POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

From Historical F1 Strategy Streams to Pit Stop Decision Models

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Pietro Pizzoccheri**

Student ID: 10797420
Advisor: Prof. Emanuele Della Valle
Academic Year: 2025-26

Abstract

Formula 1 race strategy is a temporal decision problem in which teams must act before the final outcome is known, pit stops are a central example: they can restore tyre performance and create strategic opportunities, but they also impose an immediate time loss.

This thesis addresses pit decision support from historical Formula 1 data through a reproducible replay and evaluation pipeline, historical races are extracted, enriched, and transformed into replayable event streams, published through Kafka and processed by a Flink based Strategy Engine, which builds race context, emits rule-based pit recommendations, and persists JSONL artifacts for downstream learning. These artifacts are then converted into datasets for a Batch XGBoost model and a MOA Online Learner.

The evaluation is organized around two contracts; the first contract, `pit_any_h2`, checks whether an eligible pit stop occurs within a short horizon after a positive call, the second contract, `pit_success_h2`, is stricter: it requires the matched future pit stop to be classified as successful. This separates anticipating a pit window from recommending a pit that later produces a favourable outcome.

The experiments compare three decision paradigms: the deterministic Flink Strategy Engine, Batch XGBoost, and the MOA Online Learner. The results show that `pit_any_h2` is the cleaner and more learnable task. The learning systems improve over the deterministic rule baseline, with Batch providing the largest event coverage and MOA the strongest precision-oriented balance. In contrast, `pit_success_h2` is substantially harder: successful pit calls are sparse and threshold-sensitive.

Overall, this work shows that historical race state contains useful signal for pit timing, while successful pit-call prediction requires stricter accounting and remains harder to cover reliably. The main contribution is an executable pipeline for replaying historical races and comparing deterministic, Batch, and online decision systems.

Keywords: Formula 1 strategy; stream processing; pit stop prediction; online learning; XGBoost; MOA.

Abstract in lingua italiana

La strategia di gara in Formula 1 è un problema decisionale in cui i team devono agire prima che l'esito finale sia noto, i pit stop sono un esempio centrale: possono ripristinare la prestazione degli pneumatici, ma impongono una perdita di tempo immediata.

Questa tesi affronta il supporto decisionale per le chiamate ai box da dati storici, costruendo una pipeline riproducibile, le gare storiche diventano stream replayabili, pubblicati tramite Kafka e processati da uno Strategy Engine basato su Flink, che costruisce il contesto, emette raccomandazioni e persiste artefatti JSONL. Questi artefatti diventano dataset per un modello Batch XGBoost e un MOA Online Learner.

La valutazione è organizzata attorno a due contratti; il primo contratto, `pit_any_h2`, controlla se un pit stop eleggibile avviene entro un breve orizzonte dopo una chiamata positiva, il secondo contratto, `pit_success_h2`, è più severo: richiede che il pit stop futuro associato venga classificato di successo. Questa distinzione separa l'anticipazione di una finestra di pit stop dalla raccomandazione di un pit stop con esito favorevole.

Gli esperimenti confrontano tre paradigmi: il Flink Strategy Engine deterministico, Batch XGBoost e il MOA Online Learner. I risultati mostrano che `pit_any_h2` è il compito più pulito e apprendibile. I sistemi basati su apprendimento migliorano rispetto al baseline deterministico, con Batch che fornisce la maggiore copertura degli eventi e MOA il miglior bilanciamento orientato alla precisione. Al contrario, `pit_success_h2` è più difficile: le chiamate a pit stop di successo sono sparse e sensibili alla soglia scelta.

Nel complesso, lo stato storico della gara contiene segnale utile per anticipare il timing dei pit stop, mentre la previsione dei pit stop di successo richiede una contabilità più severa e rimane difficile da coprire. Il contributo principale è una pipeline per replayare gare storiche e confrontare sistemi deterministici, Batch e online.

Parole chiave: strategia Formula 1; stream processing; previsione pit stop; online learning; XGBoost; MOA.

Contents

1 | Introduction

Formula 1 races are often perceived through their most visible elements: the driver, the car, the overtakes, the pit stops, and the final classification; this view is natural, because the race is presented as a sequence of visible actions on track. Behind those actions, however, there is a continuous engineering and decision making process, since a race is not decided only by the maximum performance of the car or by the ability of the driver to extract lap time, it is also shaped by how the team interprets the evolution of the event and by how quickly it reacts when the conditions around the car change.

In recent seasons, several races have shown how decisive strategy box calls can be. The 2024 Italian Grand Prix is a clear example: Charles Leclerc and Ferrari converted a one-stop strategy into victory at Monza, while the McLarens of Oscar Piastri and Lando Norris finished behind after following a different strategic path while having a more performant car. An opposite example can be seen in the 2022 Monaco Grand Prix, Leclerc started from pole position and led the early part of the race, but the transition from wet tyres to intermediates and then slicks created a confused pit sequence for Ferrari. After being called in and then told to stay out when he had already committed to the pit lane, Leclerc lost track position and dropped to fourth, behind Pérez, Sainz, and Verstappen. In this case, the race showed how a poorly timed or poorly coordinated pit call can turn a strong on-track position into a lost opportunity. Examples of this kind show why pit strategy is one of the mechanisms through which race performance is transformed into a final result.

This is because race strategy belongs to a less visible layer, during a Grand Prix, the team has to decide when to stop, which tyre compound to use, whether to protect against a rival, whether to extend a stint, and how much risk to accept in exchange for a possible strategic gain. These decisions are connected to tyre degradation, traffic, track position, weather, safety car phases, and the behaviour of other cars; a box call that looks reasonable at the start of a lap can become weaker a few minutes later if a rival stops, if the track status changes, or if the expected rejoin position disappears.
Pit stops are one of the clearest examples of this dynamic. Stopping can restore tyre performance, create an undercut opportunity, exploit a safety car phase, or avoid future

degradation; while at the same time, it imposes an immediate time loss and can place the car into traffic. The same pit stop can therefore be a good decision in one race state and a poor decision in another, what matters is the context in which the stop is made.

This thesis studies this type of decision from a data and stream processing perspective. The work therefore treats historical races as sequences of events that can be replayed and processed in order; in this way, different decision systems can be evaluated in a controlled and reproducible setting, while preserving the temporal constraint that characterizes the original race.

1.1. Context

The context of the thesis lies at the intersection of motorsport strategy, streaming data engineering, and machine learning evaluation. The domain is Formula 1 pit strategy, but the underlying structure is more general; a system observes a flow of events, builds an internal representation of the current situation, emits decisions, and is judged later, when the outcome becomes known.

1.1.1. Formula 1 Strategy

A Formula 1 race can be read as a sequence of changing constraints, even when the strategic plan is prepared before the start, the assumptions that make that plan valid are constantly tested by the race itself. Tyres that are competitive at the beginning of a stint gradually lose performance, and the speed of this degradation depends on several factors, like track temperature, traffic, driving style, compound behaviour, and the way the car is being managed. At the same time, the gaps around the driver determine whether a stop creates an opportunity or introduces a risk. The car ahead may define the possibility of an undercut, while the car behind may make stopping dangerous if the rejoin window is too narrow. Track status adds another layer to the decision, because the cost of stopping under green flag conditions is different from the cost of stopping under Virtual Safety Car or Safety Car.

For this reason, the pit wall cannot evaluate a stop from a single signal, for instance a slower lap may indicate tyre degradation, but it may also be caused by traffic, fuel saving, lift-and-coast, or by a race phase in which maximum pace is not required. A large enough gap on paper may still be strategically weak if the car would rejoin behind slower traffic, while a smaller gap can become attractive if a rival has already stopped and is starting to apply pressure. The strategic meaning of a signal emerges from the combination of several signals, and especially from how that combination changes over time. The relevant

question is whether the current race state makes stopping preferable to waiting; that state includes tyre condition, gaps, rivals, traffic, track status, race phase, and expected pit loss. Since all these elements evolve while the decision window remains open, timing becomes part of the decision itself. Waiting one more lap can provide useful information, but it can also allow a rival to complete the undercut or close the rejoin window, while on the other hand stopping too early can protect against immediate pressure but create problems later in the stint. Pit decisions live in this narrow space between acting too early and reacting too late.

1.1.2. Race Data and Streaming Decisions

Modern motorsport produces detailed information about the race, like timing feeds, telemetry, lap information, tyre state, weather, pit events, and track status can be collected and studied after the race. These data make it possible to move from purely qualitative race review to computational race state analysis. Instead of only describing what happened, it becomes possible to reconstruct the context in which a decision was made and to ask whether a system could have identified the same situation at the right moment.

The difficulty is that post race analysis and live decision support are not the same task, since after the race, the complete sequence of events is available. It is known that a driver stopped two laps later, that a rival lost time in traffic, or that a safety car appeared shortly after a decision point; while during the race, those facts are not yet available. If such information enters the model input or the decision logic, the evaluation becomes optimistic and no longer represents the actual decision setting. A correct experimental setup must therefore preserve chronology and separate the information available at decision time from the information used later to judge the decision. Replay provides a practical way to respect this constraint while still using historical data because a race can be extracted, normalized, enriched, and saved once, then replayed as an event stream. This is useful both methodologically and practically: methodologically it prevents the decision logic from seeing the future, practically it allows the same race to be replayed multiple times, which supports debugging, comparison, and reproducibility.

1.1.3. Learning from Race State

Once the race has been reconstructed as an evolving state, the next question is whether that state contains useful signal for pit decision support; the state may include tyre life, current pace, gaps to nearby cars, track status, race progress, estimated pit loss, and other

contextual variables for each driver. These variables cannot be interpreted independently, for example tyre age has a different meaning depending on the circuit, the stint, the traffic situation, and the phase of the race; or a pace loss can mean tyre degradation in one context and strategic management in another. Different pit decision paradigms can be applied to this representation, a deterministic rule system can encode strategy knowledge through explicit scores, thresholds, gates, and promotion rules; a Batch learning model can learn patterns from historical races and produce scores on races kept out of the training set; an online learner can process the same type of data in stream order, predicting each row before using it for learning. These paradigms reflect different assumptions about the problem: human-designed logic, offline statistical learning, and incremental stream learning.

1.2. Problem and Purpose

The problem addressed by this thesis has both a temporal and a statistical component, the temporal component comes from the delay between decision and outcome; a recommendation is emitted at a specific driver-lap, while the event that confirms or contradicts the recommendation may happen later. The statistical component comes from the rarity of pit events, across a race season, most driver-lap situations are ordinary race progression rather than immediate pit opportunities. This makes accuracy a weak summary of system behaviour, since a system that almost always predicts no pit may appear accurate simply because the negative class dominates the data, but such a system would provide no useful decision support. The relevant question is what happens when the system emits a positive call. Is the call correct? How many real pit events are covered by at least one call? How does the behaviour change when the threshold is made more conservative or more aggressive? These questions shift the evaluation from general correctness to the quality and coverage of positive decisions.

The definition of a correct call also requires care since predicting that a pit stop will happen soon is not the same as predicting that the future pit stop will be strategically successful. The first task concerns timing: whether the system recognizes that a pit window is approaching; the second task is stricter, because the future pit must also be evaluated as a favourable outcome after it has happened. For this reason, the thesis separates pit decision support into two prediction contracts where the first contract concerns pit timing, while the second contract concerns successful pit call prediction. The separation is necessary because the two tasks have different meanings, different positive support, and different levels of difficulty.

So the purpose of this thesis is to study pit decision support as a reproducible stream and learning problem. The work does not attempt to replace a Formula 1 strategy engineer, nor does it try to solve the full race strategy problem, its objective is narrower and more methodological: to build an experimental setting in which different decision paradigms can be compared on the same historical race evidence, under the same temporal and statistical constraints. The implemented pipeline follows this objective from data extraction to evaluation; historical races are transformed into replayable streams, processed by a stream processing layer that builds race context and strategy artifacts, and then converted into datasets and outputs for deterministic rules, Batch learning, and online learning. The final comparison applies the same contracts to all systems, so that the observed differences can be interpreted as differences in decision behaviour rather than differences in evaluation setup. The thesis aims to clarify what can be learned from race state data, which aspects of the problem are harder, and which methodological choices are necessary to avoid misleading comparisons.

1.2.1. Research Questions

The work is guided by two main research questions.

- **RQ1.** Can pit timing be learned from historical race state patterns, when the evaluation respects the temporal order of the race?
- **RQ2.** Can successful pit call prediction be evaluated under a stricter operational contract, where false successful pit recommendations are penalized and ambiguous outcomes are handled explicitly?

The first question asks whether the current race context contains enough signal to anticipate that a pit stop will happen soon, the second question moves to a stricter decision: whether a system can identify situations that should lead not only to a future pit, but to a future pit with a favourable evaluated outcome.

1.3. Contributions

The thesis makes the following contributions:

- implementation of a reproducible pipeline from historical Formula 1 data to replayable event streams, stream-processing outputs, learning artifacts, and evaluation reports
- definition of two pit decision contracts, one for pit timing and one for successful pit

call prediction

- implementation and comparison of three decision paradigms: the Flink Strategy Engine, Batch XGBoost, and the MOA Online Learner
- construction of an evaluation protocol that separates row-level decisions, event-level coverage, strict operational precision, and diagnostic matched-pit quantities
- analysis of methodological issues such as leakage, shortcut features, threshold sensitivity, support sparsity, and score interpretation

These contributions are connected by the same objective: making heterogeneous pit decision systems comparable under explicit contracts.

1.4. Research Methodology

The research methodology is empirical and implementation driven, the work first defines the decision problem and the evaluation contracts, then implements the pipeline needed to generate reproducible artifacts, and finally compares the decision systems under the resulting protocol. This order is important because the learning results depend on the quality of the evaluation setup; if the temporal structure, the target definition, or the feature contract are wrong, the final metrics can be misleading even if the model itself is technically correct.

The evaluation uses metrics suited to rare positive decisions: precision, event coverage, $F_{0.5}$, Average Precision, Kappa, and G-Mean are used with different roles. Some quantities describe selected operating points, while others are used as diagnostics for ranking quality, race-level variability, or threshold sensitivity. The methodology also includes checks for leakage and shortcut features, because a learning result is useful only if the model is learning from race context rather than from information that would not be available or meaningful in the intended decision setting.

1.5. Structure of the Thesis

The document is organized as follows.

- Chapter 2 introduces the state of the art and the technical background. It covers event-time stream processing, replayable logs, motorsport strategy support, Batch learning, online learning, imbalanced evaluation, and explainability.
- Chapter 3 defines the problem. It introduces the race state representation, the pre-

diction horizon, the two pit decision contracts, the truth lenses, and the evaluation quantities used later in the thesis.

- Chapter 4 describes the implemented solution. It presents the architecture of the pipeline, the ingestion and replay logic, the Flink stream processing layer, the persisted artifacts, the learning datasets, and the implementation choices needed to compare the three decision paradigms.
- Chapter 5 reports the experimental evaluation. It fixes the evaluation protocol, compares the systems on pit timing and successful pit call prediction, and uses ranking, race-level, threshold, and explainability diagnostics to interpret the results.
- Chapter 6 summarizes the main findings, discusses the limits of the implemented work, and outlines future developments toward live ingestion, richer truth models, adaptive policies, and broader online learning.

2 | State of the Art

2.1. Overview

Formula 1 strategy analysis has progressively expanded from retrospective summaries and experience based decisions toward live-like decision support systems built from streaming data. Modern telemetry, timing data, and event logs make it possible to reconstruct the evolution of a race at fine temporal granularity, transforming each race into a sequence of events that can be processed, analyzed, and compared. For pit stop strategy, this shift matters since a pit decision depends on tyre age, gaps between drivers, lap times, traffic, and many other factors. The challenges are: estimating whether a pit stop is likely to occur soon, and evaluating whether a decision support system can emit meaningful pit calls under realistic temporal and statistical constraints.

This setting introduces methodological difficulties. First, the dataset is inherently sequential: the state available at lap k must be processed without using information from future laps, since in a live scenario that information would not be available. Second, pit events are rare compared with the total number of driver-lap observations, making evaluation based on accuracy inadequate. Third, the evaluation target is not unique: predicting that a pit will happen soon and predicting that a future pit will be evaluated as successful are related but distinct goals. Lastly, historical replay is useful for reproducibility, but it requires separation between event generation, stream processing, supervised learning, and final comparison of the results.

The state of the art reviewed in this chapter is organized around these; beginning with event-time stream processing and replayable logs, which provide the infrastructure for reconstructing historical races as live-like streams. It then examines motorsport strategy and pit stop decision support literature, which motivates the race context variables involved in pit decisions. The chapter then discusses learning approaches for pit stop prediction based on race state features, distinguishing between offline Batch learning and online prequential learning. The final sections focus on evaluation under class imbalance and temporal dependence, followed by explainability and score interpretation as tools for

validating the methodology. This structure provides the background needed to position the thesis as a complete pipeline from replay to evaluation for comparing a Flink Strategy Engine, Batch XGBoost, and a MOA Online Learner under aligned pit decision contracts.

2.2. Streaming and Event-Time Processing

The increasing availability of detailed sports timing data has made it possible to analyze races as sequences of evolving states; in Formula 1, each lap changes the information available to a strategy system: tyre age increases, gaps between drivers evolve, track status may change in seconds, and previous decisions constrain future options. Pit stop decision support is therefore inherently temporal since a recommendation at lap k must use only the information that would have been available at that moment.

Historical race analysis also requires a data sourcing layer; in Formula 1, timing, lap, telemetry, and session information can be accessed through libraries such as FastF1, which provide a practical interface to historical race data [27]. Such a source provides the raw material needed to reconstruct the evolution of a race with lap progression, driver timing, stint information, and contextual signals. Before any streaming or learning method can be applied, these data must be normalized into a representation that can be replayed and compared across experiments. This motivates an event sourcing view of the race where it can be represented as an ordered sequence of events. Each event updates part of the race state when a lap is completed, a track status changes and so on. This representation is valuable for reproducibility because the same historical sequence can be replayed multiple times under controlled conditions.

This temporal structure directly connects the problem to event-time stream processing; the Dataflow model formalizes concepts such as event time, windowing, watermarks, and out of order event handling [1]. Event time associates each event with the timestamp of the original phenomenon, while processing time describes when the system observes the event. In a replay experiment, this separation is useful because the replay can run faster or slower than real time while the logic still follows the chronology of the original race. Windowing and watermarks define how a stream processor reasons over time, windowing groups events into finite portions of the stream, while watermarks describe the progress of event time and allow the system to handle events that arrive late or out of order. These concepts matter when several streamed events are combined and later on analyzed. Telemetry, lap events, and track status updates describe different parts of the same race, and the processor must combine them into a coherent race state.

Apache Flink is a distributed stream processing framework designed to process contin-

uous event streams with stateful operators and event-time semantics [8], its role in the pipeline is to execute computations that evolve as new events arrive. This property is relevant for race analysis because the state of a driver is built progressively from the current tyre stint, the gaps to other drivers, the recent pace trend, the track status, and the actions already taken all depend on previous events. Flink supports this type of computation through keyed state and event-time operators, keyed state allows the processor to maintain separate evolving contexts for each driver and race, while event-time operators update these contexts according to the chronology of the replayed events. This makes the framework suitable for situations where the same stream contains many interleaved entities, such as different drivers, laps, and race sessions. In a pit decision setting, this stateful view is needed before any strategy signal or learning feature can be derived from the stream.

