Datenbank-Spektrum
https://doi.org/10.1007/s13222-025-00518-4

FACHBEITRAG

Impact of Preprocessing on Classification Results of Eye-Tracking-Data

Jennifer Landes[1] · **Meike Klettke**[1] · **Sonja Köppl**[2]

Received: 25 May 2025 / Accepted: 31 October 2025

Abstract
Eye-Tracking data provides valuable insights into human behavior, yet its high variability to noise require robust preprocessing to ensure meaningful analysis. This study introduces and evaluates a systematic preprocessing pipeline tailored to enhance machine learning classifier performance in the context of Eye-Tracking data, on a dataset on academic cheating detection. Unlike prior work focusing on isolated preprocessing steps, our approach explores 193 configurations by combining techniques for missing value imputation, outlier handling, normalization, smoothing, feature limiting, and filtering. A Random Forest classifier is used consistently across all configurations due to its robustness and prior success in similar domains. Our results demonstrate that well-designed preprocessing pipelines can substantially improve classification accuracy. Additionally, a feature importance analysis reveals that static spatial and camera-based metrics outperform traditional gaze dynamics in predictive power. This research aims to create a reusable framework for Eye-Tracking data.

Keywords Data Preprocessing · Random Forest · Classification · Eye-Tracking

1 Introduction

Eye-Tracking data has become an invaluable resource for understanding cognitive processes, attentional shifts, and behaviors such as deception in assessments [1]. Despite its potential, the *complexity and noise* inherent in Eye-Tracking data pose significant challenges [2]. Individual variability, data fluctuations, and measurement artifacts necessitate preprocessing techniques to extract meaningful signals and ensure robust machine learning outcomes. Without appropriate handling, *noise in raw Eye-Tracking data* can severely compromise the accuracy of classification models. Despite the widespread use of Eye-Tracking data, preprocessing techniques tailored to its noise-prone and temporally irregular nature are rarely evaluated systematically. So, this paper focuses on the *essential preprocessing steps* required to enhance the accuracy of classification models for Eye-Tracking datasets. The dataset analyzed originates from an experimental study where students' eye movements were recorded during exam-related tasks to identify predictive markers of cheating behavior. Challenges such as high variability, missing values, and outliers in the data necessitate a structured preprocessing approach.

We propose and evaluate a comprehensive preprocessing pipeline addressing key data quality concerns, including managing missing data, outliers, normalizing features to reduce individual variability, and employing smoothing techniques to minimize noise. Various strategies for imputation (mean substitution, Last Observation Carried Forward, and deletion), outlier detection, and feature scaling (z-score normalization, robust, and min-max scaling) are systematically compared. Using Random Forest classifiers, we assess the impact of these preprocessing steps on classification performance, aiming to identify optimal combinations. So, the *Research Questions* are:

- *RQ1:* How do different preprocessing methods, such as handling missing values, normalization, and outlier detection, change the Eye-Tracking data and thus influence the classification accuracy?

✉ Jennifer Landes
jennifer.landes@informatik.uni-regensburg.de

Meike Klettke
meike.klettke@informatik.uni-regensburg.de

Sonja Köppl
sonja.koeppl@hnu.de

[1] Data Engineering Group, University of Regensburg, Regensburg, Germany

[2] Business Informatics, Hochschule Neu-Ulm, Neu-Ulm, Germany

Published online: 20 November 2025

- *RQ2:* Which tested preprocessing configuration yields the most suitable classification performance using a Random Forest on our dataset?

To address these research questions, we design and evaluate a preprocessing pipeline and analyze its impact on classification accuracy. To validate the proposed methods, additional experiments are planned, including a second Eye-Tracking study and analyses of datasets from other domains.

Long-term, this work aims to establish a robust, empirically grounded preprocessing framework for Eye-Tracking data by defining an optimized sequence of preprocessing steps. Building on this, the goal is to create a modular system for systematically testing preprocessing steps and their combined impact on classifier performance, enabling broader applicability to other Eye-Tracking datasets. The dataset analyzed in this study serves as a representative example to guide the development of the framework, which will be further validated on additional datasets to ensure generalizability and the empirical basis of the findings.

Structure *Chapter 1* reviews the prior research done on preprocessing techniques for Eye-Tracking data and classifier comparisons, providing the foundation for this study. *Chapter 2* presents a literature review, focusing on how preprocessing impacts machine learning performance in Eye-Tracking data analysis. *Chapter 3* describes the design of and modularization of the preprocessing pipeline, a feature importance analysis, and the training of the Random Forest classifier. In chapter 4, the different preprocessing combinations are presented, addressing *RQ2* by identifying the optimal configuration for classification performance based on their effect on model accuracy and statistical analysis. Finally, *Chapter 5* offers an conclusion, outlook on future research and limitations, including further validation and generalization of the preprocessing pipeline to other datasets.

Contribution This paper advances the field by:

1. Identification of needs for the development of specialized data engineering algorithms for preprocessing Eye-Tracking data.
2. Design of a comprehensive framework for composing and evaluating data engineering pipelines tailored to Eye-Tracking datasets.
3. Introduction of a methodology to systematically generate, evaluate, and optimize data engineering pipelines to improve analysis.

Preliminary Research The analysis of Eye-Tracking data comprises literature review, experimental design, data collection, preprocessing, and machine learning classification. A *quantitative survey* first captured students' cheating behaviors and contextual factors [2], forming the basis for an Eye-Tracking experiment within the *ii.oo—digital kompetenzorientiert Prüfen* project. The study investigates methods for detecting academic misconduct, a growing concern with the shift to online exams during COVID-19 [3].

Prior work focused on designing preprocessing pipelines for Eye-Tracking data and evaluating classifiers to identify the most effective model for predicting cheating behavior. Random Forest outperformed Support Vector Machines, k-Nearest Neighbors, and Decision Trees, accurately classifying participants into cheating and non-cheating groups. Building on these findings, this work applies the optimized preprocessing pipeline and Random Forest classifier for further analysis [2, 4].

An earlier conference paper [5] presented an initial, simplified implementation, limited to an 80/20 hold-out evaluation and based on a partially preprocessed dataset. The current work employs the latest, refined pipeline, incorporating additional validation schemes (*GroupKFold, Out-of-Bag*) and leveraging the fully cleaned dataset for more robust and generalizable results.

2 Related Work

Preprocessing has a significant impact on machine learning performance, especially with increasing data complexity and volume [6]. For Eye-Tracking data, optimized preprocessing can improve classification outcomes substantially. Burdack et al. [7] demonstrated that filtering and data reduction enhance gait classification with SVMs and CNNs, while normalization has limited effect. Kamiran and Calders [8] showed that preprocessing can also mitigate bias without degrading accuracy.

Preprocessing for Eye-Tracking Data Few studies address preprocessing systematically despite its importance. Olsson [9] emphasized real-time filtering and smoothing to counteract saccadic noise. Adalı et al. [10] used adaptive filters to stabilize pupil signals under variable lighting. Li et al. [11] applied low-pass and Fourier transforms for gaze path smoothing. Kret and Sjak-Shie [12] proposed a full pipeline including interpolation, z-score normalization, and artifact removal. These works underline that preprocessing is essential for improving data quality and classification performance—motivating the systematic pipeline evaluation in this study.

Domain-Specific Techniques Vortmann [13] transformed time series into images to boost CNN classification. Zheng et al. [14] used multichannel preprocessing to treat sen-

sor axes as color layers, improving activity recognition. Mishra [15] applied ensemble preprocessing in chemometrics to enhance robustness, while Gholizadeh [16] found that tailored derivative-based preprocessing improved soil contaminant predictions.

