid x) / \hat{p}(\text{train} \mid x)$ for each training row. This is the theoretically optimal correction for covariate shift, directly estimating how much each training sample should be upweighted to match the test distribution. The classifier is trained on a 50,000-row subsample for efficiency, with predictions applied to the full training set in batches.

4.1.4 Label & Volume Drift Correction

Training months do not always share the same class distribution or record volume as the overall dataset. For each month whose churn rate deviates more than 5% from the global prior, positives and negatives are reweighted independently so the month's effective class distribution matches the global one: pos_weight = overall_churn / month_churn, neg_weight = (1 − overall_churn) / (1 − month_churn). Separately, months whose record count deviates more than 20% from the monthly average are scaled by avg_records / count, preventing high-volume months from disproportionately dominating training. Both corrections are applied only to flagged months and clipped to prevent extreme values.

The five weight components — proximity/categorical alignment, temporal similarity, CBDT density-ratio, concept drift, and label/volume correction are multiplied together, clipped to [0.05, 20.0], log-compressed via log(1 + w), and normalised to a mean of 1.0 before being passed to the model.

4.2 Feature Augmentation

In the previous section, we explained how Sample Reweighting corrects for drift by teaching the model which training rows to trust more. Feature Augmentation takes a complementary approach. It reshapes the feature space itself so that the inputs the model sees are inherently less sensitive to distributional shift. Rather than fighting drift at training time, drift-robust features sidestep it by design, expressing customer behaviour in terms that remain meaningful regardless of when the data was collected.

4.2.1 Z-Score, Ratio & Missing Indicator Features

Beyond sample reweighting, the feature space is augmented with drift-robust features applied consistently to both training and test sets. These are engineered to be insensitive to distributional shift, converting drifting absolute values into stable relative ones.

For the top drift drivers identified by the CBDT domain classifier (AvgMonthlyLongDistanceCharges, NumberofReferrals, MonthlyCharge, TotalRevenue, TotalLongDistanceCharges), within-month z-scores are computed using per-month means and standard deviations derived from the training set alone and applied to both splits. This removes month-specific scale effects while preserving each customer's relative position within their cohort.

For the same drivers, ratio features are constructed by dividing each by a stable anchor feature (defined as any feature with PSI < 0.10), with up to two anchors selected per driver. Ratios capture relationships that survive distributional shift: if pricing rises uniformly across the board, the ratio of long-distance charges to total revenue stays constant even as the numerator and denominator both drift.

Finally, for features whose null rate shifts significantly between periods, binary missing-value indicator flags are added. PrioritySupport's null rate rose from 17.6% in training to 29.3% in test, while DigitalInvoicing's fell from 17.3% to 0%. These flags let the model learn whether missingness itself is predictive under the test distribution rather than treating the shift as noise.

In total, 13 new features (5 z-scores, 8 ratios) and 2 missing indicators were added, expanding the feature set from 44 to 59 columns before preprocessing.

5.0 Preprocessing & Feature Engineering

5.1 Design Philosophy

We know that every transformation relies solely on training data. As such, we built the preprocessing pipeline with two strict rules: zero data leakage and extreme scalability. Additionally, a secondary constraint drove the architectural choices: the pipeline must scale to 10 million rows without memory fragmentation or computational bottlenecks. All operations are therefore vectorised using NumPy and Pandas' native categorical machinery, with no Python-level loops over individual rows.

5.2 The QuantileTransformer

To handle the heavy-tailed distributions typical of financial data (e.g., TotalCharges) and protect against absolute scale shifts in the hidden dataset, we apply a non-linear QuantileTransformer (1,000 quantiles, Gaussian output) to all numerical features. Effectively, this helps compress outliers and provides LightGBM with stable information-gain split points.

5.3 Data Normalisation & Pruning

All categorical string values are standardised before any feature engineering begins. For example, whitespace is stripped, values are lowercased, and hyphens and underscores are collapsed to spaces. This ensures that inconsistent representations introduced by data entry, such as "Bank-Withdrawal", "bank_withdrawal", and "bank withdrawal", are all treated as a single category throughout the pipeline. This effectively prevents double cardinality and corrupt downstream features. This normalisation is thus applied over unique values only (not over every row), giving a significant speedup on large datasets.