Apache Kafka is a distributed event streaming platform based on persistent logs [22]. A Kafka topic stores an ordered sequence of records that can be produced by one component and consumed by one or more downstream components, this model is useful when several parts of a system need to observe the same stream without tightly coupling the producer to the consumers. For race replay, Kafka provides the ingestion layer between historical data and stream processing; once historical race events are published to a topic, the same ordered sequence can be consumed by the stream processor and replayed again for debugging or experimental regeneration. This supports reproducible comparison because different processing or learning components can consume the same race stream and be evaluated under the same input conditions.

Together, the Dataflow model, Flink, and Kafka provide the streaming foundation needed for replay-based race analysis. Historical race data can be represented as ordered events, replayed through a log, and processed by stateful operators that maintain the evolving race context.

A reproducible pipeline also needs persistent intermediate outputs, columnar formats such as Parquet are useful for storing replayable snapshots, while data lake outputs allow stream processing results to be reused by later evaluation and learning stages. This storage layer separates the production of race events from downstream analysis because the same processed artifacts can feed deterministic evaluation, Batch learning, online learning, and report generation. The next section moves from this stream infrastructure to the race strategy concepts needed to interpret pit stop decisions.

2.3. Pit Stop Strategy and Motorsport Decision Support

Pit stop strategy is one of the main tactical choices in circuit racing because a call to the boxes can restore tyre performance and create future pace advantage, while also introducing an immediate time loss and changing the driver's track position; after the stop, the driver may also rejoin behind slower cars and lose additional time in traffic. The value of a pit stop therefore depends on the race because the same stop can be beneficial in one situation and damaging in another, depending on tyre degradation, gaps to rivals, circuit characteristics, safety car phases, traffic after the stop, and the possibility of undercut or overcut moves.

The undercut and the overcut are two common strategic mechanisms used to gain position around a pit sequence, an undercut attempts to pit earlier than a rival and use the pace of fresh tyres to gain time before the rival stops, an overcut delays the stop, relying on clean air, tyre management, or traffic effects to remain ahead after the pit sequence. Both cases show that the value of a stop depends on timing and on the surrounding race state, rather than on tyre age alone.

A second family of approaches is rule-based strategy support, in this case, domain knowledge is encoded through explicit conditions or scoring rules. A rule may increase the urgency of a pit call when tyre degradation is high, when a rival is within undercut range, or when the expected rejoin position avoids traffic. This type of approach is useful because the reasoning is interpretable and can be inspected by a human strategy engineer, its behaviour, however, depends on the quality of the chosen signals and thresholds.
Existing motorsport strategy research has approached pit stop planning from different perspectives. Optimization work formalizes the choice of stop laps and tyre compounds as a search problem, where the objective is to minimize race time or maximize expected race outcome [9, 29, 31], other contributions focus on virtual strategy engineering and data-driven motorsport analytics, combining historical or live data with domain reasoning [20, 36]. These works identify the variables that are usually needed to reason about pit stops: pace evolution, tyre degradation, pit loss, traffic, rival gaps, race phase, and track conditions.

This literature provides the domain background for interpreting pit decisions from replayed race states. At a given lap, the available information can be used to ask whether a pit event is becoming likely and whether such a pit event would be strategically favourable; the focus is therefore on decision support from observed race context where the system

must reason from the state available at that moment, and the evaluation must later check whether the resulting pit calls correspond to meaningful race events.

The strategy literature motivates the race context variables used by the system, while the comparison between deterministic rules, Batch learning, and online learning requires an explicit predictive formulation. Once race context has been extracted from the stream, the same problem can be studied as prediction over driver-lap feature rows. This learning perspective is introduced next.

2.4. Learning Approaches for Race State Prediction

After race context has been transformed into driver-lap observations, pit decision support can also be studied as a supervised learning problem. Each row describes the performance of a driver at a given lap, while the target describes whether a relevant event occurs in a short future horizon. The formulation is used for two related contracts: predicting whether any pit stop will occur soon, and predicting whether a successful evaluated pit stop will occur soon. This representation leads to a tabular learning setting where the input features describe different aspects of the race state like lap time information, tyre age, relative pace, race progress, gaps to other drivers, and other context signals. These variables may interact in non-linear ways, for example tyre age may have a different meaning depending on track position, traffic, race phase, or the gap to a rival. **Gradient-boosted tree models** are often used for this type of structured data because they can model such interactions without requiring the same assumptions as linear models. This makes XGBoost a suitable Batch learning baseline for the thesis [10]. Its tree structure also supports later inspection of feature influence, which is useful when checking whether the model relies on meaningful race state signals. The output of a supervised model is usually a score rather than a final decision by itself, this matters in pit decision support because the operating point can be changed depending on the desired balance between false alarms and missed opportunities. A high threshold may emit fewer pit calls with higher precision, while a lower threshold may cover more events at the cost of more false positives. For this reason, score-based learners naturally connect to threshold analysis.

Recent motorsport machine learning work shows that predictive models can be applied to racing data and pit-related problems [13, 21, 34]. These works provide domain context, although the exact validation setting remains important, rows from the same race are temporally and contextually dependent, and features derived from future outcomes can easily create leakage if they enter the learning matrix.

For this reason, the Batch learner can be evaluated through race-wise and race-sequential

out-of-fold predictions because the objective is to make the learned predictions comparable to decisions produced in stream order, where future rows are not available at prediction time. This leads to the second learning paradigm considered in the work: online learning under prequential evaluation.

2.5. Online Learning and Stream Evaluation

Batch learning assumes that the training set is available before the model is evaluated while Online learning considers a different setting: samples arrive sequentially, predictions are produced as the stream progresses, and the model may be updated after each observation. This view is relevant for decision support over streams, where the chronological order of the observations is part of the problem since a model that receives race states one after another should be evaluated in the same order, since future rows would not be available at prediction time.

MOA is a framework for stream mining and online learning experiments over large or evolving data streams [4], its role is practical in this context because it provides learners, stream readers, and evaluation protocols designed for sequential data. Instead of fitting a model once and then applying it to a fixed test set, MOA supports an incremental workflow where the learner can update its internal state as new samples arrive; this matches the structure of a replayed race, where the model observes a sequence of driver-lap states rather than an unordered collection of rows. This makes MOA a natural experimental counterpart to Batch learning: the same feature rows can be consumed as a stream, and the learner can produce predictions in the same order in which the rows are observed.

A common evaluation protocol in this setting is prequential evaluation, each instance is first used for testing and only afterwards for training. This test then train order preserves the causal constraint of the stream since the model is evaluated before it can learn from the same row. In a racing context, this protocol is important because it avoids using information from a lap before producing the prediction for that lap. It also gives a direct way to compare online predictions with the chronological order used by the streaming pipeline.

In many applications, the relation between the input features and the target can change over time, a phenomenon usually described as concept drift [2, 16]. This aspect is not evaluated in this thesis as a separate drift-detection task, but it motivates the choice of an online learner that is designed for evolving streams rather than for a fixed training table because race data have a naturally changing structure: the first laps of a race are different from the final stint, tyre behaviour changes with compound and age, and the

same feature value may have a different meaning depending on circuit, season, race phase, or track status.
For this reason, adaptive stream-learning methods provide the appropriate methodological background for the MOA branch; ADWIN is relevant because it is a standard adaptive windowing method for detecting changes in a data stream, while Adaptive Random Forests represent a broader family of drift-aware online ensembles [3, 17].

Within this stream-learning setting, the selected MOA configuration is OzaBoostADWIN. It combines online boosting with adaptation based on ADWIN, and it provides per-instance votes that can be recorded during prequential evaluation [3, 28], these votes can be converted into a continuous score for the positive class, allowing threshold frontiers and precision–recall diagnostics to be computed in a way comparable to Batch scores.

OzaBoostADWIN is used as the configured online baseline for comparison with Batch XGBoost, the analysis focuses on the behaviour of the two learning paradigms under aligned feature and truth contracts. Concept drift remains part of the methodological background, while the experimental comparison concentrates on the difference between offline race-wise learning and online prequential learning.

2.6. Evaluation under Imbalance and Temporal Dependence

Pit stop prediction is a rare event problem, in a full race dataset, most driver-lap rows do not correspond to an imminent pit stop, and an even smaller subset corresponds to an imminent successful evaluated pit stop. This makes accuracy a weak summary of performancesince a system that predicts the majority no-pit class most of the time can obtain a high accuracy while producing few meaningful pit recommendations.

Precision–recall analysis is often more informative than accuracy or ROC summaries when the positive class is rare [12, 33], average Precision summarizes ranking quality across thresholds, while precision and recall describe the behaviour of selected decision policies. These metrics are interpreted together with the number of emitted positive calls. Threshold policy is also part of the evaluation problem because a model may assign informative scores, but the system must decide at which score it should emit a pit call. Cost-sensitive learning motivates the idea that thresholds should reflect the relative cost of false positives and false negatives [14]; in pit strategy, false positive advice can be costly because an unnecessary stop may damage track position or compromise the race. This motivates the use of measures that give more weight to precision, such as $F_{0.5}$, and the

Table 2.1: Evaluation quantities used to analyze rare pit decision targets.

Quantity	Evaluation level	Goal
Precision	Emitted positive calls	Measures the quality of pit calls emitted by the system
Recall / coverage	Positive rows or pit events	Measures how many relevant opportunities are detected
$F_{0.5}$	Selected operating point	Gives more weight to precision than recall
Average Precision	Score ranking	Summarizes ranking quality across thresholds
G-Mean	Confusion matrix diagnostic	Checks balance between positive and negative recognition
Kappa	Confusion matrix diagnostic	Measures agreement beyond chance

explicit analysis of threshold sensitivity.

Additional metrics help inspect behaviour under imbalance, Cohen's Kappa measures agreement beyond chance [11], while G-Mean captures the balance between positive and negative recognition [19]. These metrics complement precision–recall analysis and help inspect inspect the behaviour of the system under different decision thresholds. They are especially useful as diagnostics, since rare positives can make a single metric difficult to interpret in isolation.

Temporal dependence creates a second evaluation challenge; rows from the same race share track conditions, event history, and strategic context, so random splits at the row level can overestimate generalization. Structured validation is therefore important when examples are grouped or temporally correlated [32]. Leakage risks are particularly relevant when variables derived from the target, future outcomes, or matched event markers are accidentally included in the feature matrix [6]. The evaluation therefore separates row-level and event-level accounting, precision is computed on emitted decision rows, while coverage is computed over unique pit events. This distinction makes clear whether a system is producing accurate individual recommendations, covering actual pit events, or trading one objective against the other.

2.7. Explainability and Score Interpretation

Predictive models over race state features can produce informative scores, but the score alone does not explain which parts of the race context influenced the prediction. In a pit

decision setting, this matters because some variables have a clear strategic interpretation, such as tyre age, relative pace, gaps, traffic, and race phase. Other variables may encode dataset structure or shortcuts that improve apparent performance without representing a meaningful race signal; explainability therefore serves both for interpretation and for methodology debugging.

SHAP provides a general framework for additive feature attribution [24], the method assigns a contribution value to each feature, so that the prediction can be decomposed into the effects of the input variables. For tree ensembles, TreeSHAP provides an efficient and consistent attribution method [25]; this is particularly relevant for Batch XGBoost, where explanations can be computed directly from the trained tree model.

Score interpretation is a related issue since a model score can be informative for ranking examples even when it is not a calibrated probability. Literature on probability estimation highlights the distinction between discriminative ranking and calibrated probability estimates [18, 26]. This distinction matters when scores are used to build threshold frontiers: the score can order candidate pit situations, while the selected threshold determines when the system emits a positive call.

2.8. Positioning of This Thesis

The reviewed literature provides the building blocks for this thesis; event sourcing and streaming systems provide the concepts needed to represent a race as an ordered sequence of events, stream processing adds event-time execution, stateful computation, and replayability; motorsport strategy research identifies the variables and trade-offs involved in pit stop decisions, such as tyre degradation, traffic, gaps, pit loss, and undercut or overcut opportunities; batch and online learning provide alternative ways to learn from race state data, while evaluation under imbalance and validation designed to avoid leakage define the constraints needed for trustworthy comparison.

The remaining difficulty is the alignment between these parts; a pit decision system must operate on the race state available at a given moment, while the evaluation must later compare emitted calls with future pit events and their outcomes. This creates a link between stream processing, supervised target construction, and metric design; the same race context must be usable by deterministic rules, Batch learning, and online learning, and the resulting decisions must be evaluated under the same truth and evaluation contracts.

The contribution of this thesis is to combine these elements into a reproducible pipeline for pit decision evaluation. The emphasis is on comparing different decision paradigms

under aligned artifacts, targets, and metrics: a Flink Strategy Engine, a Batch XGBoost learner, and a MOA Online Learner.

3 | Problem Framing

3.1. From Historical Races to Decision Instances

Historical Formula 1 data are rich, but they are usually available after the race has already happened, sources such as FastF1 can expose lap records, timing data, telemetry derived signals, stint information, pit stops, gaps between drivers, track status, weather observations, and final classifications [35]. Taken together, these elements allow a race to be reconstructed with a level of detail that is useful for analysis; the difficulty starts when the same material is used to reason about decisions that, during the race, would have been made with only a partial view of the session. A strategy call lives inside that partial view; at lap k, the available information includes what has happened up to that point: the driver's current situation, the tyre stint, the recent pace, the gaps to rivals, the track status, and previous race events. The completed historical record also contains what happens afterwards, but those future elements cannot be part of the decision state since a pit call at lap k cannot rely on a pit stop that occurs at lap $k + 1$, on a gap measured after the call, or on the final outcome of the pit sequence. The first problem is therefore temporal: the race must be reconstructed in a way that keeps decision-time information separated from future information. [1, 6]

The raw data do not arrive already shaped for this purpose because different sources describe different aspects of the same race, and they often do so at different granularities; for example, lap data are organized around completed laps, telemetry can be sampled more frequently, weather and track status may change at irregular points, and pit information can appear as direct events or as stint summaries, before any decision problem can be defined, these pieces have to be aligned on a common race timeline. Missing values, repeated records, delayed updates, and inconsistent timestamps can all change the state that a decision process would see. This is where an event-based view becomes useful, a race can be read as a sequence of observations that progressively change what is known about the session; a lap is completed, a timing value is updated, a track status changes, a stint becomes older, a pit event occurs; each one of these events may modify the state of the race. Event sourcing captures this idea by keeping the ordered history of these state

changes. [15, 22] The point is to store the final race table and to preserve the path that leads to each race state.

Event streaming adds the next difficulty since once the race is represented as an ordered history, that history has to be made available as a flow of observations. The processing order has to respect the race chronology, even when the original data come from different sources; some records may be late, some may be duplicated, and some may become meaningful only after they are combined with previous state. A stream formulation therefore has to handle ordering, synchronization, and replayability before any pit decision logic can be evaluated. [1, 8, 22]

After the stream is available, the processing problem becomes stateful, but a single event rarely carries enough information to judge a pit situation because the current lap context depends on what happened before, like tyre history, pace evolution, gaps to rivals, track status, incidents, and earlier strategic choices. The processor has to maintain an evolving view of the race and update it as new observations arrive [1, 8], only then can a decision point be interpreted as a meaningful race state rather than as an isolated record. The same reasoning applies to alerts, a pit alert is emitted while the race is still unfolding, so it can be early, repeated across nearby laps, or made obsolete by a sudden change in context. An incident, a safety car, a weather change, or a rival pit stop can alter the value of a call immediately after it is produced; for this reason, alert emission and later assessment are two different moments of the problem. The alert belongs to the live decision stream, its correctness can only be checked after the relevant future race events have occurred.

The last part of the chain is storage, a live alert may be useful during a replay or a real-time setting, but learning and evaluation require artifacts that can be inspected after the fact. The stored data have to preserve the state available at each decision point, the alerts produced from that state, and the later events used to assess them. Without this boundary, different decision approaches could be compared on different versions of the race evidence. [6, 32]

This reasoning leads to the notion of a decision instance. Historical race data have to be transformed into temporally valid observations, each tied to a specific moment of the race and to the information available at that moment. A driver-lap point is a natural candidate because it connects one driver, one race, and one lap index with the surrounding race context. The following sections refine this framing by describing the race context, the prediction contracts, and the evaluation boundary.

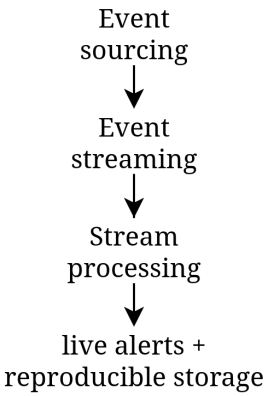

Figure 3.1: Conceptual flow from historical race data to stream-based decision support and reproducible storage.

3.2. Stateful Race Context

A pit decision cannot be described by the current lap time alone since at a given lap, the value of stopping depends on several parts of the race that evolve at the same time like the tyre stint becomes older, the pace of the driver changes, gaps to rivals open or close, track status may change, and previous strategic choices constrain the options that remain available [9, 20, 29, 31]. A pit call is therefore tied to a local race context since the current situation of a driver is the result of what happened in the previous laps: when the current stint started, whether the driver lost time in traffic, whether a rival already stopped, and other possible factors. For this reason, the problem is stateful; a decision point must summarize the information accumulated up to that moment and update it as the race changes.

The same signal can also have different meanings depending on the surrounding context, for instance an old tyre stint may suggest that a stop is becoming attractive, but the same stop may be poor if the driver would rejoin behind slower cars or is at 2 laps before the end of the race. A safety car, a yellow flag, or changing weather can modify the effective cost of stopping and can make a previously unattractive stop suddenly reasonable or even necessary. So the main difficulty of race state interpretation is that the relevant variables do not act independently [7, 20]. A pit stop places the driver back into the race at a different point of the field, the expected rejoin context determines whether the driver returns in clean air, in traffic, or close to a rival; this means that the same pit loss can have different strategic consequences depending on the cars around the rejoin position [9, 20].

Table 3.1: Race context factors involved in pit decision framing.

Factor	Role in the decision problem
Tyre state	Describes stint age, degradation, and potential pace loss
Recent pace	Indicates whether the driver is gaining or losing performance
Gaps and rivals	Defines pressure, traffic risk, and undercut or overcut chances
Drop zone	Describes the expected rejoin position and traffic context
Track status	Changes pit loss, safety conditions, and strategic opportunity
Race phase	Separates early, middle, and late-race decision behaviour
Weather	Affects tyre behaviour, risk, and the reliability of pit timing

A useful race context therefore has to combine information about the driver, the rivals, and the session; Driver-specific information describes the current stint, pace, and position; Rival information describes gaps, pressure, and possible undercut or overcut situations; Session information describes the race phase, track status, weather, and other external conditions. The decision problem emerges from the interaction between these layers.