Feature Selection Reducing dimensionality improves performance and prevents overfitting in Eye-Tracking models. Arnold et al. [17] show the relevance of task-specific metrics (e.g., fixation dispersion), and Lim et al. [18] use fixation and pupil features for detecting cognitive load. Vortmann and Putze [19] present that combining handcrafted and learned features enhances accuracy. Selection techniques include recursive elimination, permutation importance, and Random Forests.

3 Design and Evaluation of Preprocessing Pipeline

This chapter presents the systematic preprocessing framework developed to address *RQ1* and *RQ2*. We introduce a modular, extensible methodology for generating, evaluating, and optimizing preprocessing pipelines for time series classification, tailored to Eye-Tracking data. The pipeline covers feature selection, imputation, outlier detection, normalization, smoothing, low-pass filtering, and feature thresholding. Each step supports multiple methods, whose combinations form a comprehensive search space. Classification accuracy serves as the benchmark metric, and the methodology is implemented in Python.

3.1 Preprocessing Pipeline

The preprocessing pipeline methodically prepares Eye-Tracking data for analysis by progressively refining the dataset to ensure reliable results. The aim is to maximize the accuracy of the classification of the eye-trackig data.

Pipeline Design and Step Sequence

- Iterative testing and domain knowledge: Missing values handled first, then outlier removal, Normalization for feature comparability, Smoothing and filtering applied later to reduce noise without distortion.
- This structure improved classification accuracy and data stability. Aligns with recommendations in eye-tracking and time series literature [9, 11, 12], further survey studies support sequence: NaN replacement → outlier detection → imputation → normalization [20]. Automated frameworks like *Preptimize* use similar order to reduce errors ([21]; Fig. 1)

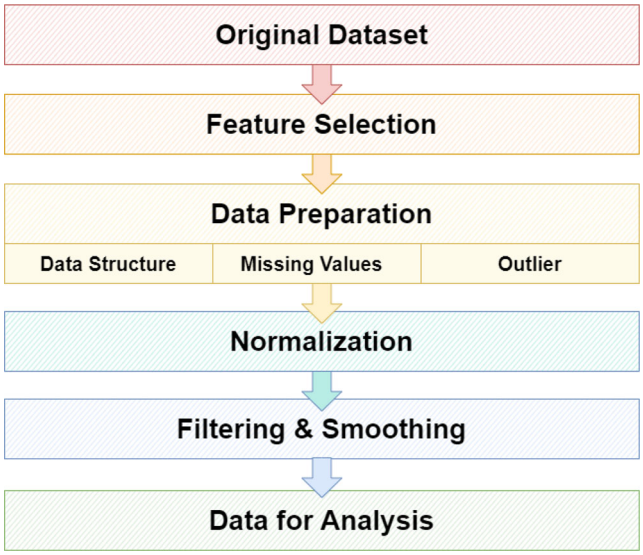

Fig. 1 Preprocessing Pipeline for Eye-Tracking Data

The preprocessing pipeline $\mathcal{P}$ is defined as a sequence of transformations applied to the raw dataset D, producing a processed dataset D': $\mathcal{P} : D \rightarrow D'$. Each feature $f_i \in F = \{f_1, f_2, \ldots, f_m\}$ undergoes a specific sequence of preprocessing steps $\mathcal{P}_i(f_i)$, tailored to its characteristics. For each feature f, the pipeline is described as:

$$\mathcal{P}(f) = P_{\text{limits}} \circ P_{\text{filter}} \circ P_{\text{smooth}} \circ P_{\text{norm}} \circ P_{\text{out}} \circ P_{\text{mv}}(f)$$

Where each part P of the pipeline represents the corresponding preprocessing step:

- P_{limits} for applying feature limits,
- P_{filter} for applying Butterworth filter,
- P_{smooth} for applying smoothing,
- P_{norm} for normalization,
- P_{out} for outlier detection and handling,
- P_{mv} for missing value detection and handling.

The performance $\mathcal{A}$ of $\mathcal{P}_j$ is evaluated by classification accuracy: $\mathcal{A}(\mathcal{P}_j) = \text{Accuracy}(\text{Classifier}(\mathcal{P}_j(F)))$. So, the pipeline $\mathcal{P}^*$ is determined by maximizing accuracy: $\mathcal{P}^* = \max_{\mathcal{P}_j} \mathcal{A}(\mathcal{P}_j)$. The pipeline provides a modular design for Eye-Tracking data.

Feature selection is followed by *Data Preparation*, comprising data structure validation, handling of missing values, and outlier. *Data Structure* checks and removes errors, inconsistencies, and number format issues. *Missing values* are detected and treated per feature [22, 23], while *Outlier Detection* identifies atypical eye movement data points that could distort results [24]. *Normalization* then scales features to a uniform range for balanced influence [12, 25–27]. Optional steps address Eye-Tracking-specific noise: a *Low-Pass Filter* reduces high-frequency fluctuations [10], en-

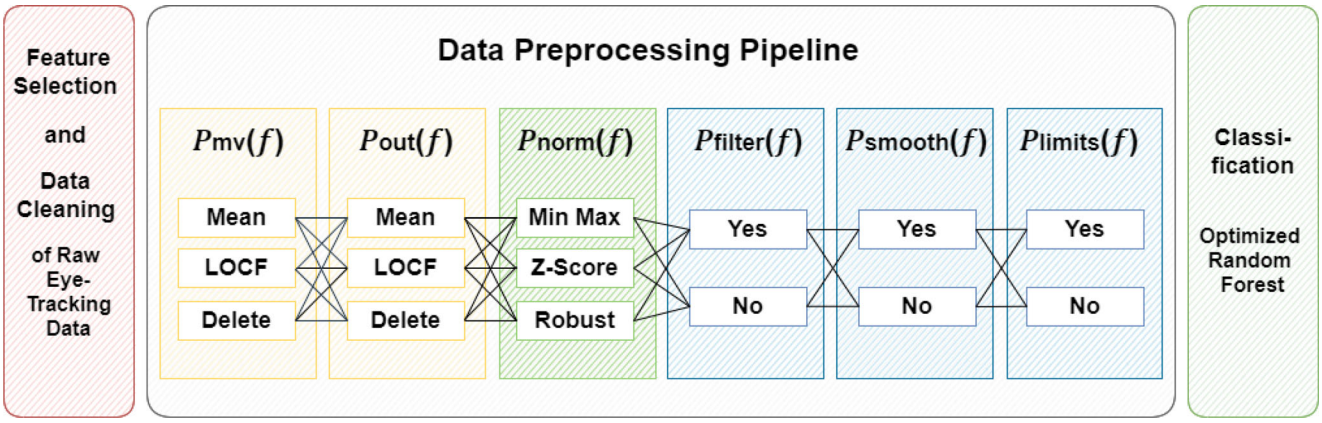

Fig. 2 Search space of preprocessing configurations

hancing quality and consistency [9, 28]; *thresholding* applies min/max limits to selected features; and *Smoothing* via Fourier transformation reduces minor fluctuations to oultine significant patterns [2, 11].

3.2 Modularization of Preprocessing Pipeline

To enhance the accuracy of classification, we systematically test and evaluate each step of the preprocessing pipeline. This involves experimenting with various methods for the preprocessing steps and evaluating their impact on classification performance. The variations of each preprocessing step are displayed in Fig. 2.

The workflow starts with *Feature Selection*. Table 1 present relevant features based on domain knowledge and previous literature [17], with a focus on analyzing gaze behavior, while system-level metrics and metadata, such as PC CPU values, StimType, and Eventsource, were omitted as irrelevant to the classification task. To identify which Eye-Tracking variables are most relevant for predicting behavior, we utilized *Feature Importance*. These scores indicate how much each feature contributes to the classification decisions, based on the reduction of impurity (specifically, Gini impurity) across all trees [2].

The first step in the pipeline involves the treatment of *missing values and outliers*, as these can significantly skew results if not properly managed. The methods to be explored are shown in Table Fig. 3.