(Fig. 14: Stacked Bar Chart of Contract Type Distribution across months)

As demonstrated in the raw distribution above, failing to normalise categorical strings can cause identical semantic states (Month-to-Month vs month-to-month, One Year vs one year), to be treated as distinct variables. This artificially doubles the feature's cardinality and splits the predictive signal, which is why we enforce strict string normalisation at load time.

Constant columns (only one unique value across the entire training set) are also automatically detected and dropped before any modelling begins. In the dataset, Country and State were hence identified and removed at this stage.

5.4 Zero-Variance Feature Removal

VoiceService was identified as a near-zero variance feature, as 97% of all customers share the same value. A feature with this distribution contributes no discriminative signal as the model cannot learn any meaningful split, but does add noise that can distort other feature interactions. As such, it is dropped in-place at the start of _add_features, before any other transformations.

5.5 Semantic Null Handling

Standard imputation fills nulls with the column mode, which was the most common observed value. This is statistically safe but incorrect for several columns in this dataset. For instance, a null in Offer does not mean "the customer probably had Offer A". Instead, it means the customer had no promotional offer at all, which is a meaningfully different state that correlates with higher churn risk.

To preserve this signal, null values in the following columns are filled with explicit semantic labels before the standard imputation step runs:

Column	Null Fill	Rationale
Offer	“No Offer”	Absence of promotion is a distinct customer state
InternetType	“No Internet”	Customer has no internet service
ConnectivityType	“No Connection”	Customer has no connectivity product

(Table 15: Semantic null handling for categorical features)

This ensures the model can learn "customers with No Offer churn at rate X" as a genuine pattern rather than treating those rows as noise.

5.6 Engineered Features

All engineered features are ratio-based rather than absolute. To enforce robustness to distribution shift, we came to a conclusion: if Singtel raises prices between the training period and the hidden test period, MonthlyCharge = $85 means something different in each period. But MonthlyCharge / HistoricalAverage = 1.4 (40% above the customer's own average) means the same thing regardless of when it is measured.

Tenure-normalised totals: Every column whose name contains "total" is divided by tenure (clipped to a minimum of 1 month) to produce a per-month rate. This converts cumulative lifetime figures into rates that are comparable across customers of different tenures.

Charge deviation (Bill Shock): The customer's historical average monthly charge is computed as TotalCharges / TenureinMonths. The difference between the current MonthlyCharge and this historical average is the fe_charge_deviation feature. A large positive value signals a sudden price increase, which is one of the strongest known predictors of telecom churn.

Refund rate: TotalRefunds / TotalRevenue captures the proportion of lifetime spend that was returned to the customer. A high refund rate signals repeated dissatisfaction and service disputes.

Extra and long-distance charge ratios: Extra data charges and long-distance charges are expressed as fractions of total revenue. These ratio features capture the degree to which a customer is being penalised by overage fees, independent of their absolute spend level.

CLV-to-revenue ratio: Customer lifetime value expressed relative to actual total revenue. Divergence between these two figures can indicate customers whose predicted value differs significantly from their revealed behaviour.

Referral rate: Referral rate is the number of referrals divided by tenure. A customer who referred many friends early in their tenure demonstrates strong brand affinity and is less likely to churn.

Referral interaction: Referral interaction refers to a multiplicative interaction between the ReferredaFriend binary flag and NumberofReferrals. This captures the joint signal, where a customer who both referred a friend and has multiple referrals is categorically different from one who referred once. This feature relies on our upstream global string normalisation to seamlessly handle any casing inconsistencies, allowing for a fast, vectorised boolean check without row-level string manipulation.

Service count: All binary yes/no service columns are automatically detected by inspecting their unique value sets (fitted on a 10,000-row sample for efficiency). The count of active services is summed into fe_service_count, capturing ecosystem lock-in: customers with many bundled services face higher switching costs and lower churn rates.

Cyclical month encoding: The integer month_idx is encoded as sine and cosine components (fe_month_sin, fe_month_cos). This preserves the cyclical continuity of time (e.g. December and January are adjacent), which a raw integer encoding would destroy.