The stateful nature of the problem also affects how alerts should be interpreted, for instance a positive pit signal may appear because several context factors align for a few laps, it would be much more difficult to be sent because a single row contains a decisive value; the signal may persist across nearby laps, weaken if the context changes, or disappear after a rival reacts. The problem can therefore be framed as the construction of race state observations from an evolving stream of information where each observation should describe the strategic context of a driver at a specific point in the race, while avoiding future information [1, 6]. The next step is to define what such observations are expected to predict, which leads to the two prediction contracts introduced in the following section.

3.3. Prediction Contracts

Once the race state has been defined, and once a pipeline or system is able to interpret it, the next problem is to decide what the pit prediction should be based on; in other words,

it is necessary to define the prediction contract that the system is expected to follow.

Predicting the exact pit lap is difficult in practice because team decisions may shift the stop by one lap, for example when two drivers should ideally stop in the same phase but cannot be served at the same time. Yellow flags, virtual safety cars, traffic, or sudden changes in race conditions may also make the exact lap of the stop less predictable; for this reason, it is reasonable to extend the prediction horizon and allow a small margin around the decision point. A system that suggests a pit one lap before the actual stop may still be producing a meaningful recommendation [13, 21, 34].

A second distinction concerns the difference between predicting that a pit stop will happen and predicting that the pit stop will actually be useful, which are related problems, but they are not equivalent. Predicting a pit stop is slightly simpler, because the target only requires the occurrence of a pit event; predicting whether the pit stop is useful also requires reasoning about the outcome of the stop, which is more complex. A pit stop may look useful at the moment of the decision, but later become less favourable because of a safety car, traffic after the stop, a poor rejoin position, or the reaction of rival cars. For this reason, the two prediction contracts are based on the same race data but represent two distinct problems. The first contract concerns the occurrence of a pit stop in the near future; the second contract concerns the occurrence of a pit stop that is later evaluated as successful, this one is expected to be more difficult, because it combines pit timing with the strategic outcome of the stop [9, 20, 34].

The difficulty is increased by the strong imbalance of the dataset since in a clean race state, without accidents, rain, or other exceptional events, each driver usually performs only one or two pit stops. As a result, most driver-lap observations are negative examples, and only a small fraction of them correspond to an imminent pit stop, the number of observations associated with successful pit stops is even smaller [19, 33].

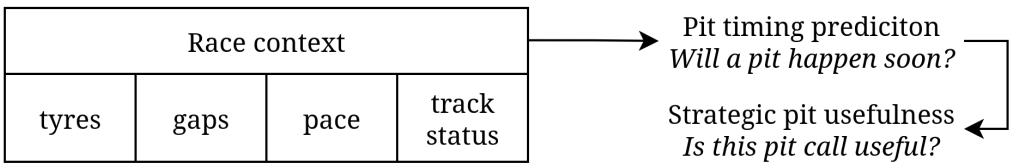

Figure 3.2: Conceptual distinction between pit occurrence prediction and successful pit prediction.

Table 3.2: Naive baselines and why they are insufficient for pit decision support.

Baseline approach	Limitation in this problem setting
Always no pit	High apparent accuracy, no operational pit support
Random pit calls	Unstable behaviour and large false-positive volume
Pit every lap	High event reach with unusable operational precision

3.4. Imbalance and Baseline Difficulty

The issue introduced in the previous section leads to one of the main difficulties of the problem, and arguably to its core challenge: the strong imbalance of the data. In Formula 1, a driver usually performs only one or two pit stops in a race when there is no rain, accident, or unusual race disruption. Considering the ground-effect era, with seasons made of roughly twenty to twenty-four races, the number of pit events remains small when compared with the number of driver-lap observations produced across all races, around 3/4% of total laps are pit laps. As a result, the pit class is much smaller than the no-pit class [19, 33].

In a setting with this level of imbalance, even a trivial system that always predicts no pit can obtain a very high accuracy which would be misleading, such a system would give no information about the reasoning behind the race and would provide no support for pit prediction. The problem becomes even harder when considering good pit stops, since they represent a smaller subset of an already rare set of pit events.

This is why accuracy alone is not enough to describe the task [12, 19, 33]; the relevant events are rare, and the objective is not simply to classify most rows correctly, but to emit pit calls that are meaningful. Metrics such as precision and recall are therefore more informative, because they focus on the positive calls and on the events that the system is able to capture. Since a false pit suggestion can be costly in a race strategy setting, a precision-oriented measure such as $F_{0.5}$ is also useful [14].

Threshold based analysis is important for the same reason, a model may assign different scores to different race states, and the selected threshold determines how many pit calls are emitted. Precision–recall curves and Average Precision help describe this behaviour across possible thresholds, rather than at a single fixed operating point [12, 33]. Additional diagnostics such as Kappa and G-Mean can also help inspect the behaviour of a system under strong imbalance, especially when comparing approaches that produce their decisions in different ways [11, 19]. These metrics make the problem more analyzable and

provide a common language for comparing the paradigms introduced in the next section.

3.5. Comparison Boundary

The pit prediction problem under strong imbalance can be approached through different decision paradigms, the first paradigm is heuristic and rule based. In this case, pit calls are produced from human interpretation of race dynamics, using explicit rules and thresholds over variables such as tyre age, gaps to rivals, track position, race phase, and other context factors. This type of approach is close to how a strategy engineer may reason about the race, because the decision logic is directly encoded in terms of interpretable race signals [20]. A second paradigm is offline machine learning; here, the problem is treated as a supervised prediction task over race-state observations. The main difficulty is to decide which information should be given to the learner, how the race context should be represented, and how the model should be trained and tuned without introducing future information. Compared with a rule based system, the model does not rely only on manually written thresholds, but learns patterns from historical examples [10, 13, 21, 34]. The third paradigm is online learning. Conceptually, this is closer to the streaming setting because the learner receives observations sequentially and updates its behaviour over time. Like a streaming rule based system, it processes the race in order, one observation after another. The difference is that the decision rule is learned incrementally from the stream, rather than being fixed entirely by manually defined conditions [3, 4, 28].

The important point is that these paradigms can only be compared if the evaluation boundary is shared. because all systems must be evaluated on the same prediction contracts, the same targets, and the same interpretation of positive and negative cases; otherwise the comparison would mix different problems. An apparent advantage in precision or coverage could come from a better interpretation of race dynamics, but it could also come from a different target definition, a different horizon, or a different denominator in the evaluation [6, 32].

For this reason, the comparison requires a common boundary before any result can be interpreted. The systems may produce their decisions in different ways, but the meaning of a pit call and the set of race situations used for evaluation must remain aligned, the next section introduces this boundary through the notion of a shared truth universe and a clean evaluation lens.

3.6. Evaluation framing

A fair comparison between decision paradigms requires a shared truth universe [6, 32]. The systems may produce pit calls in different ways, but those calls must be scored against the same set of eligible race situations; otherwise a difference in precision or coverage could depend on the denominator used for evaluation, rather than on the quality of the decision logic. This is especially important for successful pit prediction since the label of a successful or failed pit is not directly observable from the pit event alone; it depends on the interpretation of what happens around the stop, such as the effect on position, gaps, rival behaviour, and race context. For this reason, the comparison needs a canonical interpretation of pit outcomes, so that all paradigms are evaluated against the same outcome evidence, while at the same time, not every historical situation can be treated as clean evidence. Some pit outcomes may be unresolved, noisy, affected by weather, or difficult to classify in a stable way; including these cases as ordinary negatives would make the task less clear, because the system would be penalized or rewarded on situations where the strategic meaning is uncertain. A clean evaluation lens removes these ambiguous cases and keeps the comparison focused on situations where a pit recommendation can reasonably be judged [6, 32].

The evaluation also has to distinguish between row-level and event-level accounting because row-level accounting scores the individual calls emitted by a system, while event-level accounting measures how many unique pit events are covered by those calls. The two views are related, but they are not interchangeable [12, 33] since a system may emit several calls around the same pit event, or it may emit many calls that are not followed by any pit.

For the pit occurrence contract, a positive call is correct when an eligible pit occurs in the future decision window, while for successful pit prediction, the accounting is stricter: a positive call with no pit in the future window or matched to a failed or disadvantageous pit outcome is a false positive [14]. Unresolved, weather-related, unmapped, or noisy outcomes are kept outside the clean operational scoring instead of being forced into clean negatives.

Table 3.3: Operational map for positive recommendations under successful pit prediction.

Matched case in the decision window	Score class	Accounting role
Successful evaluated pit outcome	TP	Supports strict precision and successful-event coverage
No pit in the future window	FP	Operational false alarm
Failed or disadvantageous evaluated pit outcome	FP	Incorrect strategic recommendation
Unresolved, weather, unmapped, or noisy outcome	EXCLUDED	Outside clean operational scoring

From the problems described in this section derives the need of a system which evaluates pit calls in a fair and operationally meaningful way, by scoring them against a shared truth universe and by using a clean evaluation lens to focus on situations where the strategic meaning of a pit call can be reasonably judged.

3.7. Feature Engineering Opportunities

The final aspect of the problem framing concerns a conceptual but important point, since the problem can also be approached with both offline and online machine learning learners, the construction of the input features becomes part of the problem itself.

The base dataset that can be obtained from event sourcing contains many raw race variables. These variables are not always directly comparable across races [20, 32], because a race may have forty laps or more than seventy laps, lap times can change significantly from circuit to circuit, and stint lengths may vary depending on strategy, tyre degradation, track conditions, and race interruptions [7, 20]. As a consequence, the same raw value may have a different meaning depending on the race in which it appears. This opens the possibility of revisiting the raw data through feature engineering, where the objective is to represent race states in a more homogeneous form, so that they can be processed more effectively by a machine learning model [10, 13, 21]. For example, lap time information can be expressed relative to recent pace instead of being used only as an absolute value; similarly, race progress can be represented as the percentage of the race already completed, rather than only through the absolute lap number. These transformations can help describe comparable race situations even when the races have different lengths or different pace profiles [7, 32].

Table 3.4: Examples of feature engineering opportunities for race-state representation.

Raw information	Possible representation issue
Absolute lap number	Races have different lengths, so race progress may be more comparable
Absolute lap time	Circuit pace differs, so relative pace may be more informative
Stint length	Tyre age depends on strategy, compound, and race phase
Gaps to rivals	Strategic meaning depends on pit loss, traffic, and rejoin position

This does not remove the difficulty of the prediction problem, but it defines an important part of how the problem can be approached. Before comparing decision systems, the race state has to be represented in a form that is meaningful for both rule-based reasoning and learning-based methods.

This concludes the problem framing chapter. The main difficulties have been identified: transforming historical races into decision instances, maintaining stateful race context, defining the prediction contracts, handling class imbalance, setting a fair comparison boundary, and representing the race state in a way that can support learning. The next chapter describes how these issues are addressed in the proposed solution.

4 | Problem Solving: Architecture and Implementation Experience

4.1. Chapter Scope

This chapter explains how the problems described in Chapter 3 are addressed. The focus moves from the conceptual framing of the task to the operational design of the system, describing the architectural and implementation choices adopted to make the pipeline work.

The chapter is divided into two parts, the first part presents the architecture of the solution at a high level; it mainly covers the streaming data engineering layer, including event sourcing, event streaming, and stream processing with Kafka and Flink. It then describes the architectural choices used to construct the feature pipeline, feed the learning models, and compare their outputs. The final part of the architectural overview introduces the models and their connection with the pipeline. The goal of this part is to explain how they are integrated into the overall system.

The second part zooms into the architecture and describes the implementation experience in more detail, it explains the practical choices behind the implementation and how the problems introduced earlier were addressed at code and pipeline level. This includes the choices that allowed the three decision paradigms to run, produce comparable outputs, and be evaluated under the same contracts, the section also discusses the main algorithms, corrections, and implementation decisions that shaped the final system.

The quantitative results of the implementation and the comparison between paradigms are discussed in the following chapter.

4.2. Architecture of the Solution

This section describes the architecture of the implemented system, the emphasis is placed on the streaming data engineering part of the pipeline, since this is the component that connects historical race data, live-like replay, stateful processing, and the artifacts later used for learning and evaluation. The section first introduces the complete system view, and then discusses the main architectural blocks separately.

4.2.1. Full Architecture Overview

The system architecture is organized into three logical layers, as shown in Figure 4.1. The first layer is the *Streaming Core Layer*, it contains the components used to transform historical Formula 1 data into a replayable and processable stream. Historical data are first extracted from FastF1, then converted into race events and replayed through Kafka topics, these topics are consumed by Flink, which processes the stream, maintains the race state, emits live alerts, and persists JSONL artifacts used by the following stages [8, 15, 22, 35]. This is the operational base of the pipeline and is described in more detail in the next subsections. The second layer is the *Evaluation and Modelling Layer*, this starts from the artifacts produced by the streaming core and converts them into datasets, feature matrices, and truth contracts. It includes the preparation of the Parquet training dataset, the definition of the prediction contracts, the evaluation of the Flink Strategy Engine, and the training and evaluation of the learning based approaches. After the models produce their scores or labels, the comparator and the threshold frontier are used to make their outputs comparable under the same evaluation rules. The final layer is the *Delivery Layer*, once the comparison artifacts have been produced, this is responsible for generating the material used to inspect and report the results; it includes result analysis, figures, and audit checks over the produced data. The goal of this layer is to make the outputs traceable and usable for the final experimental discussion presented in Chapter 5.

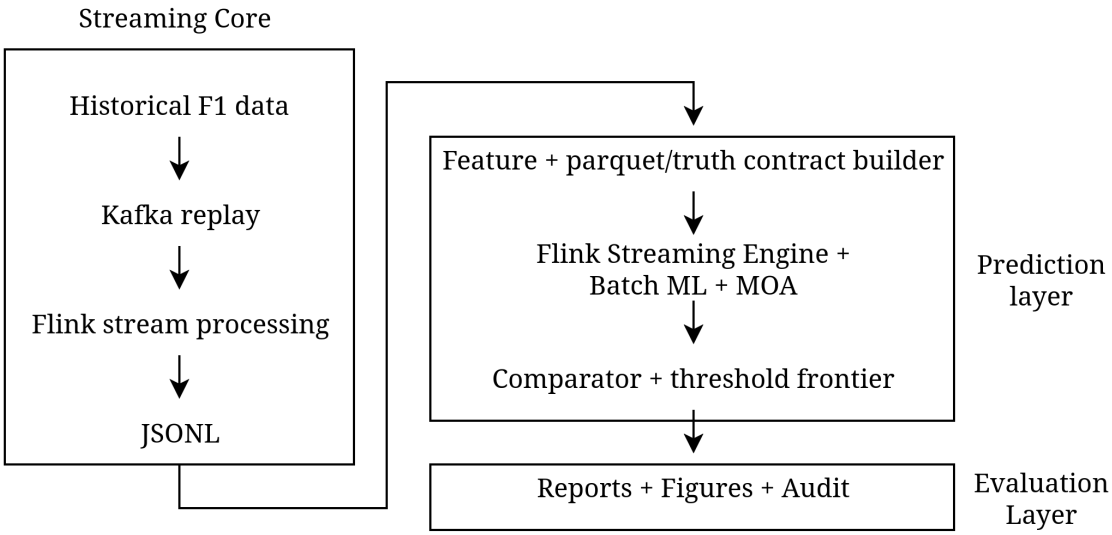

Figure 4.1: High-level architecture of the pipeline.

4.2.2. Ingestion and Replay Architecture

The pipeline starts in the Streaming Core Layer with the event sourcing stage, in this stage a Python script retrieves historical race data from FastF1 and prepares the first replayable representation of the race. The extracted data are cleaned, normalized, and enriched with the fields required by the replay and strategy pipeline; the details of this preparation step are discussed later in Section 4.3.1. At this level, the relevant architectural point is that the race is prepared once and stored as a Parquet artifact, so that the same session can be replayed multiple times without repeating the extraction process [15, 35]. The replay stage is handled by a second Python script, it reads the prepared Parquet data and publishes the race events to Kafka, race by race; the replay follows the original event time of the race, so the order of the streamed observations reflects the natural progression of the session rather than the speed at which the experiment is executed. This separation between preparation and replay makes the stream controllable: the same race snapshot can be replayed again for debugging, regeneration of artifacts, or comparison between different downstream configurations.

Kafka organizes the replay into three topics with different granularities: telemetry, laps, and track status [22]. The telemetry topic contains the highest frequency stream and describes continuous car level signals; the laps topic emits one event per driver and completed lap, carrying more aggregated information such as lap time, position, tyre life, stint information, and pit related markers; the track status topic is sparser and represents global race conditions, such as yellow flags, virtual safety cars, safety cars, and simi-

lar session-level events. Keeping these streams separated allows the processing layer to combine high-frequency car behaviour, lap-level race state, and global race context when needed.

The ingestion and replay architecture therefore provides a reproducible entry point for the whole system; historical data are fetched, prepared, stored once, and then streamed through Kafka in a consistent and analyzable form, this supports the main experimental requirement of the pipeline: the same historical race can be replayed several times while preserving the same input evidence for the following Flink, learning, and evaluation stages.

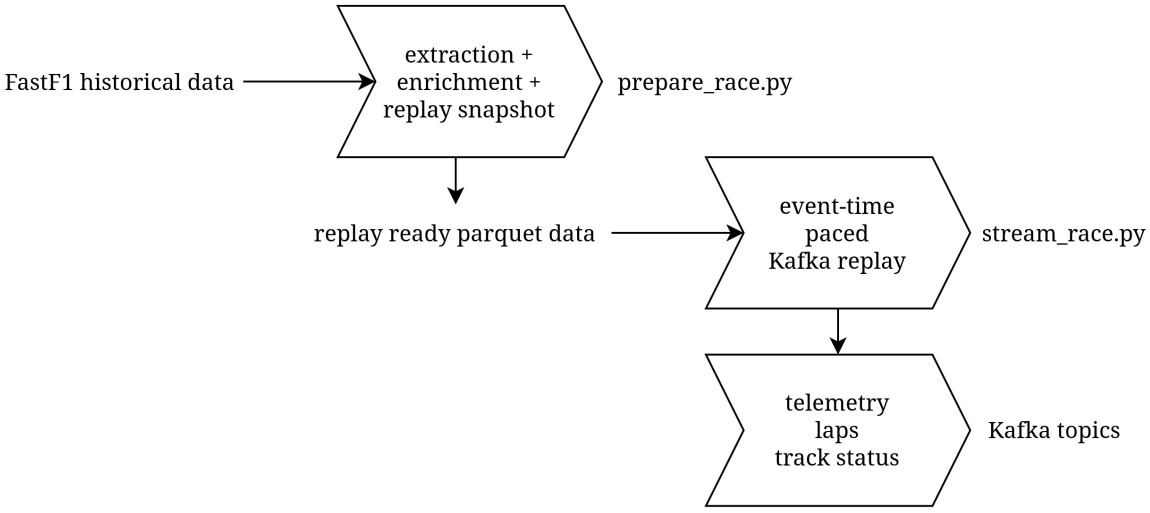

Figure 4.2: Ingestion and replay architecture. Historical race data are extracted and stored once as replay ready Parquet artifacts, then replayed through Kafka as telemetry, lap, and track status streams.