Handling missing values (P_{mv}) is defined as:

$$P_{\text{mv}}(f) = \begin{cases} \text{mean}(f), & \text{mean imputation} \\ \text{locf}(f), & \text{LOCF imputation} \\ f \setminus f_{\text{missing}}, & \text{deletion} \end{cases}$$

Outlier Detection and Handling (P_{out}):

Outliers (P_{out}) are identified using z-score:

$f_{\text{out}} = \{x \in f \mid |z(x)| > z_{\text{threshold}}\}$, $z(x) = \frac{x-\mu}{\sigma}$, where μ = mean, σ = standard deviation.

$$P_{\text{out}}(f) = \begin{cases} \text{mean}(f_{\text{out}}), & \text{imputation with mean} \\ \text{locf}(f_{\text{out}}), & \text{imputation with LOCF} \\ f \setminus f_{\text{out}}, & \text{delete outliers} \end{cases}$$

Table 1 Features of Eye-Tracking Dataset.

Features	Description
GazeLeft, GazeRight	X and Y coordinates of left and right eye
PupilLeft, PupilRight	Pupil size of left and right eye
DistanceLeft, DistanceRight	Distance to camera of left and right eye
CameraLeft, CameraRight	Camera coordinates of left and right eye
ValidityLeft, ValidityRight	Validity of gaze data from left and right eye
Gaze X, Y	X and Y coordinates of gaze point
Interpolated Gaze, Distance	Interpolated X and Y coordinates and distance of the gaze
Gaze Velocity, Acceleration	Velocity and acceleration of the gaze point
Saccade X, Y	Rapid eye movements between fixations
Saccade Duration	Time it takes to move from one point to another
Fixation Duration	Time interval from beginning to the end of a fixation
Fixation X, Y	X and Y coordinates of participant's fixation point

Method	Description
Mean	Replace missing values with the mean of the corresponding feature.
LOCF	Impute missing values by carrying forward the last observed value (Last Observation Carried Forward).
Deletion	Remove rows or columns with missing values, especially when the proportion of missing data is small.

Fig. 3 Table 2: Methods for Handling Missing Values and Outliers [22, 24, 29]

Method	Description
Min-Max	Scales data to a fixed range ($[0, 1]$).
Z-Score	Standardizes features by removing the mean and scaling to unit variance, μ = mean and σ = standard deviation.
Robust Scaler	Scales data according to the interquartile range, reducing sensitivity to outliers, IQR is the interquartile range.

Fig. 4 Table 3: Normalization Methods [25, 26]

Next, *feature scaling and normalization* are focussed, essential for ensuring that all features contribute equally to the classification model. Table Fig. 4 summarizes the scaling methods under consideration.

Normalization methods (P_{norm}) include:

$$P_{\text{norm}}(f) = \begin{cases} \frac{f - f_{\min}}{f_{\max} - f_{\min}}, & \text{Min-Max Scaling} \\ \frac{f - \mu}{\sigma}, & \text{Z-score} \\ \frac{f - \text{median}(f)}{\text{IQR}(f)}, & \text{Robust Scaling} \end{cases}$$

Further refinement of the data involves *setting thresholds, applying low-pass filters, and smoothing* [2]. Table Fig. 5 details these techniques.

Smoothing (P_{smooth}) can be applied by using Fourier Transformation:

$$P_{\text{smooth}}(f) = \begin{cases} \mathcal{F}^{-1}(\mathcal{F}(f)), & \text{smoothing} \\ f, & \text{no smoothing} \end{cases}$$

Low-pass filtering (P_{filter}) with Butterworth filter:

$$P_{\text{filter}}(f) = \begin{cases} \text{Butter}(f, \text{cutoff}, \text{order}), & \text{filter} \\ f, & \text{no filter} \end{cases}$$

Feature limits (P_{limits}) ensure data validity:

$$P_{\text{limits}}(f) = \begin{cases} f, & \text{if } f \in [l_{\min}, l_{\max}] \\ \text{NaN}, & \text{if } f \notin [l_{\min}, l_{\max}] \end{cases}$$

Technique	Description
Setting Thresholds on Features	Defines upper and lower limits for each feature to exclude extreme values, where $l_{\min}$ and $l_{\max}$ are bounds.
Low-Pass Filtering	Reduces high-frequency noise, cutoff = frequency, order = Filter steepness.
Fourier Transformation	Smooths data by filtering out high-frequency components, emphasizing underlying eye movement patterns. $\mathcal{F}(f(t)) = \int_{-\infty}^{\infty} f(t)e^{-i2\pi\nu t}dt$

Fig. 5 Table 4: Data Refinement Techniques [9, 11]

3.3 Implementation

To address *RQ1* (impact of individual preprocessing steps) and *RQ2* (optimal pipeline), the implementation systematically tests combinations of preprocessing methods on gaze-tracking data. Each pipeline configuration is evaluated using a `RandomForestClassifier` with three complementary validation strategies.

- **Tools:** `pandas, numpy, scikit-learn` for data handling and modeling
- **Signal Processing:** `scipy.signal` used for filtering operations
- **Data Handling:** CSVs are loaded with encoding handling; all values are converted to numeric (non-numeric → `NaN`)
- **Search Space:** 216 possible pipeline combinations ($3\times3\times3\times2\times2\times2)+1$ baseline → 217. Due to the guard clause in our evaluation code

`(if not data.empty and not data.isnull().values.any():)`, we only trained pipelines whose preprocessed data contained no `NaN`s and at least one valid sample. This led to **192** valid configurations; together with the baseline **193** evaluated setups. The excluded configurations failed because aggressive deletion (missing values, outliers) in combination with screen boundary limits removed all rows, low-pass filtering or FFT-based smoothing introduced `NaN`s on very short segments, or scaling on constant-variance columns produced invalid values.

Classifier Selection

- **Comparison** of k-NN, SVM, Decision Trees, Naive Bayes, Random Forest
- Random Forest showed highest accuracy and robustness, so using exclusively to focus on preprocessing effects.

Experiment and Data Set

- 30 students from Hochschule Neu-Ulm solved 5 different exam tasks, with/without cheating.

```python
# Prepare features and labels
X = df.drop('target', axis=1).apply(pd.to_numeric, errors='coerce').dropna()
y = df['target'].loc[X.index]

# Scale and split
X_scaled = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, stratify=y)

# Train model and extract feature importances
rf = RandomForestClassifier(n_estimators=100).fit(X_train, y_train)
importances = rf.feature_importances_
```

Fig. 6 Listing 1: Random Forest training and feature importance extraction

- **Data:** 150 CSV files (30 students × 5 tasks), collected via iMotions 5.1, standardized and aligned, see https://github.com/eyetracking-data/eyetracking_Cheating.git
- **Labels:** Self-reported indication of cheating.
- **Ethical Considerations:** Written informed consent from all participants, GDPR-compliant, anonymized, securely stored data for research use only, conducted at Hochschule Neu-Ulm (Fig. 6).

Feature Engineering Calculation of feature importance (Listing 1):

- Irrelevant features were removed.
- Dropping Rows/columns with excessive mv.
- Numeric features standardized (z-score).
- 80/20 train-test split.
- Random Forest trained with 100 trees.

As a reference point, a **baseline pipeline** was evaluated that uses the raw gaze coordinates without preprocessing steps. Minimal handling of missing values was applied (row-wise deletion). A Random Forest classifier was trained on features, and its accuracy was assessed using three validation strategies: 80/20 GroupSplit, 5-Fold GroupKFold, and Out-of-Bag (OOB) estimation.

Performance with Preprocessing To address *missing data*, imputation by mean or LOCF or deletion is done (Listing 2, Fig. 7).