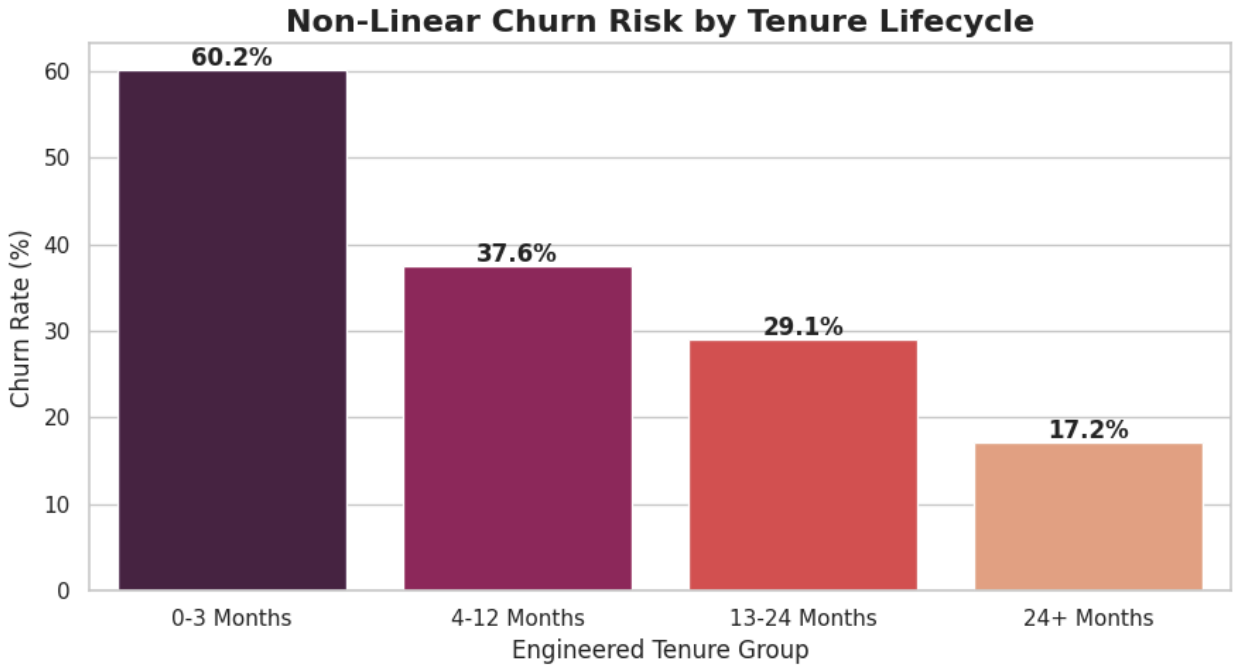
5.7 Tenure Lifecycle Binning

Continuous tenure in months is transformed into four lifecycle segments using np.searchsorted against hardcoded breakpoints [3, 12, 24]:

Segment	Range	Behavioural Meaning
Trial	0-3 months	Trial Period - highest churn risk
Evaluation	4-12 months	Evaluation Period - moderate churn risk
Established	13-24 months	Established Period - lower churn risk
Loyal	24+ months	Loyal - lowest churn risk

(Table 16: Tenure lifecycle bins aligned with contract renewal cycles)

The breakpoints were chosen to align with contract renewal cycles (monthly, annual, biennial) rather than statistical quantiles. Because churn risk is discontinuous at these boundaries, it does not change smoothly with each additional month but spikes at contract expiry points. This domain knowledge is preserved by the manual bin selection. **np.searchsorted** executes this transformation in a single vectorised operation with no row-level loops.



(Fig. 17: Non-Linear Churn Risk Chart categorised by Tenure Lifecycle)

Leaving tenure as a continuous numerical feature forces a tree-based model to approximate a smooth decay curve, which contradicts the telecom reality. This data hence validates our decision to deploy np.searchsorted binning. As the chart shows, churn risk is fundamentally discontinuous, spiking aggressively to 60.2% during the 0-3-month trial phase before drastically dropping off in later periods. By engineering explicit categorical lifecycle phases, we allow LightGBM to split cleanly on these high-risk temporal boundaries rather than fighting the non-linear curve.

5.8 High-Cardinality Geographic Encoding

LocationCity contains over 6,000 unique values. Standard label encoding assigns an arbitrary integer to each city, which implies a spurious ordinal relationship (City 3000 is not "more" than City 1) and causes the model to memorise city-specific patterns that will not generalise to the hidden test set, which may contain entirely different cities.

The solution is frequency encoding. Each city is replaced with its normalised frequency in the training set, which is the proportion of training rows that belong to that city. This is fitted on training data only and applied to both train and test. Cities that do not appear in training are assigned a frequency of 0. This approach:

- Eliminates the spurious ordinal relationship
- Scales to any number of unique cities without memory growth
- Degrades gracefully for unseen cities rather than crashing

The frequency map is stored in eng_params during the fit=True pass and reused during fit=False, ensuring no test distribution information influences the encoding.