4.2.3. Flink Stream Processing Architecture

After the replay stage publishes the race topics to Kafka, Flink consumes them and becomes the main stream processing component of the system. Its role is to transform the incoming race streams into race aware strategy signals and persistent artifacts; the processing follows event time, so the state of the race is updated according to the original race chronology rather than according to the execution speed of the replay [1, 8].

At this architectural level, the relevant point is that Flink maintains the evolving race context needed by the downstream components; event-time processing is used to order race observations, state is used to remember information such as track status, gaps, tyre history, and previous driver context, while windows group events around race and lap-level situations. On top of this stateful layer, strategy operators compute the main intermediate

signals used by the system, including rival context, tyre drops, drop zones, pit evaluations, and pit suggestions. The detailed logic used to construct these signals is described later in Sections 4.3.3 and 4.3.4.

The output of this stage has two forms; live alerts are sent back to Kafka so that they can be consumed during the replay, while JSONL artifacts are persisted to the data lake and reused by the feature, learning, and evaluation stages. Flink therefore acts also as the boundary between live-like stream processing and reproducible downstream analysis.

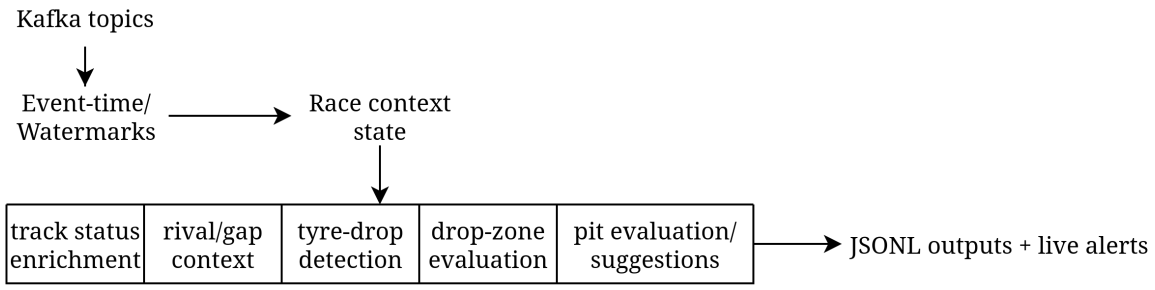

Figure 4.3: Flink stream processing architecture.

4.2.4. Data Lake and Feature Architecture

After the Flink processing stage, the stream outputs are persisted in the data lake as JSONL artifacts, this storage step defines the boundary between the live-like streaming part of the system and the reproducible learning and evaluation part. The same race replay can generate live alerts during execution, but the downstream models and comparators require stable files that can be inspected, joined, and reused without running the stream again. The data lake contains several families of artifacts, the pit timing and pit evaluation outputs describe the truth side of the problem, since they record real pit events and the post pit outcome labels; the drop-zone and tyre degradation outputs describe additional race context generated by the stream processing logic. The pit suggestion stream stores the rule based decisions emitted by the Flink Strategy Engine, while the ML feature stream stores the per-driver, per-lap rows used by the learning pipeline. Dataset preparation starts from these JSONL files and converts them into a Parquet training dataset; this step performs deduplication, joins the different stream outputs, and builds the horizon based labels used by the prediction contracts. The prepared dataset may still contain diagnostic fields, matched-pit information, and truth-related columns, because these fields are useful for audit and downstream comparison, so leakage-safe filtering is applied later when the learner feature matrix is built, by excluding target columns, truth

fields, matched-pit fields, eligibility fields, and other metadata that should not be available as model input [6].

Dataset preparation also defines the feature contract used by the learning branches. This step applies feature filtering and selected feature transformations, fields that would expose target information or shortcut information are removed, while some raw race quantities are converted into more comparable representations. For example, race progress and driver pace can be expressed as percentage features, so that the learner receives values that are less tied to a specific circuit length or absolute lap time scale; the detailed motivation for source year removal and the percent feature profile is discussed later in Section 4.3.8 and Section 4.3.9.

This architecture keeps the feature construction step traceable, the learning models do not access the original race outcome directly; they receive a filtered and aligned dataset derived from the persisted stream outputs. In this way, the data lake acts as the contract boundary between streaming engineering, Batch and MOA learning, and the final evaluation protocol.

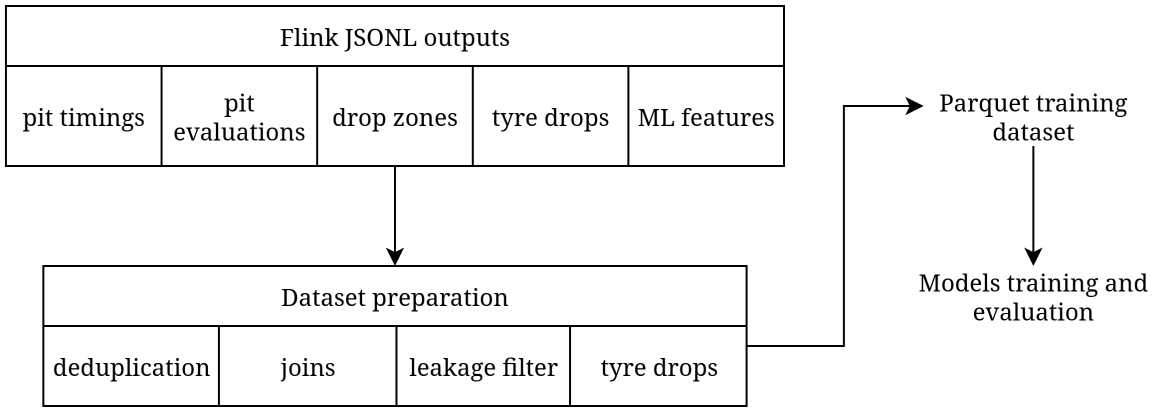

Figure 4.4: Data lake and feature architecture.

4.2.5. Truth and Comparator Architecture

After the data lake boundary has been defined, the next architectural step is to define how predictions are compared with the truth, this layer is needed because the three decision systems produce different types of output; before these outputs can be compared, they have to be converted into positive calls and matched against the same prediction contracts.

The two main contracts are `pit_any_h2` and `pit_success_h2`, both use a two-lap horizon, but the matching convention depends on the contract. In the `pit_any_h2` contract, a

positive call at lap k is correct if an eligible pit stop is matched in the same-lap-inclusive window $[k, k+2]$, this contract represents the pit timing problem. In the `pit_success_h2` contract, a positive call is correct only if an eligible future pit stop is matched in the decision window and that stop is later classified as successful by the post pit evaluation pipeline, this second contract is stricter because the system must predict both the timing of the pit and the fact that the pit will correspond to a positive strategic outcome [9, 20].

The truth used for training and comparison comes from real pit events and from their classification after the stop has occurred, the comparator uses this truth layer to evaluate the calls produced by Flink, Batch XGBoost, and MOA with the same horizon and the same eligible event universe.

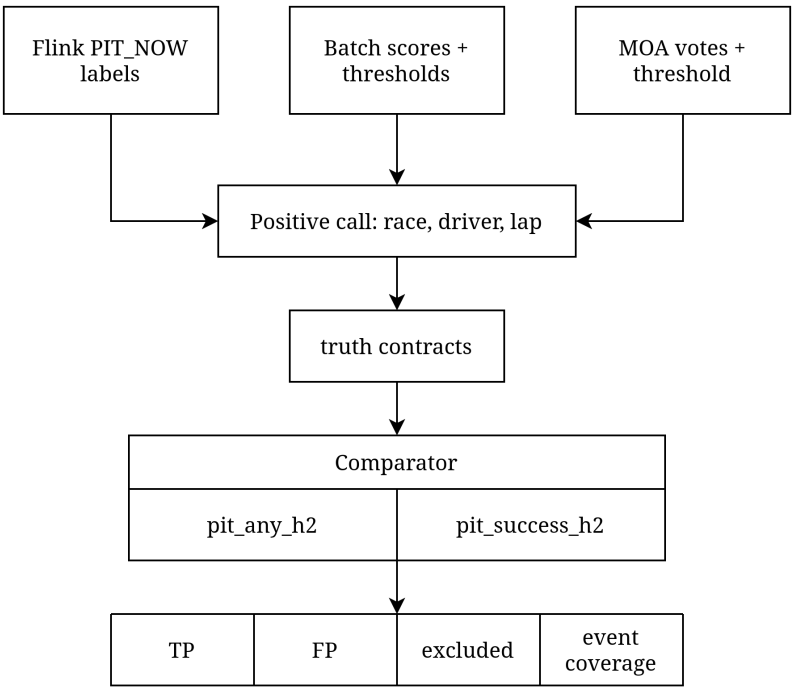

Figure 4.5: Truth and comparator architecture.

4.2.6. Model Architecture

The final architectural block concerns the way the three decision systems are connected to the common evaluation pipeline. At this point, the truth contracts and the feature artifacts have already been defined, the remaining issue is that the systems do not produce the same type of output, even though they must be evaluated under the same contracts.

The Flink Strategy Engine emits rule based recommendation labels, such as `MONITOR`, `GOOD_PIT`, and `PIT_NOW`, for the operational comparison, `PIT_NOW` is treated as a positive

pit call. Batch XGBoost instead produces out-of-fold probability scores, while the MOA Online Learner produces vote based scores during prequential evaluation [4, 10, 28]. For both learning based systems, positive calls are obtained by applying a selected threshold to the score.

The model architecture therefore hides the difference in output format behind a common decision representation. After conversion, each system produces the same logical object: a positive call associated with a race, a driver, and a lap; the comparator can then match these calls against the same future horizon truth contracts. In this way, the three systems keep their different decision mechanisms, while the final scoring remains shared.

4.3. Implementation Experience

This section moves from the architectural view to the implementation choices that made the pipeline executable; the previous section described the system as a set of connected layers, this section explains how those layers were implemented, how the main components exchange data, and which corrections were needed to obtain comparable outputs from the three decision paradigms.

All the work discussed can be found in the code repository: `https://github.com/pitesse/Thesis`.

The discussion follows the main stages of the implemented pipeline:

- historical race replay
- persisted stream artifacts produced by Flink
- construction of race context
- separation between pit suggestions and pit outcome classification
- corrections introduced to reduce invalid comparisons and leakage risks
- Batch and MOA branches, including feature profiles, MOA export, and vote logging

4.3.1. Historical Race Replay

The entry point of the pipeline is the `prepare_race.py` script, this script loads a FastF1 race session and builds the replay snapshot used by the rest of the system [35]. The prepared snapshot contains three families of events:

- telemetry events, which are the highest frequency observations and describe car level signals
- lap events, which are produced once per driver and completed lap
- track-status events, which describe global race conditions, such as yellow flags, virtual safety cars, and safety cars

Each event family is normalized before being inserted into the replay dataframe, in this context normalization means that the different sources are converted into a common event representation with a timestamp, a routing topic, and the fields required by the downstream Java models. Each row is tagged with the Kafka topic that will receive it during replay: `f1-telemetry`, `f1-laps`, or `f1-track-status`, this topic tag is kept inside

the prepared snapshot and later used by the streaming script to route each event to the correct Kafka topic.

The lap events receive additional context needed by the strategy pipeline, in particular the preparation step computes an approximate `GapToCarAhead` from cumulative lap time differences, enriches the lap rows with weather columns, preserves team and stint information, and adds the total number of laps for the race. Pit loss priors are also injected into the replay dataframe through the `PitLoss`, `VscPitLoss`, and `ScPitLoss` columns, these values give the downstream logic a consistent estimate of the pit cost under green-flag, virtual-safety-car, and safety-car conditions. After the three event families have been built, they are merged into a single replay dataframe and sorted by their absolute event timestamp, this produces the chronological event time order used during replay. The preparation step also removes the long post race tail by keeping only events that occur within a configurable buffer after the last completed lap, this avoids replay pauses caused by irrelevant post race feed activity, while preserving the useful end-of-race context.

The result of this stage is a replay ready Parquet file named according to the pattern `<year>_<race>_<session>_prepared.parquet`, this file freezes the historical race into a deterministic artifact. The race can then be replayed multiple times without repeating the FastF1 extraction and preparation process.

The second stage is implemented in `stream_race.py`, this script loads the prepared Parquet file, optionally filters the replay from a selected starting lap, and publishes the rows to Kafka. The replay follows the original event-time spacing, scaled by a configurable speed multiplier, instead of sleeping incrementally between rows, the script anchors the first event to the current wall clock and computes the target wall-clock time of every following event from its original timestamp, this avoids cumulative drift during high-speed replay and keeps the stream aligned with the intended race chronology.

4.3.2. Persisted Stream Outputs

Once the race events have been published to Kafka, the `F1StreamingJob` consumes the three input topics and processes them inside Flink, the detailed logic used to build race context and strategy decisions is discussed later in Sections 4.3.3 and 4.3.4. The focus here is the persistence layer: after the stream has been processed, the intermediate and final outputs are serialized and stored so that they can be reused by the offline stages of the pipeline.

The Flink job builds the race state using event-time processing, watermarks, keyed state, broadcast track status state, and session windows [1, 8], these mechanisms allow the

processor to combine events that arrive from different topics and to maintain a coherent view of the race while the replay is running. The strategy and truth operators then emit typed output streams, which are serialized as newline delimited JSON and persisted through Flink `FileSink` operators.

The persisted artifacts are grouped according to their role in the pipeline:

- `pit_timings`, which records where and when real pit stops occurred
- `pit_evals`, which stores the post pit strategic outcome labels, such as success, failure, or unresolved cases
- `drop_zones`, which estimates the position and traffic context in which a driver would rejoin after a pit stop
- `tire_drops`, which records tyre degradation alerts detected from lap pace behaviour
- `lift_coast`, which records lift-and-coast alerts, related to early throttle release before braking phases
- `pit_suggestions`, which stores the rule based strategy labels emitted by the Flink Strategy Engine, such as `MONITOR`, `GOOD_PIT`, and `PIT_NOW`
- `ml_features`, which contains the denormalized driver-lap rows used by the Batch and MOA learning branches

The sink behaviour is also relevant for reproducibility, the JSONL outputs are written as rolling part files in the data lake, using a policy based on file size and time. This avoids keeping the produced artifacts only in memory or in transient stream state since once the replay has completed and the files have been finalized, the downstream preparation scripts can reuse the same artifacts for dataset construction, model evaluation, and comparison reruns. In parallel, some live outputs are also routed back to Kafka, strategic alerts are sent to `f1-alerts`, while ML feature rows are also published to `f1-ml-features`, these Kafka outputs are useful for live inspection and online consumption during the replay; the thesis evaluation, however, uses the persisted JSONL artifacts as the authoritative source, because they provide stable files that can be reused across reruns and across system comparisons.

The persisted stream outputs therefore form the operational boundary between the streaming execution and the reproducible offline stages. Once these files have been produced, the rest of the pipeline can prepare datasets, build truth contracts, train models, and compare systems without depending on the live execution state of Flink.

4.3.3. Race Context Building in Flink

This subsection describes how the main race context signals are computed inside Flink. As introduced in Section 4.3.1, the stream processor receives three input streams from Kafka: telemetry events, lap events, and track-status events, these streams describe different parts of the same race so they have to be combined before the system can produce strategy suggestions, truth artifacts, and learning features [9, 20].

The simplest context signal is the track status. Track-status events are sparse global events, while telemetry events arrive at a much higher frequency for each car, the `TrackStatusEnricher` uses Flink broadcast state to make the latest track status available to the keyed telemetry stream. Each telemetry event can then be enriched with the current race status; if no status update has been received yet, the default value is green flag, this gives the later operators a consistent view of whether the race is running under normal conditions, virtual safety car, safety car, or another status.

The rival and gap context is built at lap level, lap events are grouped with event-time session windows keyed by race and lap. Inside each window, the events are sorted by race position, and the `RivalIdentificationFunction` identifies the driver ahead, the driver behind, and the corresponding gaps for each car, this position based construction keeps the local race context more stable than a purely timestamp based grouping, especially in cases where lapped cars or irregular lap completion times could make the ordering harder to interpret. The same operator also emits one `MLFeatureRow` side output for every valid driver-lap row, combining lap information with the local gap context, these side outputs become the denormalized driver-lap rows used later by the learning pipeline.

Tyre degradation is computed per driver, the `TireDropDetector` keeps keyed state for the current stint, the best lap observed in that stint, and the number of consecutive slow laps; the detector ignores the opening laps, pit in-laps, pit out-laps, non-green laps, and large isolated anomalies. For the remaining laps, it compares the current lap time with the best lap of the stint using a dynamic percentage threshold:

$$lapTime > stintBestLap \cdot (1 + thresholdPct)$$

where:

$$thresholdPct = \max(compoundBase - fuelAdjust - hotTrackAdjust, ; thresholdFloor)$$

The compound base changes with tyre compound, the fuel adjustment makes the detector more sensitive as the race progresses, and the hot-track adjustment lowers the threshold

when the track temperature is high. An alert is emitted only after two consecutive laps above the threshold, so that isolated traffic or driving mistakes are less likely to be treated as tyre degradation [7].

The drop-zone context is computed at race level by the `DropZoneEvaluator`, the operator maintains a recent full-grid snapshot and evaluates candidate pit situations from the current race order. For a driver at position P, the evaluator walks the cars behind that driver and accumulates their `GapToCarAhead` values until the accumulated gap reaches the pit loss estimate:

$$\sum_{j=P+1}^{m} gapToCarAhead_j \geq pitLoss$$

The first position where this condition is reached defines the estimated rejoin slot after the pit stop, the pit loss value is selected according to the current track status using the green-flag, virtual-safety-car, or safety-car estimate carried by the lap event. The operator also emits explicit status codes when the situation cannot be evaluated cleanly, for example because the tyre age is too low, the track status is not suitable, or the gap context is incomplete.

These context-building operators provide the state used by the downstream strategy logic; track status, rival information, tyre degradation, and drop-zone estimates are used by the Flink Strategy Engine to score pit situations and are also persisted as artifacts for later inspection and learning. The strategy scoring and the separation between pit suggestions and pit classification are described in Section 4.3.4.

4.3.4. Pit Suggestion and Pit Classification Logic

After the race context has been constructed, the pipeline separates two different types of reasoning; the first one is the online recommendation path, implemented by the `PitStrategyEvaluator`, the second one is the post pit classification path, implemented by the `PitStopEvaluator`. The separation is intentional because recommendation labels are decisions produced before a pit stop, while pit evaluation labels are truth artifacts produced after a real pit cycle has unfolded.

The `PitStrategyEvaluator` acts before the pit decision is known, tt does not use future information such as whether a pit will actually happen or how the gap will evolve after the stop. Its input is the current race context: lap state, tyre life, pace behaviour, track status, traffic after a possible stop, rival pressure, race progress, and uncertainty. From these signals, the operator computes a continuous pit desirability score in the range 0–100 [9, 20].