Outlier detection is implemented using a z-score threshold of 3. Identified outliers are managed by one of three approaches: replacing them with mean, LOCF, or deleting

```python
def handle_missing_values(df, method):
    if method == 'mean':
        df.fillna(df.mean(), inplace=True)
    elif method == 'locf':
        df.fillna(method='ffill', inplace=True)
    elif method == 'delete':
        df.dropna(inplace=True)
    return df
```

Fig. 7 Listing 2: Function to handle missing values in Eye-Tracking data

rows. The Python function iterates over each column to calculate z-scores, identifies values exceeding the threshold, and applies the chosen method to handle these outlier (Listing 3, Fig. 8).

Data Normalization (Listing 4) is carried out by MinMaxScaler, which scales features to a specified range; RobustScaler, which reduces the influence of outliers by scaling data based on percentiles; or StandardScaler, which standardizes data by centering and scaling it to unit variance (z-score normalization) (Fig. 9).

The last three steps are optional. First, the application of *feature limits*. This ensures that the values of the features `'Gaze X'` and `'Gaze Y'` lie within realistic bounds (dimensions of a screen, from 0 to 1920 for X and 0 to 1080 for Y), filtering out noisy data points. Second, for *data smoothing*, the function applies a Fourier transformation to reduce noise, which stabilizes the data. Third, the pipeline includes the option to apply a *low-pass filter*. This function filters out high-frequency noise from the data using a Butterworth filter, which passes signals below a cutoff frequency and attenuates higher frequencies (Listing 5, Fig. 10).

```python
def handle_outliers(df, method, z_thresh=3):
    z_scores = np.abs((df - df.mean()) / df.std())
    outliers = (z_scores > z_thresh)
    if method == 'mean':
        for col in df.columns:
            df.loc[outliers[col], col] = df[col].mean()
    elif method == 'locf':
        for col in df.columns:
            df.loc[outliers[col], col] = df[col].ffill()
    elif method == 'delete':
        df = df[~outliers.any(axis=1)]
    return df
```

Fig. 8 Listing 3: Handling outliers by z-score method

```
def normalize_data(df, method):
    if method == 'minmax':
        scaler = MinMaxScaler()
    elif method == 'robust':
        scaler = RobustScaler()
    elif method == 'zscore':
        scaler = StandardScaler()
    df[df.columns] = scaler.
        fit_transform(df[df.columns])
    return df
```

Fig. 9 Listing 4: Normalization with scaling methods

```
def apply_feature_limits(df, limits=
    True):
    if limits:
        df = df[(df['Gaze X'] >= 0) &
            (df['Gaze X'] <= 1920)]
        df = df[(df['Gaze Y'] >= 0) &
            (df['Gaze Y'] <= 1080)]
    return df
```

Fig. 10 Listing 5: Apply screen boundary limits

Once preprocessing is complete, a `target` label is assigned to each sample to indicate whether it corresponds to a "cheating" (1) or "non-cheating" (0) scenario, based on participants' self-reports. The `load_and_preprocess_data` function iterates through all CSV files, applies the specified preprocessing steps, and systematically evaluates different *combinations of preprocessing configurations*. This allows us to assess how choices such as missing-value handling, normalization, or filtering affect model performance. For model training, we employ `RandomForest`, an ensemble method that builds multiple decision trees on random subsets of the data and features, then aggregates their predictions (majority voting). To ensure robust and unbiased evaluation, we apply three complementary validation strategies, each designed to estimate generalization performance while avoiding *data leakage* (information from the test set influencing the training process):

1. **80/20 GroupShuffleSplit:** The dataset is split into 80% training and 20% testing data, while keeping all samples from the same participant (group) in the same split. This prevents the model from indirectly "seeing" related samples during training and ensures that the test performance reflects truly unseen participants.
2. **5-Fold GroupKFold Cross-Validation:** The data is divided into 5 folds (partitions) at the group level. In each iteration, 4 folds are used for training and 1 fold for testing, cycling through all folds. The reported accuracy is the mean across all folds.
3. **Out-of-Bag (OOB) Scoring:** When training the Random Forest, each decision tree is built on a bootstrap sample (random sample with replacement) of the training data. On average, about one-third of the samples are *not* used to train a given tree; these are the *out-of-bag* samples. The model predicts these unused samples during training for a fast internal performance estimate without explicit validation set.

While both OOB and explicit validation methods (e.g., 80/20 hold-out, *k*-fold cross-validation) estimate generalization performance, they differ fundamentally. Explicit splits partition the data into separate training and testing sets before training. OOB scoring, is an internal mechanism in bootstrap-based ensembles: each tree is trained on a bootstrap sample, and the unused (*out-of-bag*) samples are used for evaluation. The final OOB score aggregates predictions from all trees where a sample was out-of-bag, providing an efficient built-in performance estimate ([30–32]; Fig. 11).

4 Results and Discussion

This chapter reports the experimental results and their interpretation. A baseline model is established, followed by an analysis of feature importance. Subsequently, multiple preprocessing pipelines are compared across validation methods to identify settings and performance gaps.

4.1 Feature Engineering

The model was trained on eye-tracking features, including gaze coordinates, pupil diameter, and camera-based spatial data. *Table 6* ranks features by importance, especially `CameraLeftY`, `CameraRightY`, and `CameraRightX`, emerging as most influential. These capture head position and posture, with vertical (Y-axis) movements more indicative of cheating-related behavior than horizontal (X-axis) ones. This likely reflects participants looking up or down at off-screen resources. `Interpolated Distance` to the screen also ranked highly, while gaze coordinates (`Gaze X`, `Gaze Y`) were of moderate importance and dynamic measures (`Gaze Velocity`, `Gaze Acceleration`) were negligible. Applying a 0.02 importance threshold suggests excluding the latter to simplify the model without loss of accuracy, underscoring the value of static spatial over rapid dynamic features (Table 2).

Although only `Gaze X` and `Gaze Y` were used in the main classification for simplicity and comparability, the feature importance analysis was exploratory and did not guide feature selection. These results nevertheless indicate that head position and distance to the screen merit further inves-

```
# Base Random Forest model (parallel execution, OOB enabled)
base_clf = RandomForestClassifier(
    n_estimators=200,
    max_features="log2",
    min_samples_leaf=2,
    min_samples_split=2,
    random_state=42,
    n_jobs=-1,
    oob_score=True,
    bootstrap=True
)

# (A) 80/20 GroupShuffleSplit (prevents data leakage)
gss = GroupShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_idx, test_idx in gss.split(X, y, groups):
    base_clf.fit(X.iloc[train_idx], y[train_idx])
    acc_8020 = accuracy_score(y[test_idx], base_clf.predict(X.iloc[test_idx]))

# (B) GroupKFold Cross-Validation (n_splits = 5)
gkf = GroupKFold(n_splits=5)
cv_scores = cross_val_score(base_clf, X, y, groups=groups, cv=gkf, n_jobs=-1)
acc_kfold = cv_scores.mean()

# (C) Out-of-Bag score (bootstrap-based validation)
base_clf.fit(X, y)
acc_oob = base_clf.oob_score_
```

Fig. 11 Listing 6: Random Forest training with three validation strategies

tigation for improving performance or reducing dimensionality, while avoiding bias from using the same algorithm for selection and classification.

4.2 Preprocessing Evaluation

The results presented in this section address *RQ1* by evaluating the effects of individual preprocessing methods and *RQ2* by comparing the performance of various preprocessing configurations.

The **baseline pipeline**, which uses raw gaze coordinates without preprocessing, achieved mean accuracies of 56.67% with 80/20 GroupSplit, 62.00% with 5-Fold GroupKFold (standard deviation: 8.84%), and 63.33% with the OOB estimation.