5.9 Categorical Encoding

All remaining categorical columns are encoded using pd.Categorical with .cat.codes. This is a fully vectorised operation as there are no Python-level loops over individual rows. Unseen categories in the test set (categories present in test but not in train) automatically receive a code of -1, which LightGBM handles natively without requiring special treatment. The category mapping is stored during the fit=True pass and applied during fit=False, maintaining strict separation between train and test.

LightGBM is notified of which feature indices correspond to categorical columns via cat_feature_indices, enabling its native categorical split algorithm. This is superior to treating encoded integers as ordinals, where LightGBM finds the optimal grouping of categories at each split rather than assuming that adjacent integers correspond to similar categories.

5.10 Missing Value Imputation

Where numerical nulls are filled with the column median, categorical nulls are filled with the column mode. Both fill values are computed in a single pass over the training data and stored for application to test, maintaining the strict no-leakage discipline.

5.11 Column Detection by Keyword

All column lookups in the pipeline use keyword matching against lowercased column names rather than exact string comparisons. For example, the tenure column is found by searching for any column whose lowercased name contains "tenure", catching TenureinMonths, tenure_months, AccountAgeMonths, or any other variant the hidden dataset might use. This makes the pipeline structurally robust to column renaming without requiring any code changes.

6.0 Model Training

6.1 LightGBM with Fixed Hyperparameters

Per the hackathon rules, the model is LightGBM v4.6.0 with the following fixed hyperparameters:

- objective: binary
- verbosity: -1
- is_unbalance: True
- random_state: 42
- importance_type: gain

is_unbalance: True instructs LightGBM to automatically reweight the loss function to compensate for the class imbalance (approximately 2.5:1 non-churn to churn in the training set). This is mathematically equivalent to upweighting the minority class without requiring explicit oversampling.

importance_type: gain measures feature importance by the total information gain contributed by each feature across all trees. This is more meaningful than split count for identifying which features are genuinely predictive versus which are merely used frequently for minor splits.

6.2 Drift-Corrected Sample Weighting

The drift mitigator computes per-row sample weights for every training row. These weights combine three signals: univariate proximity reweighting (upweighting rows whose feature values resemble the test distribution), temporal similarity weighting (upweighting months whose overall distribution resembles the test period), and CBDT density-ratio estimation (a domain classifier that estimates p(test|x) / p(train|x) to correct for covariate shift). The combined weights are passed to LightGBM's sample_weight parameter, causing the model to prioritise training examples that resemble the test distribution it will face at inference time.

6.3 Automated Zero-Importance Feature Pruning

After the initial training pass, each feature's information gain is then inspected. Features with zero gain where LightGBM was never used for any split in any tree are identified and dropped. The model is then immediately retrained on the reduced feature set, resulting in a self-healing mechanism that removes noisy correlates that consume model capacity without contributing predictive power. Hence, improving generalisation on the hidden test set. The categorical feature index list is then updated to reflect the new feature positions after pruning. Once the initial 'clean' model is established and pruned, we transition into the adaptive phase of the pipeline to exploit the unlabelled test distribution through iterative pseudo-labelling.

7.0 Pseudo-Labelling

After the initial LightGBM model is trained on the drift-corrected training set, an iterative pseudo-labelling loop is run for up to 20 rounds to exploit the unlabelled test data.

7.1 Label Assignment

At each round, the current model scores every test row. Rows with predicted probability above 0.80 are assigned a pseudo-label of 1 (churn), and rows below 0.20 are assigned a label of 0 (no-churn). Rows in the middle band are excluded, ensuring only high-confidence predictions are fed back into training. Pseudo-labelled rows are assigned a sample weight of 3.0, reflecting higher trust relative to the average training row, and are concatenated with the original training set before retraining the model from scratch each round.

7.2 Label Refreshing

At the start of each round after the first, the confidence of all previously accumulated pseudo-labelled rows is re-evaluated using the updated model. Rows whose predicted probability has since fallen into the uncertain middle band are dropped from the buffer, meaning their labels

are no longer considered reliable under the improved model. This prevents stale or incorrect pseudo-labels from compounding across rounds.