Conceptually, the score can be read as:

$$S = S_{pace} + S_{track} + S_{traffic} + P_{strategy} + S_{action} + S_{timing} + P_{eor} + B_{competitive} - P_{uncertainty}.$$

The final value is clipped to the range 0–100, the terms describe different parts of the race context:

- S_{pace} is computed from stint-relative degradation. If there is no valid stint state, or no consecutive slow laps, the term is 0. Otherwise:

$$S_{pace} = 30 \cdot \min\left(1, \left(\frac{paceRatio}{0.02}\right)^{1.5}\right)$$

  where `paceRatio` is the current pace degradation versus the stint-best lap.

- S_{track} is a crisp event term:

$$S_{track} = \begin{cases} 60, & \text{if track status is SC, VSC, or VSC ending} \\ 0, & \text{otherwise} \end{cases}$$

- $S_{traffic}$ is computed from the expected emergence gap after pit loss is applied on the position ladder. Let g be the estimated emergence gap (seconds):

$$S_{traffic} = \begin{cases} 30, & g \geq 3.0 \\ 30 \cdot \frac{g-1.0}{3.0-1.0}, & 1.0 \leq g < 3.0 \\ -30 \cdot \left(1 - \frac{g}{1.0}\right), & 0 \leq g < 1.0 \\ -30, & g < 0 \end{cases}$$

  with an additional easy-pass bonus of +5 if the car ahead has high tyre age, capped at 30.

- $P_{strategy}$ is a coverage-deficit penalty on the inferred next compound. If the next-compound max stint can cover remaining laps, the penalty is 0. Otherwise:

$$P_{strategy} = -15 \cdot \min(1, 3 \cdot deficit), \qquad deficit = \frac{lapsRemaining - maxStint_{next}}{lapsRemaining}$$

- S_{action} combines tyre-life urgency and short-horizon rival reaction:

$$S_{action} = \min(30,\ urgencyScore + rivalPitReactionBoost)$$

where

$$urgencyScore = \begin{cases} 0, & tyreRatio < 0.70 \\ 30 \cdot \min\left(1, \left(\frac{tyreRatio - 0.70}{1 - 0.70}\right)^2\right), & tyreRatio \geq 0.70 \end{cases}$$

and `rivalPitReactionBoost` is capped at 12.

- S_{timing} captures decision-window pressure from local trend terms and is bounded in $[0, 10]$. It combines pace acceleration, gap expansion, strategy pressure, rival reaction, and a traffic adjustment:

  $$S_{timing} = \text{clip}_{[0,10]}(paceComp + gapComp + strategyComp + rivalComp + trafficAdj)$$

  with gates that force $S_{timing} = 0$ under caution track status or when urgency is too low and no rival boost is active.

- P_{eor} is an end-of-race suppression term:

  $$P_{eor} = \frac{-100}{1 + \exp(-15 \cdot (raceProgress - 0.92))}$$

  where `raceProgress = lapNumber/totalLaps`.

- $B_{competitive}$ is a bounded pressure bonus from close gaps and recent rival pit activity:

  $$B_{competitive} = \text{clip}_{[0,8]}(pressure_{ahead} + pressure_{behind} + recentRivalPit)$$

- $P_{uncertainty}$ is derived from a confidence score built from missing-signal penalties (gap, lap time, pit loss availability, local context, and state quality):

  $$P_{uncertainty} = (1 - signalConfidence) \cdot 12$$

  with a small reduction under non-green track status.

The constants in these terms are not arbitrary per-race values, and they are not learned from target labels; they are fixed strategy parameters selected for three practical reasons. First, some values encode domain boundaries that are already meaningful in race context, for example the traffic term uses 1.0s as a DRS-risk boundary and 3.0s as a clean-air boundary; the track-status boost is a discrete event term for SC/VSC conditions; and pit loss is selected from the corresponding green/VSC/SC estimate. Second, the score is explicitly scaled to a 0–100 decision range, this explains the bounded components (for

example $S_{pace} \in [0, 30]$, $S_{timing} \in [0, 10]$, and $B_{competitive} \in [0, 8]$), the capped bonuses, and the final clipping; the aim is to keep the contribution of each factor interpretable and to make threshold labels (`MONITOR`, `GOOD_PIT`, `PIT_NOW`) stable across races. Third, some parameters were introduced as guard rails against common replay failure modes observed during development: premature calls early in the stint, over-aggressive calls near race end, and calls emitted from low-confidence context; examples are the urgency onset at tyre-ratio 0.70, the end-of-race logistic suppression centered at race progress 0.92, and the uncertainty penalty tied to missing or weak signals, this keeps the rule score deterministic while reducing noisy actionable recommendations.

Some terms increase the desirability of a pit stop, some reduce it, and some act as safeguards against noisy recommendations, the purpose of the score is to summarize how attractive a pit call looks from the information available at that lap.

The continuous score is converted into recommendation labels through thresholds:

$$\texttt{MONITOR} \geq 40, \qquad \texttt{GOOD\_PIT} \geq 60, \qquad \texttt{PIT\_NOW} \geq 80.$$

These raw labels are then filtered by actionability gates which reduce aggressive recommendations when the context is not suitable, for example when the estimated rejoin traffic is too poor, the call appears too early, or the timing evidence is weak. The implementation also includes emission gating, so the same driver is not repeatedly alerted on consecutive laps unless the score escalates, the label changes, or the track status creates a new opportunity. A separate `LOST_CHANCE` label is emitted when a previously strong pit opportunity decays below the alert threshold while the degradation trend is still worsening.

The `PitStopEvaluator` follows the opposite direction, it acts only after a real pit entry has been observed; its task is to classify the completed pit cycle by looking at the post pit evidence, especially the evolution of the gap with respect to the relevant rival, this output is used as truth for the successful pit contract.

The evaluator is implemented as a state machine so when a pit entry is detected, the operator registers a pending pit cycle and stores the pre pit context, including the driver, lap, stint, track status, tyre age, baseline lap time, and nearby rivals. The cycle first waits for a settle period, so that the immediate pit in lap and out lap noise does not dominate the classification, after the settle period, the evaluator waits for rival evidence or for an offset timeout. The cycle is then resolved with an outcome label, or marked as unresolved when the available evidence is not sufficient.

The main comparison is based on the change in gap before and after the pit, normalized

by a baseline lap time:

$$\Delta_{gap}\% = \frac{postGap - preGap}{baselineLapTime} \cdot 100.$$

This normalization avoids using the same raw second threshold on circuits with very different lap times, a small gap change has a different meaning on a short circuit than on a long one, so the percentage form makes the rule less tied to a specific track. In the same strategy case, a sufficiently favourable gap delta leads to success labels such as `SUCCESS_UNDERCUT` or `SUCCESS_OVERCUT`; stable gap conditions can produce `SUCCESS_DEFEND`; and poor post pit conditions can lead to failure labels such as `FAILURE_TRAFFIC` or `FAILURE_PACE_DEFICIT`.

The evaluator also handles cases that do not fit a clean same strategy comparison; if the relevant rival does not pit within the expected time, the pit cycle can be treated as an offset strategy and classified as `OFFSET_ADVANTAGE` or `OFFSET_DISADVANTAGE`. Stops under caution can be classified as free stop situations when the gap distortion is small, wet weather stops and incomplete evidence are kept in separate families, such as `WEATHER_SURVIVAL_STOP` or `UNRESOLVED_*`, these labels are important later because the clean learning and evaluation contracts do not treat every historical pit cycle as equally reliable evidence.

The two paths therefore produce different artifacts with different meanings. `PitStrategyEvaluator` produces online recommendations from the current race state, while `PitStopEvaluator` produces post pit outcome labels from completed pit cycles; keeping them separate prevents the evaluation from confusing a decision emitted by the system with the truth used to judge that decision.

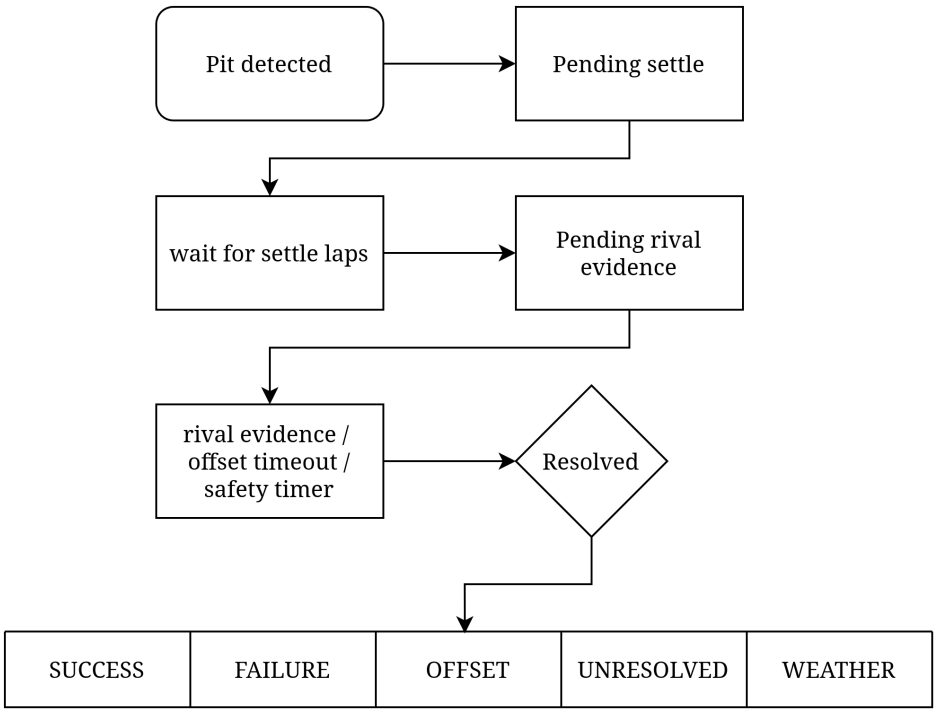

Figure 4.6: Pit cycle evaluation flow diagram.

4.3.5. Addressing Invalid Data and Leakage Risks

At this point the pipeline is able to replay races, produce stream artifacts, and feed the learning branches, but this is not sufficient for the final thesis comparison. A working pipeline can still produce unfair results if the systems are evaluated on different subsets of the data, if future information leaks into the model input, or if ambiguous pit outcomes are mixed with clean decision cases [6, 32]. A correction phase is therefore introduced before freezing the final protocol.

The first correction concerned the denominator of the comparison since the three systems do not start from the same output table, Batch XGBoost and MOA are evaluated from the learner rows produced by the ML pipeline, while the Flink Strategy Engine is evaluated from the archived `pit_suggestions` emitted during replay, these sources may cover slightly different races or drivers because of filtering, missing artifacts, replay coverage, or invalid data; if the outputs were compared directly, a difference in the final metrics could partly come from a different evaluation population rather than from a different decision method. To remove this ambiguity, the pipeline builds an explicit shared truth universe before the final comparison. The script `build_shared_truth_universe.py` extracts a `race, driver` set from the Batch out-of-fold table and another `race, driver` set from the archived Flink Strategy Engine suggestions, it then computes their intersection which

becomes the common denominator for the evaluators: Batch, MOA, and Flink are judged only on race-driver pairs that are present in both the ML outputs and the Flink replay outputs, so that the comparison is therefore based on an explicit alignment step. The alignment happens first at race-driver level, after that each evaluator still applies its own row or call logic inside the aligned pairs: Batch and MOA use their scored driver-lap rows, while the Flink comparator uses its positive strategy calls. When event universe files are provided, the same idea is also applied to pit events, so coverage and event level metrics are computed over the same eligible pit population, this avoids denominator drift between row-level scoring, positive-call matching, and event coverage.

The second correction concerned truth eligibility because historical race data contain pit events that are valid observations, but weak evidence for supervised learning or headline comparison; for example unresolved pit cycles, wet-weather stops, red-flag or suspension contexts, early race situations, and cases with missing or out-of-range pit context. Keeping all of them in a single truth definition would make the target easier to misunderstand.

For this reason, the pipeline defines explicit truth lenses; the `raw` lens keeps all deduplicated `pit_timings` events inside the evaluated universe; the `clean_actionable` lens keeps only pit events that belong to the aligned universe and removes lap-one cases and red-flag or suspended situations; the `clean_dry_strategy` lens starts from `clean_actionable` and further removes wet, intermediate, rain-affected, and early caution-sensitive cases, these lenses are stored as boolean eligibility flags, for example `eligible_raw`, `eligible_clean_actionable`, and `eligible_clean_dry_strategy`. Target construction and comparator metrics then read those flags instead of rebuilding the eligibility logic independently.

A third correction was required for temporal evaluation because random row splits are not appropriate for this task; rows from the same race share context, and neighbouring laps often describe the same strategic episode, if one row from a race is used for training and a nearby row from the same race is used for testing, the split can expose information that would not be available in a real decision setting. The Batch branch therefore uses protocols structured by race, this grouped race protocol keeps all rows from the same race inside the same fold, while the expanding race sequential protocol uses the race order already present in the prepared dataset. It builds an ordered sequence of `(source_year, race)` pairs, trains on the prefix of that sequence, and tests on the next unseen race. This produces out-of-fold scores without testing on races already present in the training fold [32].

The MOA branch follows the same chronological idea, but in online form. Its evaluation is

prequential: for each instance, the learner first produces a prediction and vote information, and only after that it updates itself with the same instance, this preserves the test-then-train order of online learning. For the `pit_success_h2` comparator, the default strict setting uses future matching for the success contract, so a successful pit prediction at lap k is matched against future pit evidence in the horizon unless same-lap inclusion is explicitly enabled as a diagnostic option [4, 28].

The next risk was feature leakage, the prepared dataset intentionally keeps several columns that are useful for audit and comparison, these include matched pit information, truth lens flags, train eligibility flags, and target-specific diagnostic fields. Removing them too early would make the pipeline harder to inspect but using them as input would introduce data leakage, the correction is therefore applied when the learner matrix is built.

The matrix builder removes the selected target column and every target-like or truth-like column before training, this includes all `target_*` columns, aliases such as `pit_any_h2_*` and `pit_success_h2_*`, horizon labels, matched pit fields, truth lens fields, eligibility flags, and train eligibility indicators. For `pit_success_h2`, the target-specific `*_train_eligible` mask is also applied before the final training and evaluation rows are formed. The remaining model input is restricted to features describing the state of the race, with further exclusions depending on the selected feature profile.

The `pit_success_h2` contract required a separate correction because the outcome labels of real pit cycles are not always binary. During target construction, the lookup for `pit_success_h2` searches for evaluated pit outcomes in the future horizon $[k+1, k+2]$. Matched outcomes are then mapped into three groups. Results such as `SUCCESS_*` and `OFFSET_ADVANTAGE` are treated as successful. Results such as `FAILURE_*` and `OFFSET_DISADVANTAGE` are treated as negative. Unresolved, unmapped, weather survival, or otherwise ambiguous outcomes are marked as unknown.

Unknown outcomes are not reliable supervised labels, so they are excluded from the clean train eligible rows. Lens eligibility is applied after this mapping: a successful matched pit can still be removed from the clean contract if the pit event is outside the selected truth lens, this is different from the comparator behaviour for positive calls with no matched pit. In strict operational mode, a positive `pit_success_h2` call that finds no eligible pit inside the horizon is counted as a false positive, the reason is practical: the system still emitted a successful pit call, and in an operational setting that call would have been unnecessary.

These corrections brought the original working pipeline into the evaluation protocol used in the thesis. The aim was to make sure that Flink, Batch XGBoost, and MOA are

compared on the same evidence, under the same contracts, and without columns derived from the answer entering the learner input.

4.3.6. Flink Strategy Engine Tuning

Before freezing the Flink Strategy Engine, three rule based variants were tested; the variants were implemented inside the `PitStrategyEvaluator` through deterministic strategy modules and runtime configuration flags. The comparator, the truth contracts, and the persisted `pit_suggestions` schema were not changed between variants, this made it possible to isolate the effect of the decision policy itself, while keeping the downstream evaluation path fixed.

The first part of the decision path is shared by all variants, the operator computes the same pit desirability score and converts it into the same base recommendation labels: `MONITOR`, `GOOD_PIT`, and `PIT_NOW`, these labels are then passed through deterministic gates that check whether the recommendation is actionable. The gates reduce promotions when the traffic context is poor, when the call is too early, when the signal confidence is too low, or when the timing evidence is not strong enough to justify an immediate pit call, therefore all variants start from the same score and the same actionability logic.

The difference between the variants is concentrated in the optional promotion path after the main gates. In the tuned runs considered here, prior based opportunity promotion was disabled, so the relevant changing mechanism is the rival pressure promotion, this mechanism can promote a `GOOD_PIT` situation into `PIT_NOW` when recent rival pit activity makes the decision more urgent.

The first variant is the *Strict* profile, in this configuration rival pressure promotion is disabled. A `PIT_NOW` label can therefore be emitted only when the ordinary score and gate path already supports an immediate call. Rival activity can still influence the continuous score through the general race context, but there is no final promotion step based specifically on recent rival stops, this profile represents the conservative rule baseline: it reduces noisy calls, but it can miss situations where nearby rivals create a short strategic window.

The second variant is the *Rival pressure* profile, this profile keeps the same base score and gates, but enables the rival pressure promotion. The promotion is considered only after the system has already classified the situation as `GOOD_PIT`; it is accepted when a relevant rival has recently pitted, the local gap context is compatible with a tactical reaction, and the current timing evidence is strong enough, this variant increases the reach of the rule engine, because it can react to competitive pressure that is not fully captured by the base

score alone. The drawback is that it can also introduce noisy `PIT_NOW` calls when the rival signal is present but the rest of the context is weak.

The third variant is the frozen *Flink Strategy Engine*, this profile keeps the rival pressure idea, but adds stricter guards around the promotion. A rival pressure promotion is allowed only when the recent rival signal is very close in time and when urgency, timing pressure, and local gap conditions support the call. Additional guards block promotions in dense traffic unless there is enough supporting evidence. The goal is to preserve the useful reach of the rival pressure profile while filtering weak or isolated promotions.

After this tuning step, the guarded profile was selected as the Flink Strategy Engine used in the final comparison. From that point on, the rule engine configuration was frozen.

All three variants produce the same output artifact: `pit_suggestions`, with labels such as `MONITOR`, `GOOD_PIT`, and `PIT_NOW`. The only changing element is the internal decision policy that determines when a situation is promoted to a stronger recommendation, this separation keeps the rule based baseline comparable with Batch XGBoost and MOA, while still documenting how the final deterministic profile was selected.

4.3.7. Batch ML Pipeline

The Batch learning branch is implemented through `train_model.py`, this script acts as the entry point for the offline learner pipeline; it can prepare the dataset, validate the replay contracts, run cross-validated training, write the model leaderboard, and export the out-of-fold prediction table used by the downstream comparator. In the thesis runs, the Batch branch is evaluated through the `expanding_race_sequential` split protocol, so that model scores are produced on races that were not present in the corresponding training prefix.

The first step is the construction of the learner matrix, here the prepared Parquet dataset still contains targets, audit fields, matched pit information, and eligibility columns, these fields are useful for inspection, but they cannot become model inputs. The matrix builder therefore applies the target specific row mask, especially for `pit_success_h2`, where the corresponding `*_train_eligible` column removes rows with unreliable success labels. It then drops metadata and leakage columns such as `race`, `driver`, all `target_*` columns, matched-pit fields, truth-lens fields, and eligibility markers, these columns remain available for grouping, audit, out-of-fold reconstruction, and comparator matching, but they are not passed to XGBoost as features.