We selected the *5-Fold GroupKFold* validation for the final analysis because it offers a balanced trade-off between bias and variance. Compared to a single 80/20 split, it provides more reliable performance estimates by averaging across multiple, non-overlapping partitions, while preserving group integrity to prevent data leakage. Although Out-of-Bag (OOB) scoring achieved slightly higher mean accuracy, it is intrinsically tied to the Random Forest algorithm and cannot be directly compared across classifiers. The 5-Fold scheme, in contrast, is model-agnostic and widely accepted as a standard for robust performance evaluation. For comparison, in **table 6** pipelines are grouped into top-10, mid-10, and bottom-10 performers for each validation method. OOB achieves the highest mean accuracy in its top group and maintains more consistent performance across the ranking. In contrast, the 80/20 split has a wider gap between top and bottom pipelines, indicating higher sensitivity to preprocessing choice. Additionally, 95% confidence intervals were computed for the mean accuracies (Table 3) using the standard formula $\pm 1.96 \times \mathrm{SD}/\sqrt{n}$ with $n = 193$. The resulting intervals ($\pm 0.45\%$–$\pm 0.77\%$) confirm that the observed 8.67 pp improvement over the baseline is statistically significant.

Table 2 Feature Importance of Classification

Feature	Import.	Feature	Import.
CameraLeftY	0.1051	Int. Gaze Y	0.0357
CameraRightY	0.1036	Gaze Y	0.0351
CameraRightX	0.1015	GazeLefty	0.0314
Int. Distance	0.0942	GazeRighty	0.0291
CameraLeftX	0.0914	Int. Gaze X	0.0290
DistanceRight	0.0750	Gaze X	0.0277
DistanceLeft	0.0741	GazeLeftx	0.0260
PupilLeft	0.0523	GazeRightx	0.0230
PupilRight	0.0451	Gaze Velocity	0.0118
–		Gaze Acc.	0.0089

Table 3 Compact overview of classification accuracy (%). For each method, the mean accuracies of the 10 best, 10 mid-ranked, and 10 worst pipelines are reported. The two rightmost columns show the overall mean ± SD across all 193 pipelines and the corresponding 95% confidence interval.

Method	Top	Mid	Low	Overall	95% CI [±%]
5-Fold	70.67	64.00	54.80	63.60 ± 3.32	±0.47
80/20	68.00	60.00	46.67	58.63 ± 5.46	±0.77
OOB	73.33	66.00	58.67	65.87 ± 3.16	±0.45

Table 4 Top-5 and worst-5 preprocessing pipelines ranked by mean 5-Fold accuracy. Each pipeline is shown as: Missing/Outlier/Normalize/Limits/Smooth/LowPass.

Top-5 Pipelines	Acc. (%)	Worst-5 Pipelines	Acc. (%)
Mean/locf/minmax/F/F/T	70.67	Locf/delete/minmax/T/F/T	56.67
Mean/locf/minmax/T/F/T	70.67	Locf/locf/minmax/T/T/T	55.33
Delete/mean/robust/F/T/T	69.33	Locf/mean/minmax/T/T/T	54.00
Delete/mean/robust/T/T/T	69.33	Locf/mean/minmax/T/F/T	54.00
Mean/mean/minmax/F/T/T	69.33	Locf/locf/minmax/T/F/T	52.67

Preprocessing Compinations We evaluated $N = 193$ preprocessing pipeline configurations using 5-Fold GroupKFold. Table 4 compares the five highest- and lowest-performing preprocessing configurations. Each tuple lists the selected method for missing value handling, outlier handling, normalization, feature limits, smoothing, and low-pass filtering. The results confirm that performance varies strongly depending on the chosen combination, certain settings, frequently appear among the top pipelines, while poorly aligned combinations can drop accuracy below baseline.

Statistical Analysis Each pipeline's mean served as a single observation in the statistical analysis. We performed: Descriptive statistics, Analysis of Top and Worst 25 Pipelines, Kruskal-Wallis *H*-tests, and visualisations.

Descriptive Values The 5-Fold accuracy averaged **63.60%** (SD 3.32%, min 52.67%, max **70.67%**). The top two configurations tied at **70.67%**, both combining: Missing: `mean`, Outlier: `locf`, Normalize: `minmax`, Smoothing: `F`, LowPass: `T`, Limits: `T` or `F` (no impact on top score), Feature limits had no effect on the top rank.

Top-25 Pipelines To identify patterns among the highest-performing configurations, we counted the occurence of preprocessing methods within the top 25 pipelines ranked by 5-Fold GroupKFold accuracy (See Fig. 12).

- For **Missing value handling**, `mean` imputation was most common (60%), followed by `delete` (40%), while `locf` did not appear in any top-performing pipeline.
- For **Outlier handling**, `mean` replacement dominated (56%), with `locf` (28%) and `delete` (16%) used less frequently.
- Regarding **Normalization**, `robust` scaling (52%) slightly outperformed `minmax` (40%), while `zscore` normalization (8%) was rare.
- The factors **Limits** and **Smoothing** appeared in both states (T/F) in roughly equal proportions.
- **LowPass** filtering was present in 68% of the top-25 pipelines, suggesting a potential performance benefit in combination with other preprocessing steps.

These occurences support the statistical findings: avoiding `locf` for missing values and `zscore` normalization, while preferring `robust` or `minmax` scaling and enabling LowPass filtering, is associated with higher accuracy.

Worst-25 Pipelines To identify recurring patterns among the lowest-performing configurations, we counted the occurrence of preprocessing methods within the worst 25 pipelines ranked by 5-Fold GroupKFold accuracy (See Fig. 13).

- For **Missing value handling**, `locf` imputation was most frequent (44%), followed by `mean` (32%) and `delete` (24%).
- For **Outlier handling**, `mean` replacement appeared most often (44%), with `delete` (32%) and `locf` (24%) less common.
- Regarding **Normalization**, `minmax` scaling dominated (72%), while `robust` (16%) and `zscore` (12%) occurred far less frequently.
- Feature **Limits** (`T`) in the majority of cases (72%), compared to disabled (`F`) in 28%.
- **Smoothing** was absent in most low-performing pipelines (76%), suggesting that omitting this step may contribute to reduced accuracy.