7.3 Early Stopping

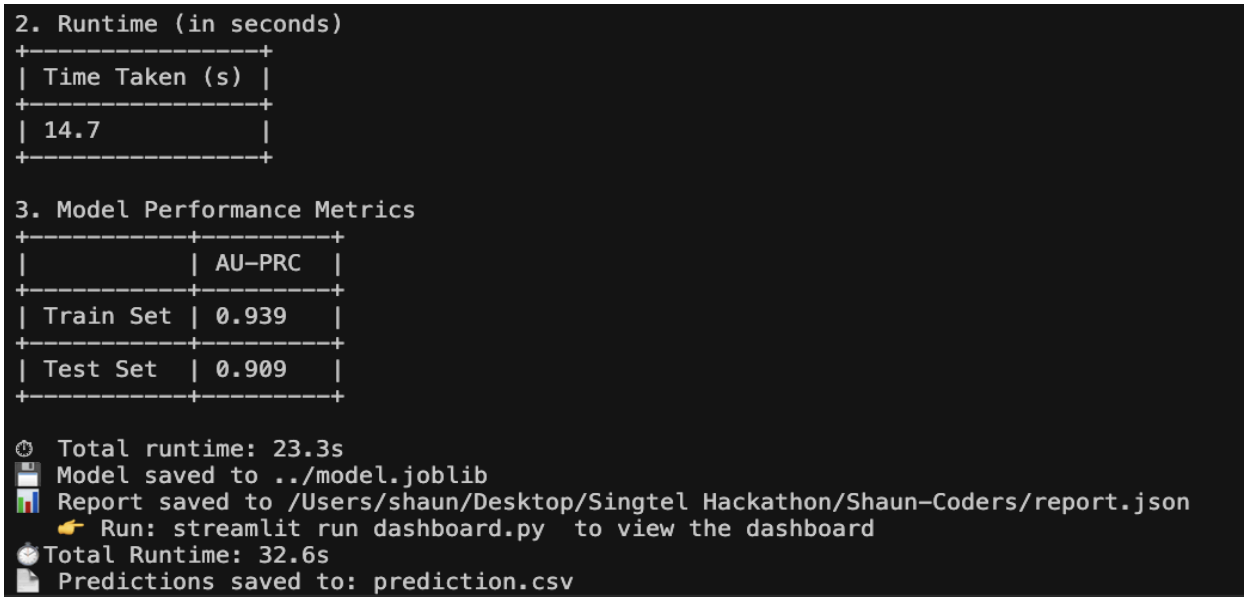
Two stopping conditions are checked each round. First, if no new test rows meet the confidence threshold, the loop terminates immediately. Second, AU-PRC on the test set is monitored after each retraining; if it fails to improve by at least 0.001 for two consecutive rounds, the loop stops early. A hard cap of 300,000 pseudo-labelled rows is also enforced to limit memory usage.

The result is a final model trained on the union of the original drift-corrected training set and all accumulated high-confidence pseudo-labelled test rows, with the pseudo-label weight ensuring the test distribution is adequately represented without overwhelming the supervised signal.

8.0 Results & Evaluation

With our public datasets:
The final AUPRC scores [zero rounds, cold start] were:
On the Trained dataset: **0.923**
On the Test dataset: **0.801**

The final AUPRC Scores [**7 rounds** of Pseudo(early stoppage)] were:
On the trained dataset: **0.939**
On the test dataset: **0.908**



(Fig 18: Pipeline Terminal Output showing Runtime, train & test AU-PRC scores)

To prepare for the large dataset that the pipeline will ingest, we designed the architecture to strictly control memory usage while maintaining a strong AUPRC score.

To prevent the system from running out of memory during massive data ingestion, we shrink the data size immediately by converting numbers to float32 and using pyarrow to load data faster. This cuts the memory footprint in half before any heavy processing even begins.

Fixing "data drift" on huge datasets usually breaks standard computing limits, so our pipeline forces these heavy tasks into small, manageable chunks. For example, our domain classifier learns from a small random sample of data, and then applies what it learned to the rest of the dataset in strict batches.

Furthermore, when the model attempts to guess labels for new, unseen data (pseudo-labeling), we restrict its guesses to a small, dynamic percentage of the total test size. We also force it to stop entirely if the AUPRC score stops improving. This guarantees that as the incoming data scales up, the model throttles its own feedback loop, extracting real patterns without running out of memory or blindly repeating its own mistakes.