The model input is therefore a description of the state of a car at a given lap, the identity

of the car instead would've introduced shortcuts for the learner. In the baseline representation, the remaining candidate features include variables such as position, tyre life, track temperature, air temperature, humidity, rainfall, speed trap, team, gaps to nearby cars, lap time, and estimated pit loss. In this way, the learner is trained on race context signals rather than on identifiers such as driver name or race name.

After the feature selection step, the remaining categorical variables are one-hot encoded, and boolean columns are cast to numeric values before training, this produces the numerical matrix used by XGBoost, while the original metadata columns are kept separately so that predictions can later be traced back to a specific race, driver, and lap.

The split protocol is designed around races, in the race sequential setup the dataset is read as an ordered sequence of `(source_year, race)` pairs. Each fold trains on the races that appear before the current test race and evaluates on the next unseen race, this produces out-of-fold predictions while preserving the chronological structure of the dataset; the implementation also checks that the race groups used for training and testing do not overlap; if an overlap is detected, execution stops instead of silently accepting a leaking fold [32].

The model used in this branch is `XGBClassifier` with binary logistic objective and `aucpr` as evaluation metric. Class imbalance is handled through `scale_pos_weight`, either from a fixed grid or from the observed class distribution, depending on the configuration [10, 19].

Each outer fold also contains an inner calibration and threshold selection step, the training part of the fold is split again by race groups into a fit block and a calibration block, the model is fitted on the fit block, then its scores on the calibration block are used to fit the selected calibrator and to choose threshold policies. The outer test race is not used during this step, this keeps threshold selection separate from the rows on which the fold is finally evaluated.

Two threshold policies are produced for each fold; the unconstrained policy selects the threshold with the best F_1 score on the calibration rows, while the constrained policy uses a precision floor and a simple utility score,

$$utility = TP - fpCost \cdot FP,$$

so that false positive calls can be penalized more strongly; if no threshold satisfies the precision floor, the policy falls back to the maximum precision candidate. The selected thresholds are then applied to the calibrated scores of the outer test race [14].

The output of the Batch branch is the winner out-of-fold table; for each evaluated driver-lap row, the table stores the race, driver, lap number, target label, raw probability, calibrated probability, fold metadata, selected thresholds, and binary predictions under the different threshold policies, this table is the authoritative Batch prediction artifact used by the comparator and by the later threshold frontier analysis.

The leaderboard aggregates the fold metrics for each configuration and selects the winner according to the configured model selection policy, with PR-AUC used as the main ranking signal and additional diagnostics kept for inspection. Calibration is treated as a probability quality diagnostic and as an input to threshold selection, it does not make the scores perfect probabilities, but it gives the Batch branch a more controlled score scale before comparison with Flink and MOA [5, 18, 23, 26, 30].

4.3.8. Source Year Removal

The prepared datasets include `source_year` by construction, this field is useful for ordering races, building temporal splits, reconstructing out-of-fold predictions, and checking which season each row comes from. It is therefore kept as metadata in the dataset and in the output artifacts.

The problem is that `source_year` is risky as a model input; if the learner receives the source year, it can exploit season identity instead of learning from the state of the race, this is especially problematic because different seasons may contain different regulations, calendars, car performance distributions, or pit strategy patterns. A model that uses this column could partly learn a shortcut from the year distribution rather than from variables such as tyre life, gaps, pace, and race progress [6].

For this reason, the final No-Year and Percent comparisons are run with the source year removed from the feature matrix, the implementation exposes this choice through the `-drop-source year-feature` flag. When the flag is active, `source_year` is excluded during feature selection before the learner matrix is passed to the model. The column still remains available as metadata for split construction and output tracing, but it is not used as an input feature.

4.3.9. Percent Feature Profile

The Percent feature profile was introduced to make selected race quantities less tied to absolute circuit scale because raw values such as lap number and lap time are not always directly comparable across races, because circuits have different scheduled lengths and

different normal lap time ranges; the goal of the Percent profile is to express part of the state of the race in a more relative form.

The final conservative profile used in the thesis removes `team`, `rainfall` and othe rminor metrics from the feature matrix. When combined with the No-Year setting, `source_year` is also removed, this gives a cleaner input contract: the learner receives race context and pace signals, but not direct season identity, team identity, or rainfall as shortcut variables.

The profile adds two main race relative features; the first one is `race_progress_pct`, computed as the current lap divided by the scheduled number of laps of the race, this tells the model whether the car is early, in the middle, or late in the race without relying on the raw lap number alone; the second one is `lapTime_vs_driver_prev_mean_pct`, computed as the current lap time divided by the driver's causal previous mean lap time, this gives the model a relative pace signal based only on past laps of the same driver. Despite the `pct` suffix, these fields are implemented as ratios rather than percentage points, where a value close to 1 means that the current value is close to its reference. Values above or below 1 indicate relative deterioration or improvement, and the implementation clips them to stable ranges to reduce the effect of extreme or noisy rows. The same feature profile logic is used when preparing the Batch matrix and when exporting the dataset for MOA, this keeps the two learning branches aligned: Batch XGBoost and MOA differ in the learning paradigm, but they receive the same feature universe for a given profile.

4.3.10. MOA Export

The MOA branch requires a different input format from Batch XGBoost, Batch training can read the prepared Parquet dataset directly and build the learner matrix inside Python. MOA instead expects a stream compatible tabular file, typically in ARFF format [4]. The export step therefore converts the same prepared dataset into a MOA readable representation. The entry point is `export_moa_dataset.py`, the script can either prepare the dataset again or reuse an already frozen dataset; before writing the MOA files, it calls the same matrix construction function used by the Batch pipeline, leakage filtering, target specific eligibility, source year removal, and feature profile exclusions are applied in the same way for both learning branches.

The exported matrix contains the numerical features selected by the shared feature contract and a binary target column named `target_y`. Categorical variables have already been converted through one-hot encoding, and boolean values have been cast to numeric values. In the ARFF file, the relation is named `moa_pit_strategy`, each feature is declared as `NUMERIC`, and the target is declared as `target_y 0,1`.

The exporter also writes a schema manifest, this file records the input dataset, the generated output paths, the number of rows and features, the feature list, the source years, the selected feature profile, the excluded features, the track mode, and the matrix contract flags; the manifest is useful because the MOA file no longer carries the original dataset context by itself.

The result is a MOA compatible stream dataset with the same semantic contract used by Batch XGBoost. Batch and MOA therefore differ only in how they learn from the data.

4.3.11. MOA Prequential Evaluation

After the MOA dataset has been exported, the online learner is executed through `run_moa_vote_logger.py`, this script is a Python wrapper around the custom Java logger used to run MOA and collect per-instance predictions. The default learner configuration is `meta.OzaBoostAdwin -s 42`, while the learner command remains parameterized so that controlled ablations can use a different MOA classifier without changing the rest of the pipeline [3, 4, 28].

The runner compiles `MoaPrequentialVoteLogger.java` and then executes it with the selected MOA jar, the exported ARFF file, the learner configuration, and the output path for the prediction log. The ARFF stream is consumed in row order, from the first exported instance to the last one. An optional instance limit can truncate the run, but this is used only for diagnostic executions.

The main invariant of this stage is the prequential order, for each row the learner first receives the instance and produces its prediction. Only after the prediction and vote values have been written does the same instance become available for training. In implementation terms, the logger calls `getVotesForInstance(...)` before `trainOnInstance(...)`, this preserves the online test-then-train protocol: the prediction for row i cannot depend on the label of row i or on later rows [4].

The Python wrapper performs several checks after the Java process finishes; it verifies that the output row index is contiguous from 0 to $N-1$, optionally checks the expected number of rows, and can compare the labels emitted by MOA against the exported `target_y` column, these checks are useful because the ARFF file no longer carries the full metadata of the original dataset, so row order and label alignment must be preserved explicitly.

The evaluation step then reconstructs an aligned pseudo out-of-fold table from the MOA prediction log, the evaluator maps MOA prediction codes back to binary labels, attaches the corresponding race, driver, and lap metadata from the prepared dataset, and stores

the decoded score as `raw_proba`. No temporal calibration layer is applied to MOA in this implementation, so `calibrated_proba` is an explicit passthrough of the decoded score. Once this table has been reconstructed, MOA can be evaluated with the same truth universe and comparator logic used for Batch XGBoost.

4.3.12. MOA Vote Logger

The MOA execution step originally produced prediction outputs that were usable as hard decisions, but weak for threshold analysis. A hard prediction only tells whether the learner selected class 0 or class 1. It does not show how close the decision was, and it gives very limited resolution for precision–recall curves, average precision, and threshold frontier analysis.

For this reason, the pipeline uses a custom Java logger, `MoaPrequentialVoteLogger`. The logger writes one row for each processed instance, with the row index, the true label, the predicted label, the class votes, and a derived positive score. The output schema is:

`row_index,true_label,predicted_label,vote_0,...,vote_7,positive_score`.

The positive score is computed from the votes of the negative and positive classes:

$$\texttt{positive\_score} = \frac{\texttt{vote\_1}}{\texttt{vote\_0} + \texttt{vote\_1}}.$$

If the denominator is zero or invalid, the logger falls back to the predicted class and writes a score of 1.0 for a positive prediction or 0.0 for a negative prediction.

This logging step does not change the learner protocol since the logger observes the votes before the model update, writes the prediction row, and then calls the normal training step on the same instance. The behaviour remains prequential: predict first, train after, the only difference is that the output contains the information needed to treat MOA as a score producing system rather than only as a hard classifier. The Python decoder adds a second layer of protection, it maps MOA class codes back to the binary target using purity checks, prefers the explicit `positive_score` column when the custom logger output is available, and can reconstruct the score from vote columns if the provided score degenerates into hard values.

The practical effect is that MOA can be evaluated with the same type of threshold diagnostics used for Batch XGBoost; the online learner still follows the test-then-train protocol, but its outputs can now support average precision, precision–recall curves, and threshold

frontier selection [4, 12, 33].

4.4. Chapter Summary

This chapter described how the problem framed in Chapter 3 was turned into an executable pipeline. The first part introduced the system architecture, from historical race ingestion and replay to Flink stream processing, persisted data lake artifacts, truth contracts, and the common comparator used to align the three decision systems.

The second part described the main implementation choices. Historical races are frozen into replayable Parquet snapshots, replayed through Kafka, processed by Flink, and persisted as JSONL artifacts, these artifacts are then used to build the learning dataset, define the truth lenses, and support the comparison between the Flink Strategy Engine, Batch XGBoost, and the MOA Online Learner.

The chapter also discussed the corrections needed to make the comparison fair: shared race and driver universes, leakage guards, target eligibility masks, race structured Batch evaluation, prequential MOA evaluation, source year removal, and aligned feature profiles, these choices define the protocol used in the following chapter, where the quantitative comparison between the three paradigms is reported.

5 | Experimental Evaluation

This chapter evaluates the resulting systems under a fixed experimental protocol, the goal is to compare the behaviour of the Flink Strategy Engine, Batch XGBoost, and the MOA OzaBoostADWIN on the same pit decision contracts [3, 4, 10, 28].

The evaluation is organized around the two targets introduced earlier: `pit_any_h2`, which measures whether a pit stop is predicted within the evaluation horizon, and `pit_success_h2`, which measures whether a successful evaluated pit stop is predicted. The chapter first defines the frozen protocol, then reports the rule engine baseline, the training and runtime characteristics of the learning branches, the headline comparison for pit timing, and the stricter comparison for successful pit calls.

5.1. Evaluation Scope and Frozen Protocol

Before presenting the results the evaluation protocol is fixed explicitly, this is necessary because the systems do not produce the same type of output and do not naturally operate on identical tables as described in Chapter 4 Section 4.3.5; the Flink Strategy Engine emits rule based recommendation labels, Batch XGBoost emits out-of-fold probability scores, and MOA emits vote based scores in prequential order [3, 4, 10, 28]. The protocol defines how these outputs are converted into comparable positive calls and how those calls are matched against truth, all headline results are computed on seasons 2022–2025 which is the ground effect F1 car era. The truth universe mode is `canonical_sde_truth`, and the main lens is `clean_actionable`, this means that the comparison uses the shared SDE-defined truth universe and removes cases that are not considered fair actionable strategy decisions under the clean lens, by doing so comparing systems on different race, driver, or event populations is avoided.

Two targets are evaluated; the first target, `pit_any_h2`, measures pit timing: a positive call is correct when an eligible pit stop is matched within the timing contract; the second target, `pit_success_h2`, measures successful pit-call prediction: a positive call is correct only when an eligible future pit stop is matched and the post pit evaluator classifies

that stop as successful. For this second contract, failed or disadvantage pit outcomes are treated as negatives, while unresolved, weather related, and unmapped outcomes are excluded from the clean success truth; both are described at 3 Section 3.3.

The compared systems are:

- the frozen Flink Strategy Engine profile
- Batch XGBoost with the No-Year feature profile
- Batch XGBoost with the Percent feature profile
- MOA Online Learner with the No-Year feature profile
- MOA Online Learner with the Percent feature profile

The Batch systems are evaluated with race sequential out-of-fold predictions, each score is produced on a race that was not used in the corresponding training prefix [32]. The MOA systems are evaluated with the prequential test-then-train protocol where each row is predicted first and then used for learning [4, 28]. The Flink Strategy Engine is evaluated from its persisted `pit_suggestions` artifacts, with the selected frozen rule profile.

The evaluation artifacts were also checked for protocol consistency before reporting the results, in particular, the `pit_success_h2` evaluation uses the corrected success contract: a positive call is matched only against future pit evidence, positive calls from the system without a real match are counted as false positives, and matched-pit success rate is kept only as a diagnostic measure. This distinction is important because successful pit call prediction is stricter than pit timing, counting these no match calls as false positives gives a more realistic operational view of the system behaviour.

5.2. Flink Strategy Engine Baseline and Freeze Decision

As described in Section 4.3.6, the Flink Strategy Engine was not used in its first rule configuration; several deterministic variants were tested before comparing the rule based system with Batch XGBoost and MOA. This step was needed to obtain a stable rule baseline, while keeping the later comparison independent from the tuning of the learning models [9, 20].

The first variant was the strict profile, in this configuration the engine mainly relied on the base score, timing gates, and actionability gates; optional promotion mechanisms were disabled, especially the rival pressure promotion. This made the system conservative: it

emitted fewer strong pit calls and avoided some noisy recommendations, but it also missed situations where nearby rivals created a relevant strategic pressure. The second variant enabled rival pressure, in this configuration recent pit activity from nearby rivals could promote a near actionable situation into a stronger recommendation; the objective was to increase the reach of the Flink Strategy Engine, especially in cases where waiting could lose a strategic window. This improved recall, but it also introduced more noisy calls. The final variant kept the rival pressure mechanism but added guards, promotions were allowed only when the surrounding context supported the call, for example through timing pressure, urgency, recent lap constraints, and gap conditions; this guarded profile was selected as the frozen Flink Strategy Engine baseline. After this point, the rule engine configuration was kept fixed and the Batch and MOA systems were compared against it under the same evaluation protocol.

Table 5.1 summarizes the tuning trajectory, the strict profile is the most conservative one, the rival pressure profile increases reach but loses precision, the frozen guarded profile recovers precision while keeping the same $F_{0.5}$ value as the rival pressure configuration.

Table 5.1: Flink Strategy Engine variant comparison before freezing the deterministic baseline.

Variant	Precision	Recall	$F_{0.5}$	Interpretation
Strict	0.232	0.068	0.156	Stable but conservative
Rival pressure	0.218	0.097	0.174	Better reach, more noise
Flink Strategy Engine	0.248	0.079	0.174	Recovered precision, equal $F_{0.5}$

The selected profile is the deterministic baseline obtained from the rule engine tuning stage; the following sections use this frozen Flink Strategy Engine when comparing the three decision paradigms.

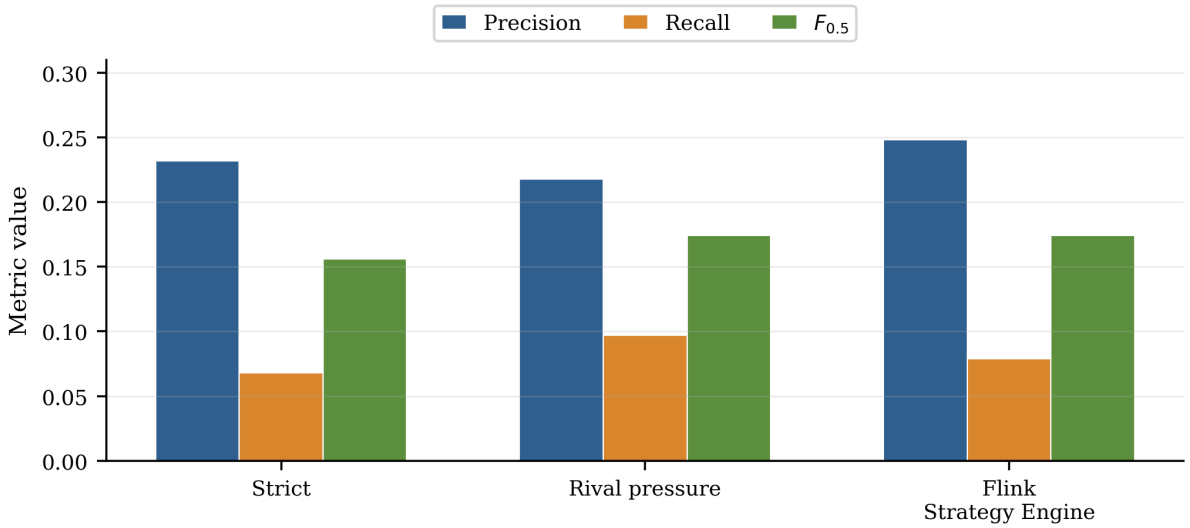

Figure 5.1: Rule-engine tuning summary for 2025 under the clean_actionable lens, showing pit_any_h2 precision, recall, and $F_{0.5}$ across the three rule-engine variants.

5.3. Training Runtime Comparison

The first experimental comparison concerns the runtime of the two learning branches, the values in Table 5.2 report the learner runtime for each profile and target; dataset preparation and MOA export are treated as separate stages and are not included in these times.

The difference between Batch and MOA is expected from the evaluation protocols described in Chapter 4, Batch XGBoost uses race-sequential out-of-fold evaluation: the model is trained on the previous races, tested on the next held-out race, then trained again on a larger prefix and tested on the following race; this repeated fitting makes the Batch branch slower [32]. MOA instead runs once over the exported stream in prequential order, each instance is predicted and then used for training, so the learner processes the stream in a single pass [4, 28].

Table 5.2: Training and evaluation runtime by learner, feature profile, and target.