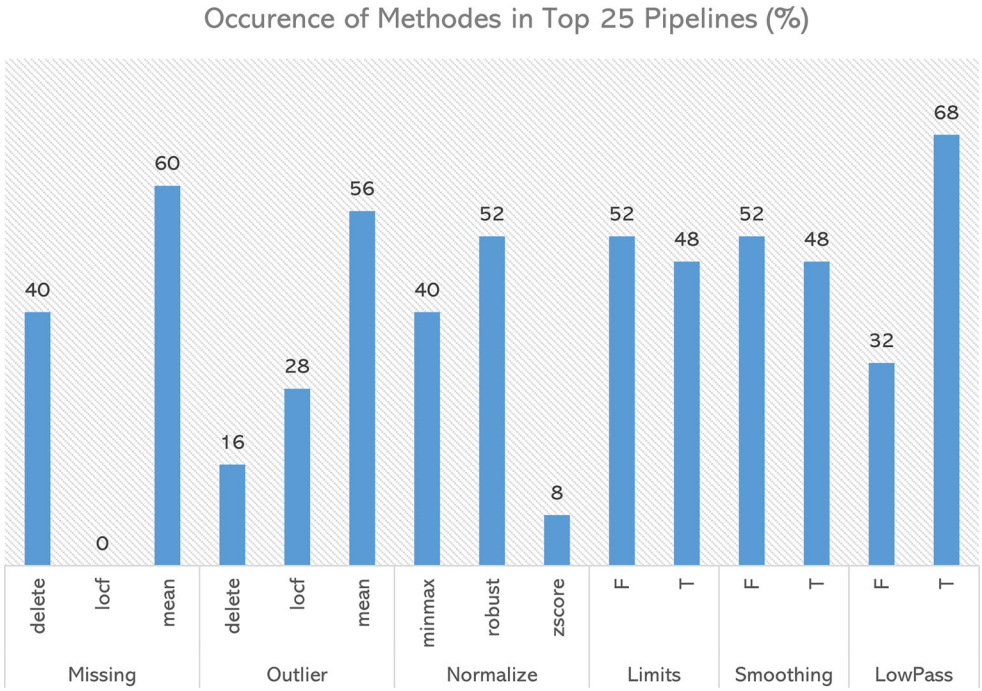

Fig. 12 Frequency Analysis in the Top-25 Pipelines

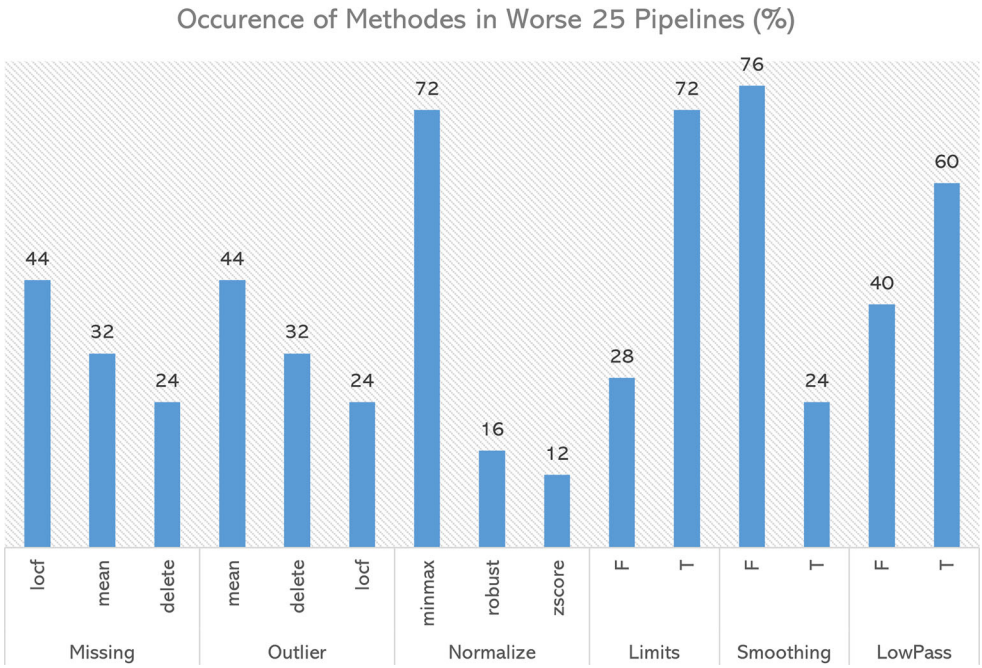

Fig. 13 Frequency Analysis in the Worst-25 Pipelines

- **LowPass** filtering occurred in 60% of the worst-25 pipelines, indicating no consistent benefit when combined with certain other settings.

These occurrences suggest that pipelines featuring `locf` for missing values, `minmax` normalization, and the absence of smoothing are more likely to underperform. This contrasts with the top-performing configurations, where `robust` scaling, `mean` imputation, and smoothing or LowPass filtering were more prevalent.

Kruskal-Wallis *H-test* To test whether different preprocessing methods produced significantly different 5-Fold accuracies, we applied the nonparametric Kruskal-Wallis *H*-test [33]. This rank-based procedure is suitable because the dependent variable (mean 5-Fold accuracy) need not follow a normal distribution, and group sizes are unbalanced. For

a factor with k levels, all N observations are pooled, ranked (average ranks for ties), and the rank sums R_j and sample sizes n_j for each level j are computed. In the H-test formulation, the uncorrected statistic is:

$$H_{\text{raw}} = \frac{12}{N(N+1)} \sum_{j=1}^{k} \frac{R_j^2}{n_j} - 3(N+1),$$

with

- N = total number of observations across all levels of the factor (here: total number of 5-Fold accuracy values across all pipelines),
- k = number of levels (distinct preprocessing methods for the factor),
- n_j = number of observations in level j,
- R_j = sum of the ranks assigned to the observations in level j after pooling and ranking *all* N values together (ties receive average ranks),
- t_m = size of tie group m (number of identical values sharing the same rank).

Because tied values occur frequently in accuracy scores, we apply the tie correction factor

$$C = 1 - \frac{\sum_m (t_m^3 - t_m)}{N^3 - N},$$

where t_m is the size of tie group m. The corrected statistic is then $H = H_{\text{raw}}/C$. Under the null hypothesis H_0 (equal population medians across levels), H approximately follows a χ^2 distribution with $k-1$ degrees of freedom, and the p-value is the upper-tail probability $\Pr(\chi^2_{k-1} \geq H)$ (Table 5).

The Kruskal-Wallis analysis shows:

- **Missing**: significant ($p < 0.001$)—median accuracy differs between levels; post-hoc tests show `locf` performs significantly worse than both `mean` and `delete`.
- **Normalize**: significant ($p < 0.05$)—`robust` scaling outperforms `zscore`; difference between `robust` and `minmax` not significant.
- **Smoothing**: borderline significant ($p < 0.05$), effect size small.
- Other factors (`Outlier, Limits, LowPass`): $p > 0.05$, no significant median differences.

The choice of missing-value handling and normalization method exerts the strongest influence on accuracy. In particular, avoiding `locf` for missing values and `zscore` normalization tends to yield higher performance.

Boxplot Visualisation Figures 14, 15 show the distribution of 5-fold accuracies for each method option within the preprocessing steps that showed statistically significant effects. Each box spans the interquartile range (IQR; 25th–75th percentile), with the horizontal line marking the median accuracy. The lines extending from the box ("whiskers") reach up to 1.5×IQR from the box edges, and points beyond this range are plotted individually as outliers.

- For **Missing** (Fig. 14), the median and upper quartile for `mean` and `delete` are clearly above `locf`, confirming Kruskal-Wallis results.
- For **Normalize** (Fig. 15), `robust` shows the highest median accuracy, `minmax` slightly lower, and `zscore` lowest, with overlapping IQRs between the first two.

Interpretation The results highlight two main levers for optimizing pipeline performance: (1) Avoid `locf` for missing-value imputation, as it underperforms significantly compared to `mean` or `delete`. (2) Prefer `robust` or `minmax` normalization over `zscore`. Other preprocessing steps showed no significant independent main effects here. The selected final pipeline reflects the best trade-off found: `mean` missing handling, `locf` outlier handling, `minmax` normalization, no smoothing, low-pass enabled, and arbitrary limits setting.

5 Conclusion

This study systematically evaluated how preprocessing pipelines affect the classification performance of Eye-Tracking data in the context of detecting cheating behavior. By comparing 193 pipeline configurations across three validation methods (80/20, 5-Fold, and OOB), we addressed both *RQ1* and *RQ2* with a focus on statistical robustness and interpretability.

Table 5 Kruskal-Wallis main effects on 5-Fold accuracy (with tie correction). Significant effects at $\alpha = 0.05$ in bold.