8.1 Pipeline Scalability

The pipeline is designed to scale to the anticipated 10 Million Rows hidden test set. This is done through four purposeful architectural choices.

To prevent the system from running out of memory during massive data ingestion, we shrink the data size immediately by converting numbers to float32 and using pyarrow to load data faster. This cuts the memory footprint in half before any heavy processing even begins.

Handling drift on huge datasets usually breaks standard computing limits, so our pipeline forces these heavy tasks into small, manageable chunks. The CBDT domain classifier trains on a small random sample, then applies its predictions to the full dataset in strict batches of 100,000 rows, keeping peak memory flat regardless of input size. All feature engineering uses NumPy and Pandas' categorical machinery with no row-level loops — string normalisation maps over unique values only, and tenure binning runs as a single vectorised operation.

When the model generates pseudo-labels for unseen data, the buffer grows dynamically rather than pre-allocating for the worst case. Confident rows are appended to Python lists per round and only concatenated into a single array at the moment of training, so RAM tracks actual volume rather than maximum capacity. The pseudo-label pool is capped at 3% of the test set size, and training halts automatically if AU-PRC stops improving for two consecutive rounds. This guarantees that as data scales up, the feedback loop throttles itself — extracting real patterns without exhausting memory or compounding its own mistakes.

Finally, all column lookups use keyword matching against lowercased names, so the pipeline adapts automatically if the hidden dataset renames columns such as TenureinMonths, AccountAgeMonths, or similar variants all resolve to the same feature without code changes.

9.0 Discussion

9.1 Key Design Decisions

9.1.1 Multiplicative Weight Combination with Log-Compression

The five sample weight components (proximity/categorical, temporal, CBDT density-ratio, concept drift, and label/volume correction) are multiplied together rather than averaged. Multiplication allows each component to independently correct a distinct axis of drift; a row that is simultaneously far from the test median, from a poorly-behaved month, and from an underrepresented distribution receives compounding downweighting. Log-compression via log(1 + w) then prevents any single extreme product from dominating, and final normalisation to mean 1.0 ensures the total gradient signal passed to LightGBM is unchanged in scale.

9.1.2 CBDT Used for Both Detection and Correction

The same Random Forest domain classifier architecture is used both in the drift detector, to identify which features drive train/test separability, and in the mitigator, to estimate density ratios for covariate shift correction. This is deliberate: the CBDT in detection provides top drift-driving features that directly seed the z-score and ratio feature engineering, creating a closed loop where detection informs mitigation.

9.1.3 Pseudo-Labelling Confidence Band and Weight

The 0.80/0.20 confidence thresholds were chosen to admit only high-certainty predictions, keeping the pseudo-label noise floor low. The sample weight of 3.0 assigned to pseudo-labelled rows is intentionally elevated above the normalised training weights to ensure the test distribution is meaningfully represented in each retraining round without overwhelming the supervised signal from the original training set.

9.1.4 Label Refreshing Each Round

Rather than treating accumulated pseudo-labels as fixed, the pipeline re-evaluates their confidence under the updated model at the start of every round. Rows that fall into the uncertain middle band are dropped. This prevents early-round mislabellings, made under a weaker model, from persisting and compounding into later rounds.

9.1.5 Comparison with Alternative Mitigation Approaches

The challenge briefing outlined several standard mitigation strategies, including sliding window retraining, exponential decay weighting, seasonality matching, invariant transformations such as log and robust scaling, delta-based features, binning and discretisation, and drift-based feature pruning. These approaches were considered and partially incorporated, but each addresses only a single axis of drift in isolation. Sliding window and decay weighting address recency but ignore covariate shift in the feature space. Invariant transformations improve robustness to scale changes but do not correct for distributional misalignment between training and test populations. Drift-based feature pruning reduces noise but discards potentially useful information. Our pipeline instead combines multiple targeted corrections simultaneously: density-ratio reweighting for covariate shift, temporal similarity weights for recency, z-score and ratio features for robustness, and pseudo-labelling to directly incorporate the test distribution, resulting in a more comprehensive adaptation to the observed drift patterns.