System	Target	Runtime
Batch No-Year	`pit_any_h2`	1m 44s
Batch No-Year	`pit_success_h2`	1m 41s
Batch Percent	`pit_any_h2`	1m 38s
Batch Percent	`pit_success_h2`	1m 29s
MOA No-Year	`pit_any_h2`	24s
MOA No-Year	`pit_success_h2`	20s
MOA Percent	`pit_any_h2`	18s
MOA Percent	`pit_success_h2`	13s

The timing results should therefore be interpreted as runtime evidence for the selected evaluation implementation, not as a general benchmark of XGBoost against MOA since in this pipeline MOA is faster mainly because it uses a single prequential pass, while Batch XGBoost repeatedly fits models across race-sequential folds.

5.4. Pit Timing Headline Results (`pit_any_h2`)

The first headline comparison concerns the `pit_any_h2` contract, this target evaluates pit timing: when a system emits a positive call at a given driver lap, the comparator checks whether an eligible pit stop occurs within the evaluation horizon. Unlike `pit_success_h2`, this contract does not require the pit stop to be classified as strategically successful after the fact, it is therefore the cleaner task for measuring whether the systems can recognize when a pit window is approaching.

The task is still strongly imbalanced, at learner-matrix level the `clean_actionable pit_any_h2` target contains 93,623 driver-lap rows, but only 5,855 of them are positive rows, which means that positive pit-timing situations represent about 6.3% of the learner rows. A low absolute precision is therefore expected: most driver-lap situations are not close to a pit stop, so emitting a positive call is difficult and false positives are easy to produce [19].

The 5,855 positive rows are not 5,855 distinct pit stops, they are positive driver-lap examples produced by the horizon label; intuitively, one real pit can make more than one previous lap positive. For example, if a driver pits on lap k, then the rows for laps $k-1$ and $k-2$ can both be labelled as positive because both are close enough to the same future pit, with a two-lap horizon each pit can therefore generate up to two positive training rows.

For event-level interpretation, the shared comparator universe contains 3,048 unique eligible pit events, this is the denominator used when discussing event recall. The ratio between row positives and unique pit events is about $5,855/3,048 = 1.92$, close to but not exactly 2.00, this is expected because some pits occur too early to have two valid previous rows, some horizons can overlap, and the row-level learner matrix and the shared comparator event universe are not exactly the same object. The important point is that the two counts answer different questions: 5,855 describes how many driver-lap rows are positive for learning, while 3,048 describes how many unique pit events can be covered in the event-level comparison. This distinction is used throughout the interpretation, precision is computed from emitted positive calls: when the system says that a pit is coming, how often is that call correct? Event recall measures coverage over unique eligible pit events: how many real pit events were covered by at least one valid call? The row-level imbalance explains why high absolute precision is difficult, while the event-level denominator explains the coverage side of the evaluation.

5.4.1. Interpretation Table

Table 5.3 reports the selected operating points for `pit_any_h2`.

The results show that `pit_any_h2` is a learnable timing task, the Flink Strategy Engine reaches good precision for a deterministic rule baseline, but its event recall is low, which means that it covers only a limited fraction of eligible pit events. Batch No-Year improves recall over the Flink baseline, while Batch Percent improves it further, showing that the engineered relative representation is more effective for this timing task. MOA gives the strongest precision-weighted trade-off: both MOA profiles reach the highest $F_{0.5}$ values, with MOA No-Year slightly ahead of MOA Percent, which is understandable since the Percent features set is engineered from Batch SHAP analysis explain later in section 5.10.

Table 5.3: Selected `pit_any_h2` operating points.

System	Precision	Event recall	$F_{0.5}$
Flink Strategy Engine	0.271	0.071	0.173
Batch No-Year	0.204	0.122	0.180
Batch Percent	0.242	0.220	0.237
MOA No-Year	0.350	0.183	0.296
MOA Percent	0.344	0.182	0.292

Overall, the timing task favours the learning-based systems; Batch Percent gives the largest event coverage among the reported systems, while MOA provides the best precision-oriented balance. This suggests that, for predicting whether a pit will happen

soon, the learned models capture useful timing patterns beyond the deterministic rule logic.

The precision values should be read in the context of the imbalance described above since positive pit-timing rows are a small minority of the learner matrix, precisions in the 0.20–0.35 range are substantially above the row-level positive-rate baseline of about 0.063. At the same time, precision alone is not sufficient: a system can be precise but cover few real pit events, this is why the table reports precision together with event recall and $F_{0.5}$.

5.4.2. Headline Figure

Figure 5.2 shows the same `pit_any_h2` comparison visually. The figure makes the trade-off between precision, event recall, and $F_{0.5}$ easier to inspect across the deterministic Flink Strategy Engine and the learning-based systems.

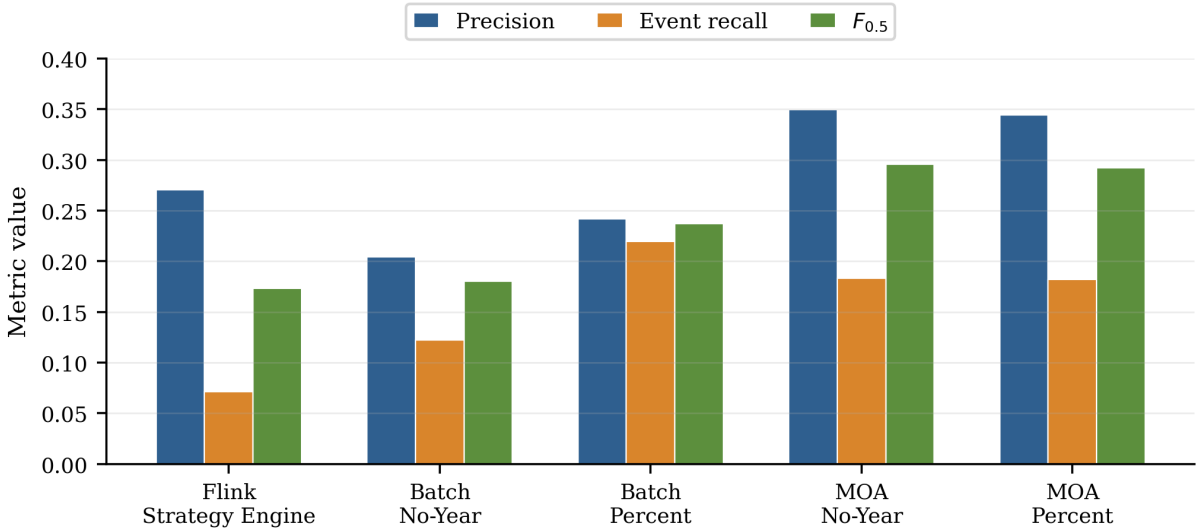

Figure 5.2: `pit_any_h2` headline comparison under the `canonical_sde_truth` universe and `clean_actionable` lens. The figure reports precision, event recall, and $F_{0.5}$ for the selected operating points of the compared systems.

The visual comparison confirms the same interpretation as Table 5.3: learned models improve the timing task over the deterministic rule baseline, with Batch Percent giving the largest event recall and MOA giving the strongest precision oriented balance.

5.5. Accounting Semantics and `pit_success_h2` Strict Contract

Before presenting the `pit_success_h2` results, the accounting semantics must be clarified since the success contract is stricter than the pit timing contract because a positive call is correct only if an eligible future pit happens and that pit is later classified as successful. This creates a distinction between decision-level quantities and event-level quantities because a system emits positive calls at driver-lap level, while the truth is defined by pit events and post-pit outcome labels. For this reason, the metrics used in this section do not all have the same denominator; the evaluation separates three quantities: strict precision, successful event coverage, and matched-pit success rate.

5.5.1. Decision-vs-Event Ledger Clarification

The first quantity is strict precision, this is the main operational precision used for the `pit_success_h2` comparison. A positive call is counted as a true positive only if it matches an eligible future pit whose outcome is successful, a positive call with no matched pit is counted as a false positive, because the system still emitted a successful-pit recommendation that did not correspond to an eligible future pit. A positive call matched to a failed or disadvantage pit is also counted as a false positive. In compact form:

$$strictPrecision = \frac{TP}{TP + FP_{no\_match} + FP_{failure}}.$$

The second quantity is successful event coverage, which is an event-level measure, it counts how many eligible successful pit events were covered by at least one positive call. Its denominator is the number of eligible successful pit events in the truth universe:

$$successfulEventCoverage = \frac{coveredSuccessfulEvents}{eligibleSuccessfulEvents}.$$

This quantity is useful because a system can have reasonable precision while still missing most successful pit opportunities.

The third quantity is matched-pit success rate, this is a diagnostic measure only. It considers only the positive calls that matched a known pit event, and then checks how many of those matched pits were successful:

$$matchedPitSuccessRate = \frac{matchedSuccess}{matchedSuccess + matchedFailure}.$$

This value is not the same as operational precision because it ignores positive calls that did not match any eligible pit, it therefore describes the quality of calls after a pit match has already occurred, but it does not describe the full operational behaviour of the system. For this reason, matched-pit success rate is not used as the main precision metric in the headline comparison.

The following sections use strict precision and successful event coverage for the main `pit_success_h2` comparison while keeping matched-pit success rate separate as a diagnostic quantity.

5.5.2. Strict `pit_success_h2` Table and Sensitivity

Table 5.4 reports the selected strict `pit_success_h2` operating points for all compared systems. For the Flink Strategy Engine, the positive call is the deterministic `PIT_NOW` label, for Batch XGBoost and MOA the positive call is obtained by applying the selected threshold to the model score.

The selected results confirm that `pit_success_h2` is substantially harder than `pit_any_h2`, the Flink Strategy Engine covers more successful events than the selected learning policies, but it does so by emitting many more positive calls, once no-match calls and failed matched pits are counted as false positives, its strict precision is low. The selected learning policies are more conservative: they emit fewer calls, obtain higher strict precision, but cover only a small fraction of successful pit events.

Among the selected learning policies, MOA Percent obtains the highest strict precision and the best $F_{0.5}$ score., MOA No-Year follows with lower coverage and lower precision, while the two Batch configurations remain more conservative in terms of emitted calls and successful event coverage.

Table 5.4: Selected strict `pit_success_h2` operating points.

System	Threshold Rule	Predicted +	Strict precision	Success coverage	$F_{0.5}$
Flink Strategy Engine	`PIT_NOW`	1235	0.080	0.083	0.081
Batch No-Year	0.37	46	0.239	0.010	0.044
Batch Percent	0.51	51	0.196	0.009	0.040
MOA No-Year	0.58	52	0.288	0.014	0.059
MOA Percent	0.58	86	0.365	0.029	0.111

The selected points are intentionally strict, they show the behaviour of each system under the frozen operating policy. However, the threshold frontier also makes it possible to inspect how the learned systems behave when the threshold is relaxed, Table 5.5 reports this sensitivity check for the Percent profiles.

The relaxed rows are diagnostic only since they are not the main headline policy since their purpose is to show the trade-off between call volume, strict precision, and successful event coverage. Lowering the threshold increases the number of emitted calls and covers more successful pit events, but it also accepts more false positives. Batch Percent increases from 51 to 844 positive calls and from 0.009 to 0.083 coverage, while precision drops from 0.196 to 0.105. MOA Percent increases from 86 to 900 positive calls and from 0.029 to 0.169 coverage, while precision drops from 0.365 to 0.205.

Table 5.5: Relaxed diagnostic sensitivity points for `pit_success_h2` Percent profiles.

System	Threshold	Predicted +	Strict precision	Success coverage	$F_{0.5}$
Batch Percent relaxed	0.22	844	0.105	0.083	0.100
MOA Percent relaxed	0.51	900	0.205	0.169	0.196

Figure 5.3 visualizes the selected strict comparison, the figure highlights the main result of this subsection: `pit_success_h2` is both rarer and stricter than pit timing, because positive calls are penalized when they do not match an eligible future pit or when the matched pit is not successful.

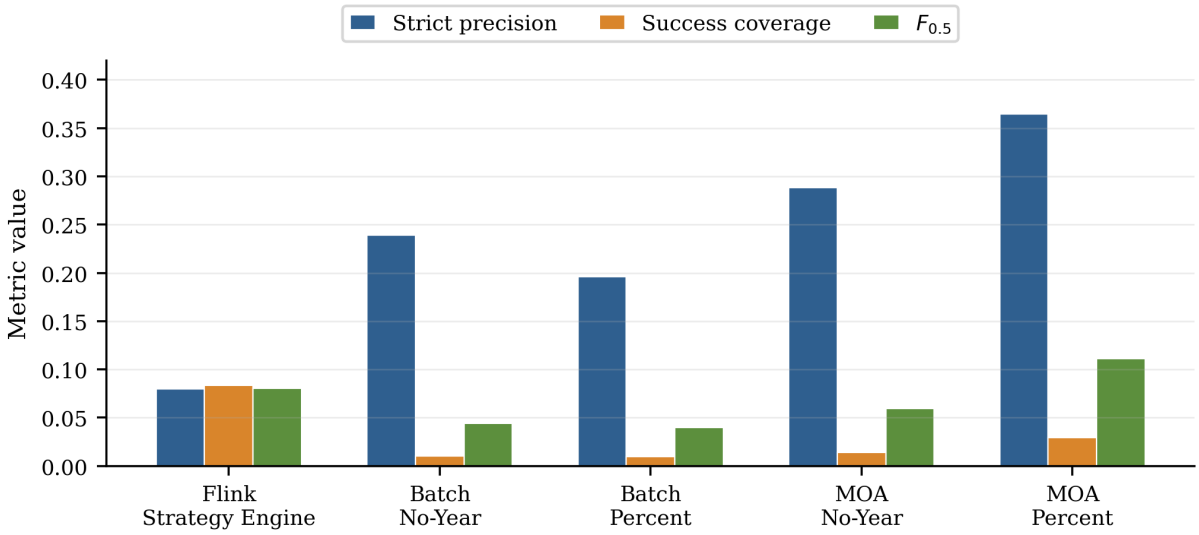

Figure 5.3: Strict `pit_success_h2` comparison under the `canonical_sde_truth` universe and `clean_actionable` lens. The figure reports strict precision, successful event coverage, and $F_{0.5}$ for the selected operating points.

5.6. Flink Strategy Engine: Diagnostic Quality vs Operational Precision

The Flink Strategy Engine also requires a separate diagnostic view, its `PIT_NOW` calls often match pit stops that are later classified as successful, but this does not mean that the engine has high operational precision under the strict `pit_success_h2` contract. The reason is the denominator, matched-pit success rate only considers calls that matched a known pit event; it ignores calls that did not match any eligible future pit. Strict precision, instead, includes those no-match calls as false positives, therefore matched-pit success rate can be high while strict operational precision remains low.

For `PIT_NOW`, the matched-pit success rate is 0.583, while strict precision is 0.080; when `GOOD_PIT` is added to the positive-call definition, the matched-pit success rate increases to 0.628, but strict precision decreases to 0.060. This means that, conditional on matching a pit, many Flink calls correspond to successful outcomes, however the engine also emits many calls that do not match an eligible future pit, and those calls reduce operational precision.

Table 5.6: Flink Strategy Engine diagnostic quality versus strict operational precision.

Positive-call definition	Matched-pit success rate	Strict precision
`PIT_NOW`	0.583	0.080
`PIT_NOW + GOOD_PIT`	0.628	0.060

Figure 5.4 visualizes the same distinction, the matched-pit success rate is useful for understanding the quality of calls after a pit match has occurred.

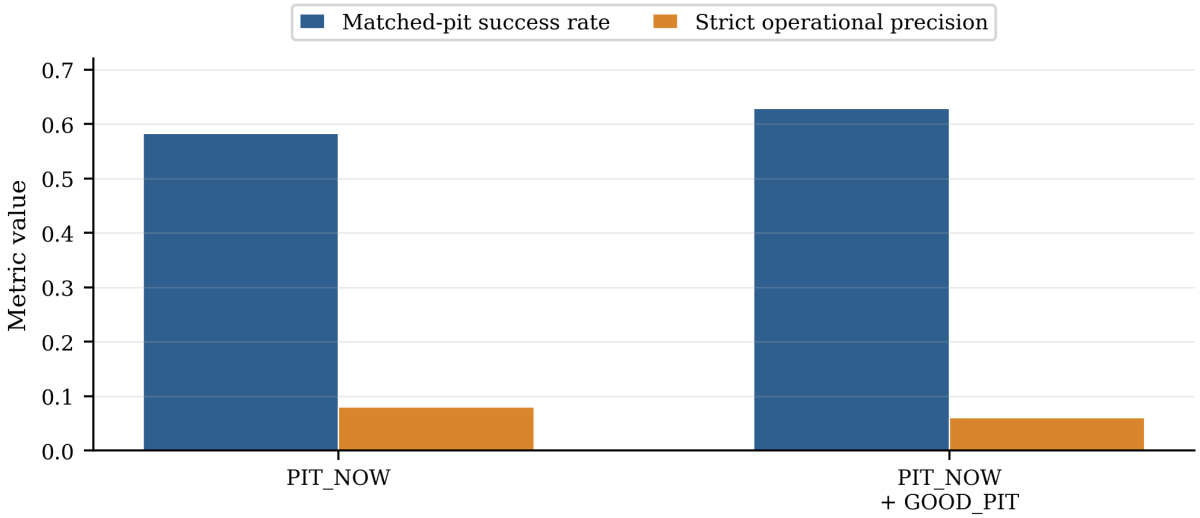

Figure 5.4: Flink Strategy Engine diagnostic matched-pit success rate compared with strict operational precision.

5.7. Cross-Task Headline Synthesis

The two headline tasks show different behaviours, the `pit_any_h2` contract is the cleaner timing task since it only asks whether a pit stop happens soon, so the systems are evaluated on their ability to recognize an approaching pit window. Under this contract, the learning-based systems improve over the deterministic Flink Strategy Engine; Batch Percent gives the largest event recall, while MOA provides the strongest precision-oriented balance.

The `pit_success_h2` contract is stricter, a correct positive call must match a future eligible pit and that pit must be classified as successful. This makes the task sparser and more sensitive to the threshold policy; Under the selected strict operating points, MOA Percent gives the cleanest successful-pit calls, with the highest strict precision and the best $F_{0.5}$ among the selected learning policies, however its successful event coverage remains low, which shows that the task is still difficult even for the best selected learner.

The Flink Strategy Engine plays a different role in the comparison because it remains the interpretable deterministic baseline and emits more calls than the selected learned policies in the success task, this gives it some successful event coverage but strict precision is low once no-match calls and failed matched pits are counted as false positives. At the same time, its matched-pit success rate is useful diagnostically: when a Flink call does match a known pit, many of those matched pits are successful.

Overall, the results suggest that pit timing is more learnable from the available race context than successful pit-call prediction. The learning branches are better at ranking and thresholding pit timing situations, while the success contract exposes the harder part of the problem: distinguishing not only when a pit will happen, but whether that future pit will also be strategically favourable under the post-pit evaluation logic.

5.8. PR Curves and Average Precision (Ranking Diagnostics)

The headline tables report selected operating points, they show what happens after each score-based system has been converted into positive calls through a threshold. Precision-recall curves provide a complementary view: they inspect how the score behaves across many possible thresholds, before committing to one operating point.