Factor	N	k	H	p-value
Missing	192	3	*24.78*	4.16×10^{-6}
Normalize	192	3	*8.10*	1.74×10^{-2}
Outlier	192	3	1.69	4.29×10^{-1}
Limits	192	2	1.55	2.14×10^{-1}
Smoothing	192	2	*4.94*	2.63×10^{-2}
LowPass	192	2	0.59	4.44×10^{-1}

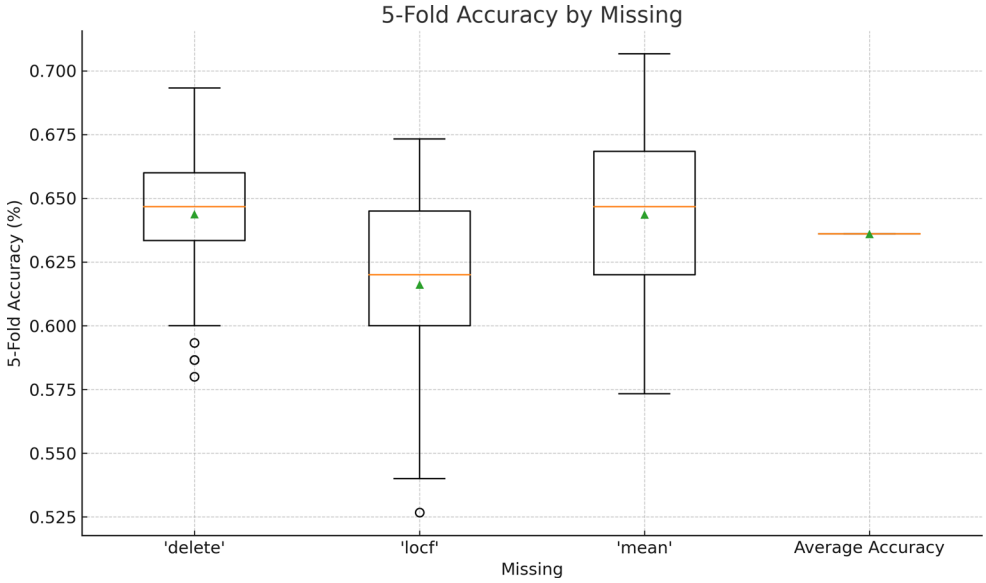

Fig. 14 5-Fold accuracy distribution by Missing-value handling method

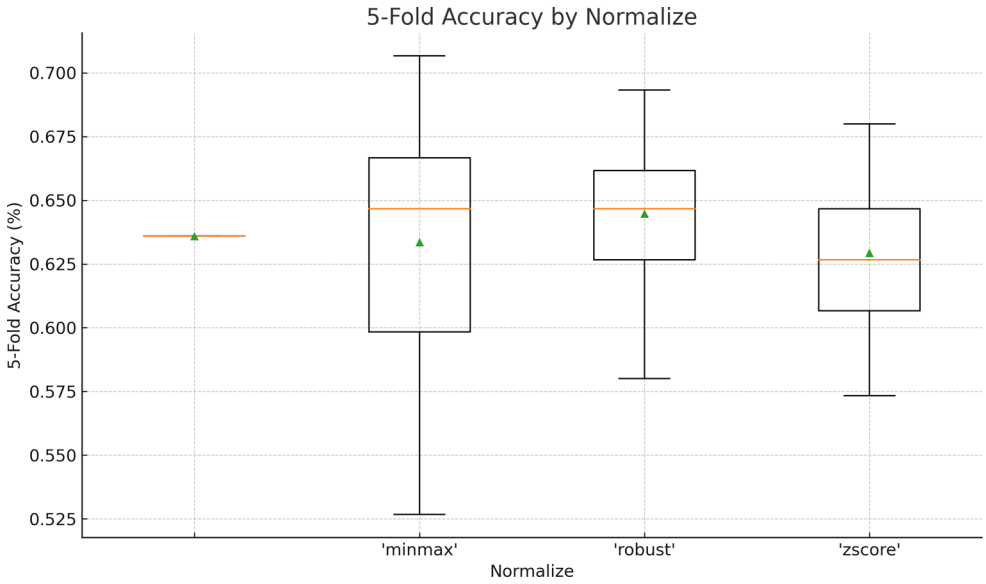

Fig. 15 5-Fold accuracy distribution by normalization method

Regarding *RQ1*, the results demonstrate that preprocessing has a measurable impact on model accuracy. The baseline pipeline (raw gaze coordinates without preprocessing) achieved 56.67% (80/20), 62.00% (5-Fold), and 63.33% (OOB). The best 5-Fold pipelines reached 70.67%, representing an 8.67% improvement over baseline. Statistical analysis via Kruskal-Wallis tests identified `Missing`-value handling and `Normalization` as the most influential factors: avoiding `locf` and `zscore` significantly improved performance. Occurrence analysis of the top 25 pipelines further highlighted the prevalence of `mean` imputation, `robust` or `minmax` scaling, and LowPass filtering.

For *RQ2*, the comparison of validation methods showed that OOB achieved the highest overall mean accuracy (65.87%) and most consistent performance. However, 5-Fold was selected as primary evaluation method due to its model-agnostic nature, lower variance than 80/20, and widespread acceptance in the literature. The two best-performing pipelines both achieved 70.67% accuracy and shared the same configuration except for the `Limits` setting: (1) Missing Value: `mean`, Outlier: `locf`, Normalization: `minmax`, Limits: `no`, Smooth: `no`, LowPass: `yes` (2) Missing Value: `mean`, Outlier: `locf`, Normalization: `minmax`, Limits: `yes`, Smooth: `no`, LowPass: `yes`. The 80/20 split produced the lowest mean accuracy (58.63%) and the largest gap between best and worst pipelines.

The Worst-25 analysis revealed common patterns among low-performing pipelines: `locf` for missing-value handling occurred most frequently (44%), followed by `mean`

(32%) and `delete` (24%). For normalization, `minmax` scaling dominated (72%), with `robust` and `zscore` appearing less often. Smoothing was absent in 76% of the worst pipelines, while Limits were enabled in 72%. These trends suggest that omitting smoothing and relying on `locf` imputation or `minmax` scaling in certain combinations can degrade performance.

The findings confirm that performance gains in Eye-Tracking classification from well-aligned combinations of preprocessing steps rather than isolated techniques. Poorly aligned combinations can underperform a minimal-preprocessing baseline.

Long-term, this work lays the foundation for a modular preprocessing framework for Eye-Tracking data. By quantifying both individual and interaction effects of preprocessing methods, the study supports the development of standardized, reproducible pipelines for behavioral research.

5.1 Summary

- Baseline performance: 56.67% (80/20), 62.00% (5-Fold), 63.33% (OOB).
- Best 5-Fold pipelines: up to 70.67% accuracy (+8.67 pp over baseline).
- Two top pipelines differed only in `Limits` setting; both used `mean` missing handling, `locf` outlier handling, `minmax` normalization, no smoothing, and LowPass.
- Significant factors: Missing-value handling (`mean` or `delete` > `locf`), and normalization (`robust`/`minmax` > `zscore`).
- LowPass filtering appeared in 68% of top-25 pipelines, suggesting synergistic effects with other steps.
- Worst-25 pipelines: `locf` for missing values (44%), `minmax` normalization (72%), smoothing absent in 76%.
- OOB achieved highest mean accuracy and consistency; 5-Fold chosen for main analysis due to broader applicability.
- Poorly chosen step combinations can degrade accuracy below baseline.

5.2 Limitations

This study is subject to several limitations. First, only the Random Forest classifier was used, selected for its robustness and prior performance; results may vary with other algorithms. Second, the dataset reflects a specific academic cheating scenario with students from one institution, limiting generalizability. Third, computational constraints restricted the preprocessing search space to a fixed grid, potentially missing optimal configurations. Finally, although feature importance was analyzed, it was not used for feature selection to avoid bias, which may have left unused potential for further performance gains.

5.3 Outlook On Future Research

Future work will extend the preprocessing framework to additional Eye-Tracking datasets, including those from varied contexts and hardware setups. A follow-up study is planned to assess the pipeline's robustness under higher-quality acquisition conditions and different task designs. Cross-domain applicability to other physiological signals (e.g., wearable sensors) will also be explored, testing the framework's scalability to data with similar noise and missingness challenges.