9.2 Limitations

9.2.1 Feature Name Generalisation on the Hidden Dataset

The public dataset provides fixed, known column names, and the pipeline's feature engineering is built around these. The hidden evaluation dataset, however, is explicitly noted to have feature names that may differ from the public dataset. While the pipeline uses keyword-based matching to locate columns such as tenure, monthly charges, and total charges, there is no guarantee that these patterns will match the naming conventions used in the hidden dataset. In cases where a keyword match fails, the corresponding engineered features are silently skipped rather than raising an error, which could reduce the richness of the feature set on the hidden test.

9.2.2 Pseudo-Labelling Performance Tied to Deployment Context

The pseudo-labelling loop monitors AU-PRC after each round to determine when to stop. In the current setup this is feasible because labelled data is available for evaluation. In a production deployment where test labels are unavailable at inference time, an alternative stopping criterion such as prediction entropy or label distribution stability would be required. The core pseudo-labelling logic remains valid, but the early stopping mechanism would need to be adapted for label-free environments.

9.2.3 Working Within Runtime and Memory Constraints

Several components operate on subsamples rather than the full dataset due to runtime and memory constraints. Drift detection runs on a 300,000 row sample, meaning subtle distributional shifts in the tail of the data may go undetected. The CBDT domain classifier is trained on 50,000 rows, which limits the precision of its density ratio estimates. Each pseudo-labelling round

subsamples 400,000 base training rows rather than fitting on the full set, introducing variance across rounds. The pseudo-label buffer is also hard-capped at 300,000 rows, which on very large datasets may cut off useful signal before the model fully converges. Each of these was a deliberate trade-off to keep the pipeline feasible within the challenge's time budget, but they represent real ceilings on what the mitigation and adaptation steps can achieve.

9.3 Potential Pipeline Enhancements

9.3.1 LightGBM-Based Domain Classifier

The domain classifier used for covariate shift correction is currently a Random Forest, constrained to a subsample as outlined in 9.2.3. Replacing it with a LightGBM classifier would allow training on the full dataset, as LightGBM's histogram-based splitting handles large row counts at a fraction of the memory and time cost of a Random Forest. This is a natural extension given LightGBM is already used elsewhere in the pipeline.

9.3.2 Automated Threshold Tuning for Pseudo-Labelling

The confidence thresholds and pseudo-label weight are currently fixed. A principled approach would be to tune these on a held-out validation set, or to adapt them dynamically across rounds based on observed label distribution stability and model confidence calibration.

9.3.3 Online Drift Monitoring

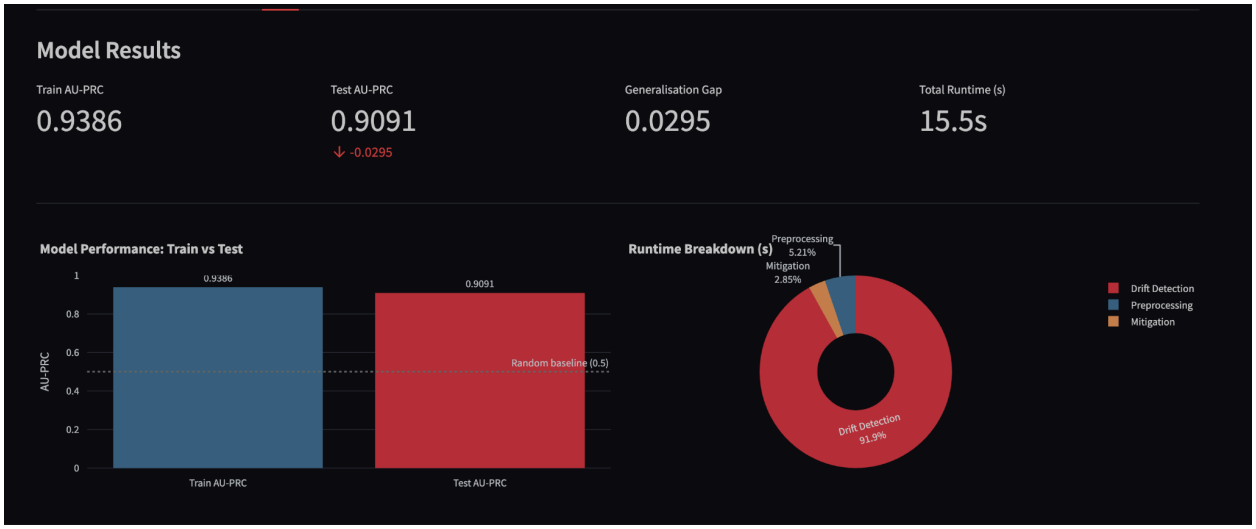
The drift detection step currently runs once at pipeline startup on a fixed sample. In a production setting, extending this to a continuous monitoring framework where feature distributions are tracked over time and reweighting is triggered automatically when drift is detected would make the system more robust to gradual or sudden distribution shifts post-deployment. This aligns directly with the challenge's broader framing of AI models as perishable products requiring continuous monitoring and revalidation.