Further experiments will investigate the sequencing of preprocessing steps, as their order (e.g., imputation before vs. after outlier handling) may interact with model performance. Advanced imputation techniques, adaptive filtering, and hybrid normalization schemes will be integrated into the search space. Incorporating temporal modeling perspectives may also enhance detection of subtle behavioral patterns in time-series eye-tracking data.

The overarching aim remains the creation of a validated, modular preprocessing architecture adaptable to other Eye-Tracking datasets while maintaining reproducibility.

All CSV files, validation results, statistical outputs, ethical approval documentation, and implementation code are openly available in the https://github.com/eyetracking-data/eyetracking_Cheating.git.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Bixler R, Blanchard N, Garrison L, D'Mello S (2015) Automatic Detection of Mind Wandering During Reading Using Gaze and Physiology. In: Proceedings of the 2015 ACM on International Conference on Multimodal Interaction. ACM, Seattle, Washington, pp 299–306 https://doi.org/10.1145/2818346.2820742

2. Landes J, Köppl S, Klettke M (2024) Data processing pipeline for eye-tracking analysis. In: Proceedings of the 35th GI-Workshop Grundlagen Von Datenbanken Herdecke, May 22–24, 2024. CEUR Workshop Proceedings, vol 3710, pp 35–42 (https://ceur-ws.org/Vol-3710/paper6.pdf Accessed 2025-04-10. CEUR-WS.org)
3. Janke S, Rudert SC, Petersen FTM, Daumiller M (2021) Cheating in the wake of COVID-19: how dangerous is ad-hoc online testing for academic integrity? Comput Educ Open 2:100055. https://doi.org/10.1016/j.caeo.2021.100055
4. Landes J, Köppl S, Klettke M (2024) Comparison of Classifiers for Eye-Tracking Data. In: 54. Jahrestagung der Gesellschaft Für Informatik, INFORMATIK 2024—Lock in or Log Out? Wie Digitale Souveränität Gelingt Wiesbaden, September 24–26, 2024. LNI, vol P-352. Gesellschaft für Informatik, Bonn, pp 1449–1462 https://doi.org/10.18420/INF2024_126
5. Landes J, Klettke M, Köppl S (2025) Impact of preprocessing on classification results of eye-tracking-data. In: BTW2025 – Datenbanksysteme Für Business, Technologie und Web – Workshopband. Lecture Notes in Informatics (LNI), vol P-363. Gesellschaft für Informatik e. V., pp 235–256 https://doi.org/10.18420/BTW2025-127 (Refereed conference paper)
6. Mitchell RS (1995) The application of machine learning techniques to time-series data. University of Waikato, Department of Computer Science
7. Burdack J, Horst F, Giesselbach S, Hassan I, Daffner S, Schöllhorn WI (2020) Systematic comparison of the influence of different data preprocessing methods on the performance of gait classifications using machine learning. Front Bioeng Biotechnol. https://doi.org/10.3389/fbioe.2020.00260
8. Kamiran F, Calders T (2012) Data preprocessing techniques for classification without discrimination. Knowl Inf Syst 33(1):1–33. https://doi.org/10.1007/s10115-011-0463-8
9. Olsson P (2007) Real-time and offline filters for eye tracking. https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-106244. Accessed 2024-12-10
10. Adali T, Haykin S (2010) Adaptive signal processing: next generation solutions. Wiley, Hoboken
11. Li Y, Deng J, Wu Q, Wang Y (2021) Eye-tracking signals based affective classification employing deep gradient convolutional neural networks. https://doi.org/10.9781/ijimai.2021.06.002
12. Kret ME, Sjak-Shie EE (2019) Preprocessing pupil size data: Guidelines and code. Behav Res 51(3):1336–1342. https://doi.org/10.3758/s13428-018-1075-y
13. Vortmann L-M, Knychalla J, Annerer-Walcher S, Benedek M, Putze F (2021) Imaging time series of eye tracking data to classify attentional states. Front Neurosci. https://doi.org/10.3389/fnins.2021.664490
14. Zheng X, Wang M, Ordieres-Meré J (2018) Comparison of data preprocessing approaches for applying deep learning to human activity recognition in the context of industry 4.0. Sensors 18(7):2146. https://doi.org/10.3390/s18072146
15. Mishra P, Biancolillo A, Roger JM, Marini F, Rutledge DN (2020) New data preprocessing trends based on ensemble of multiple preprocessing techniques. Trends Anal Chem 132:116045. https://doi.org/10.1016/j.trac.2020.116045
16. Gholizadeh A, Boruvka L, Saberioon M, Kozák J, Vašát R, Nemecek K (2015) Comparing different data preprocessing methods for monitoring soil heavy metals based on soil spectral features. Soil Water Res 10:218–227. https://doi.org/10.17221/113/2015-SWR
17. Arnold J, Jost F, Schneider M, Schwarz F, Kübler TC (2025) A systematic literature review of eye-tracking and machine learning methods for improving productivity and reading abilities. Appl Sci 15(6):3308. https://doi.org/10.3390/app15063308
18. Lim J, Mountstephens J, Teo J (2021) Eye-tracking feature extraction for biometric machine learning. Front Neurorobot 15:796895
19. Vortmann L-M, Putze F (2021) Combining implicit and explicit feature extraction for eye tracking. Sensors 21(24):8205
20. Tawakuli A (2024) Time-series data preprocessing: a survey and an evaluation. J Time Ser Data Sci
21. Usmani M et al (2024) Preptimize: automation of time series data preprocessing framework. Algorithms 17(8):332
22. Zhang Y, Thorburn PJ (2022) Handling missing data in near real-time environmental monitoring: a system and a review of selected methods. Future Gener Comput Syst 128:63–72. https://doi.org/10.1016/j.future.2021.09.033
23. Nugroho H, Utama NP, Surendro K (2021) Normalization and outlier removal in class center-based firefly algorithm for missing value imputation. J Big Data 8(1):129. https://doi.org/10.1186/s40537-021-00518-7
24. Vinutha HP, Poornima B, Sagar BM (2018) Detection of outliers using Interquartile range technique from intrusion dataset. In: Information and Decision Sciences. Springer, Singapore, pp 511–518 https://doi.org/10.1007/978-981-10-7563-6_53
25. Pires IM, Hussain F, Garcia N, Lameski P, Zdravevski E (2020) Homogeneous data normalization and deep learning: a case study in human activity classification. Future Internet 12(11):194–1905. https://doi.org/10.3390/fi12110194
26. Priyambudi ZS, Nugroho YS (2024) Which algorithm is better? An implementation of normalization to predict student performance. AIP Conf Proc 2926(1):20110. https://doi.org/10.1063/5.0182879
27. Patro SGK, Sahu KK (2015) Normalization: a preprocessing stage. https://doi.org/10.48550/arXiv.1503.06462
28. Raju MH, Friedman L, Bouman TM, Komogortsev OV (2023) Filtering eye-tracking data from an eyelink 1000: comparing heuristic, Savitzky-Golay, IIR and FIR digital filters. J Eye Mov Res. https://doi.org/10.16910/jemr.14.3.6
29. Joel LO, Doorsamy W, Paul BS (2022) A review of missing data handling techniques for machine learning. Int J Innov Technol Interdiscip Sci 5(3):971–1005. https://doi.org/10.15157/IJITIS.2022.5.3.971-1005
30. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay É (2011) Scikit-learn: machine learning in Python. J Mach Learn Res 12:2825–2830
31. Kohavi R (1995) A study of cross-validation and bootstrap for accuracy estimation and model selection. Proc 14th Int Joint Conf Artif Intell 2:1137–1143
32. Breiman L (2001) Random forests. Mach Learn 45(1):5–32. https://doi.org/10.1023/A:1010933404324
33. Kruskal WH, Wallis WA (1952) Use of ranks in one-criterion variance analysis. J Am Stat Assoc 47(260):583–621