10.0 Interactive Dashboard (Bonus)

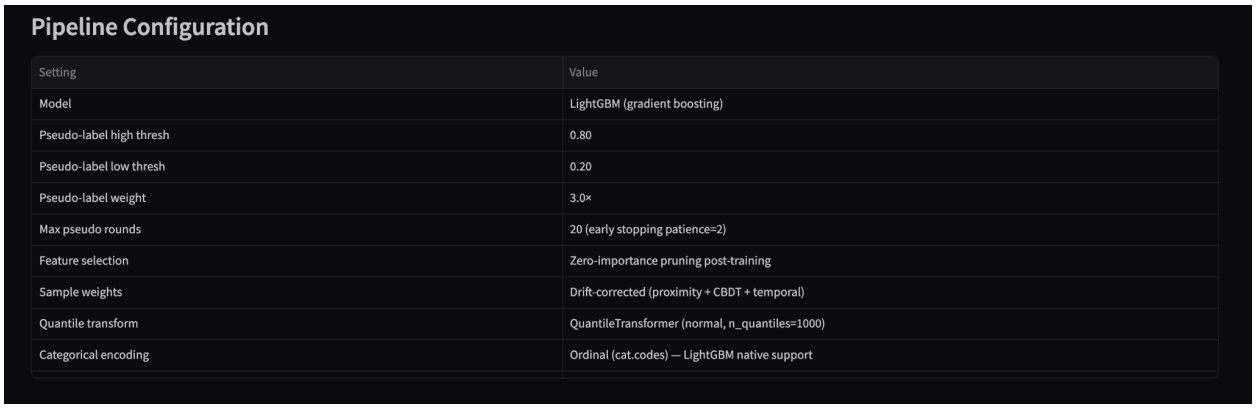
Lastly, for our bonus, we implemented an interactive dashboard built using Streamlit and Plotly. It is launched by running streamlit run dashboard.py.

Instead of running the analysis each time, it reads from a pregenerated report.json produced by the main pipeline, so all visualisations reflect a fixed snapshot of the most recent run.

The dashboard is organised into five tabs. The Data tab provides a dataset overview, including monthly churn trends, missing values, and a comparison between training and test sets. The Drift tab presents per feature PSI scores, CBDT multivariate drift, concept drift decay across months, and a feature importance heatmap. The Mitigation tab explains how each drifted feature is handled, including the reweighting strategies applied. The Preprocessing tab walks through the feature engineering pipeline, covering imputation, encoding, and scaling. Finally, the Results tab summarises performance with Train and Test AU PRC, the generalisation gap, runtime breakdown, and the full pipeline configuration in the figures below.



(Fig 19: Model Results highlighting train, test AU-PRC Score & Run-Time Breakdown)



(Table 20: Pipeline Configuration Summary)

At the top of the dashboard, the key metrics are displayed clearly. The Train AU-PRC is 0.9382 and the Test AU-PRC is 0.9076, alongside the number of drifted features, giving a quick overview of both model performance and the extent of distribution shift.

11.0 Conclusion

With that, we have come to the end of our NAISIC Singtel Hackathon Report. This report presented a comprehensive drift-aware pipeline for telecom churn prediction, spanning detection, mitigation, preprocessing, model training, and monitoring. The core insight driving our approach is that drift is not a single problem but a multi-axis challenge — covariate shift, temporal instability, and feature importance rotation each demand targeted corrections. By combining density-ratio reweighting, drift-robust feature engineering, and iterative pseudo-labelling into a unified pipeline.

This hackathon truly pushed us to think beyond static model training and engage with the reality that production models face, such as data changes, and models must adapt. Building the drift detection framework, designing the mitigation pipeline, and stress-testing scalability gave us hands-on experience with problems that textbooks describe but rarely let you solve end-to-end.

We would like to thank the organisers of the NAISC Singtel 2026 Adaptive Drift Intelligence Challenge for designing a problem that reflects genuine production challenges, and for the opportunity to develop and present this work.