This section therefore treats PR curves and Average Precision as ranking diagnostics, they are useful for understanding whether a model assigns higher scores to positive rows than to negative rows, and whether a different threshold could expose a better precision-recall trade-off.

5.8.1. Diagnostic framing

Precision-recall curves are computed at row level for score based systems, Batch XGBoost produces probability scores, while MOA produces vote-based scores through the custom logger described in Chapter 4. By sweeping a threshold over these scores, each system produces a curve showing how precision and recall change as the positive-call policy becomes more or less aggressive [12, 19, 33].

Average Precision summarizes this row-level ranking behaviour into a single value, a higher AP means that positive rows tend to receive higher scores than negative rows. However AP is not the same as strict operational precision; in particular, for `pit_success_h2`, the headline comparison must still count no-match calls and failed matched pits as false

positives.

The Flink Strategy Engine is different since it does not produce a continuous learned score used for threshold sweeping in the same way as Batch and MOA. It emits deterministic labels such as `MONITOR`, `GOOD_PIT`, and `PIT_NOW`; therefore in the PR figures the Flink Strategy Engine is best interpreted as a fixed operating point rather than as a learned score curve.

The purpose of this section is to inspect whether the learned systems rank the rows in a useful way, the comparison remains based on the selected thresholds and on the strict accounting rules defined earlier.

5.8.2. PR Figures and AP Table

Figures 5.5 and 5.6 show the precision–recall curves for the two targets, these plots compare the score-based systems across thresholds, so they are useful for inspecting ranking behaviour before selecting an operating point. The Flink Strategy Engine is shown as a fixed operating point, since it emits deterministic labels rather than a continuous learned score curve.

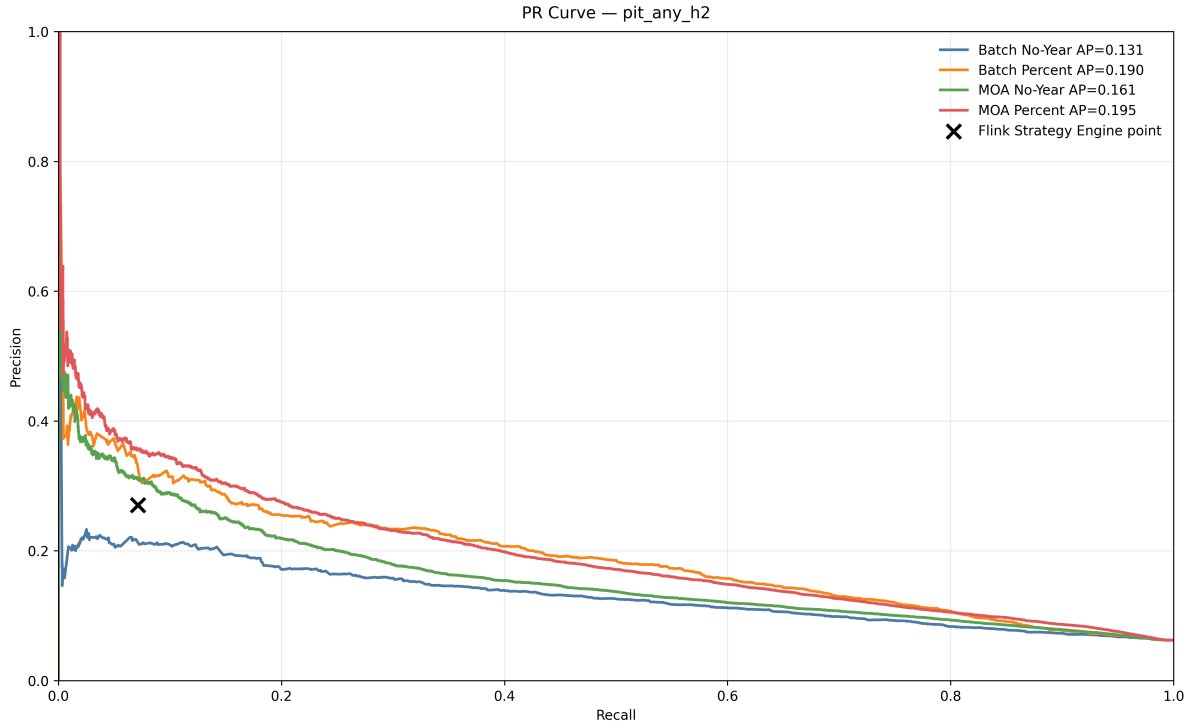

Figure 5.5: Precision–recall curve for `pit_any_h2`.

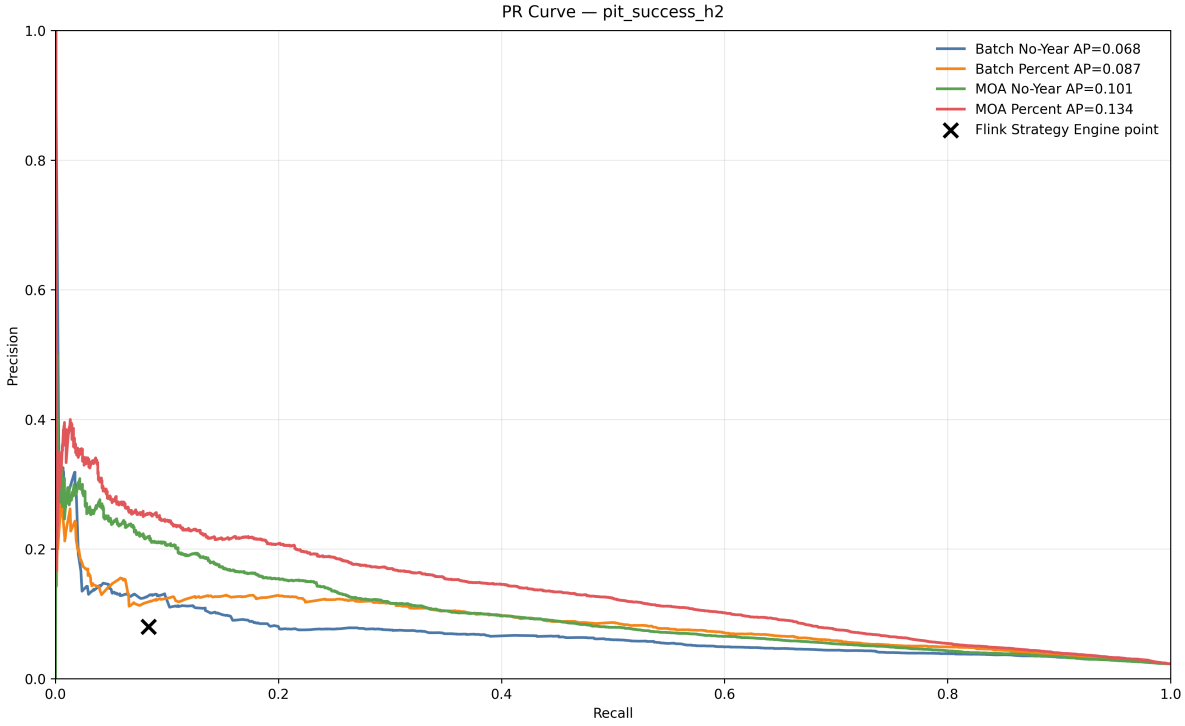

Figure 5.6: Precision–recall curve for `pit_success_h2`.

Table 5.7 reports Average Precision for the score-based systems, for both targets MOA Percent obtains the highest AP. On `pit_any_h2`, Batch Percent is close to MOA Percent, which is consistent with the strong timing behaviour already observed in the selected operating points. On `pit_success_h2`, the AP values are lower overall, confirming that the success task is harder to rank as well as harder to classify under a fixed threshold.

Table 5.7: Average Precision for score-based systems. AP is a row-level ranking diagnostic.

Target	Batch No-Year	Batch Percent	MOA No-Year	MOA Percent
`pit_any_h2`	0.131	0.190	0.161	0.195
`pit_success_h2`	0.068	0.087	0.101	0.134

The AP table should be read together with the headline metrics since a model can have a useful ranking curve while still requiring a conservative threshold to obtain acceptable operational precision. This is especially visible for `pit_success_h2`, where the best AP belongs to MOA Percent, but the selected operating point still has low successful event coverage.

5.9. Support Aware Race Level Diagnostics

The headline metrics aggregate all races into a single comparison, which is useful for ranking the systems, but it can hide whether performance is stable across races or concentrated in a smaller set of events. The race level diagnostics inspect this variability by computing Kappa and G-Mean separately for each race.

Kappa measures agreement with the target after accounting for agreement that could happen by chance while G-Mean combines sensitivity and specificity, so it is useful when the positive class is rare [11, 19].

A support filter is applied before plotting the race-level values since races with too few positives can produce unstable diagnostic scores, especially for sparse targets. The plots therefore keep only races with at least three positive examples. For `pit_any_h2`, this keeps 92 races for the Flink Strategy Engine, MOA No-Year, and MOA Percent, and 91 races for the Batch profiles. For `pit_success_h2`, the support-aware set is smaller: 87 races for the Flink Strategy Engine and MOA profiles, and 86 races for the Batch profiles.

5.9.1. `pit_any_h2`: Kappa and G-Mean per race

Figures 5.7 and 5.8 report race-level diagnostics for the `pit_any_h2` task where each point corresponds to one race after the support filter, the dashed line shows the average value for the corresponding system.

The Kappa plot shows that the timing task has visible race-to-race variability, the Flink Strategy Engine remains close to zero in many races, which is consistent with its low event recall in the headline comparison. Batch Percent reaches higher Kappa values more often, but it also shows larger oscillations across races. MOA No-Year and MOA Percent are more concentrated around moderate positive values, suggesting a more stable ranking-to-decision behaviour across the evaluated races.

The G-Mean plot gives a similar view from the sensitivity-specificity side, Batch Percent has the strongest race-level G-Mean values in many races, which matches its high event recall on `pit_any_h2`. The MOA profiles appear more stable but less aggressive, while the Flink Strategy Engine remains limited by the small number of positive calls it emits; these plots support the headline interpretation: pit timing is learnable, but the balance between coverage and precision changes substantially from race to race.

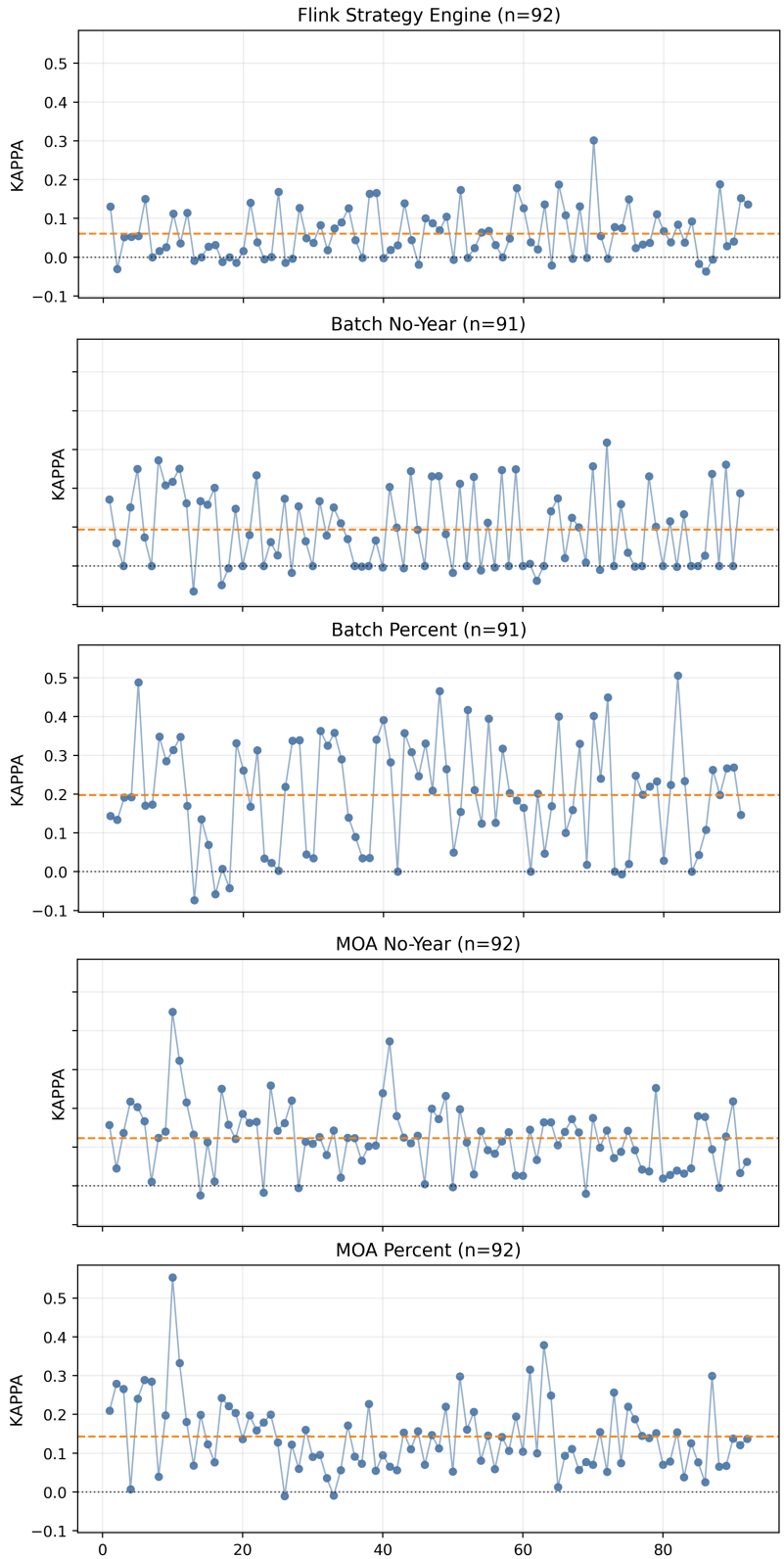

Figure 5.7: Race-level Kappa diagnostic for `pit_any_h2` under the support filter of at least three positives per race.

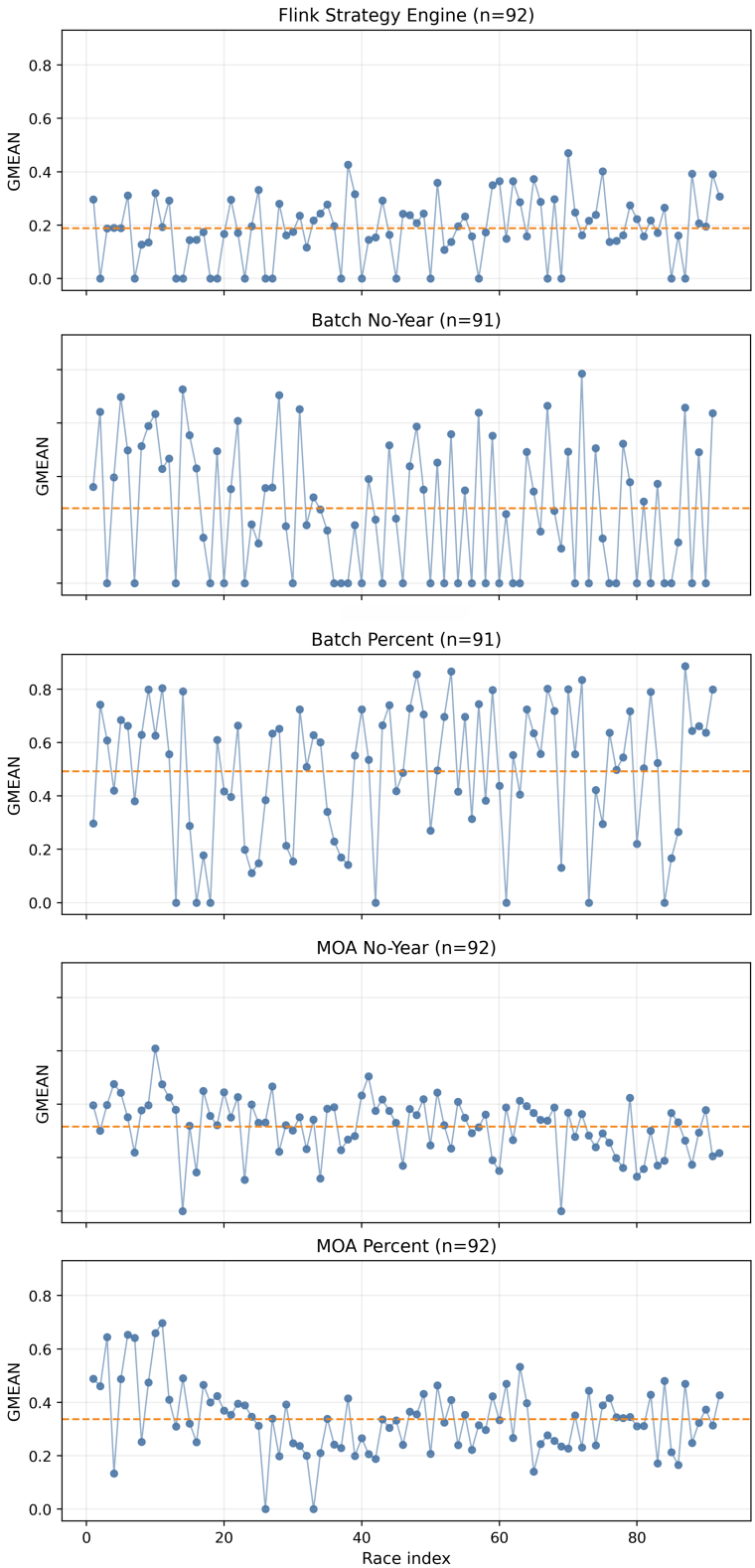

Figure 5.8: Race-level G-Mean diagnostic for `pit_any_h2` under the support filter of at least three positives per race.

5.9.2. `pit_success_h2`: Kappa and G-Mean per race

Figures 5.9 and 5.10 report the same race-level diagnostics for the stricter `pit_success_h2` task, with the support filter again set to at least three positive examples per race. After this filter, the diagnostic set contains 87 races for the Flink Strategy Engine and the MOA profiles, and 86 races for the Batch profiles.

The plots are much sparser than the corresponding `pit_any_h2` diagnostics, this follows from the definition of `pit_success_h2`: a positive case requires not only a future pit, but a future pit that is also classified as successful. Many races therefore contain very few positive success examples, and the per-race diagnostic metrics often remain close to zero.

The Kappa plot shows this sparsity clearly, where Batch No-Year and Batch Percent have many races near zero, with only isolated positive spikes; the MOA profiles show more non-zero episodes, but the values remain low for many races, the Flink Strategy Engine also shows scattered positive values, which reflects its larger number of emitted calls, but the values are still unstable across races. These patterns are consistent with the strict success results reported earlier: the task is difficult, and most systems cover only a small fraction of successful pit opportunities.

The G-Mean plot gives the same message from the sensitivity-specificity perspective. Some races contain visible positive peaks, especially for the Flink Strategy Engine and the MOA profiles, but many races remain at or near zero. The diagnostic therefore suggests that successful pit-call prediction is both harder in aggregate and less stable at race level.

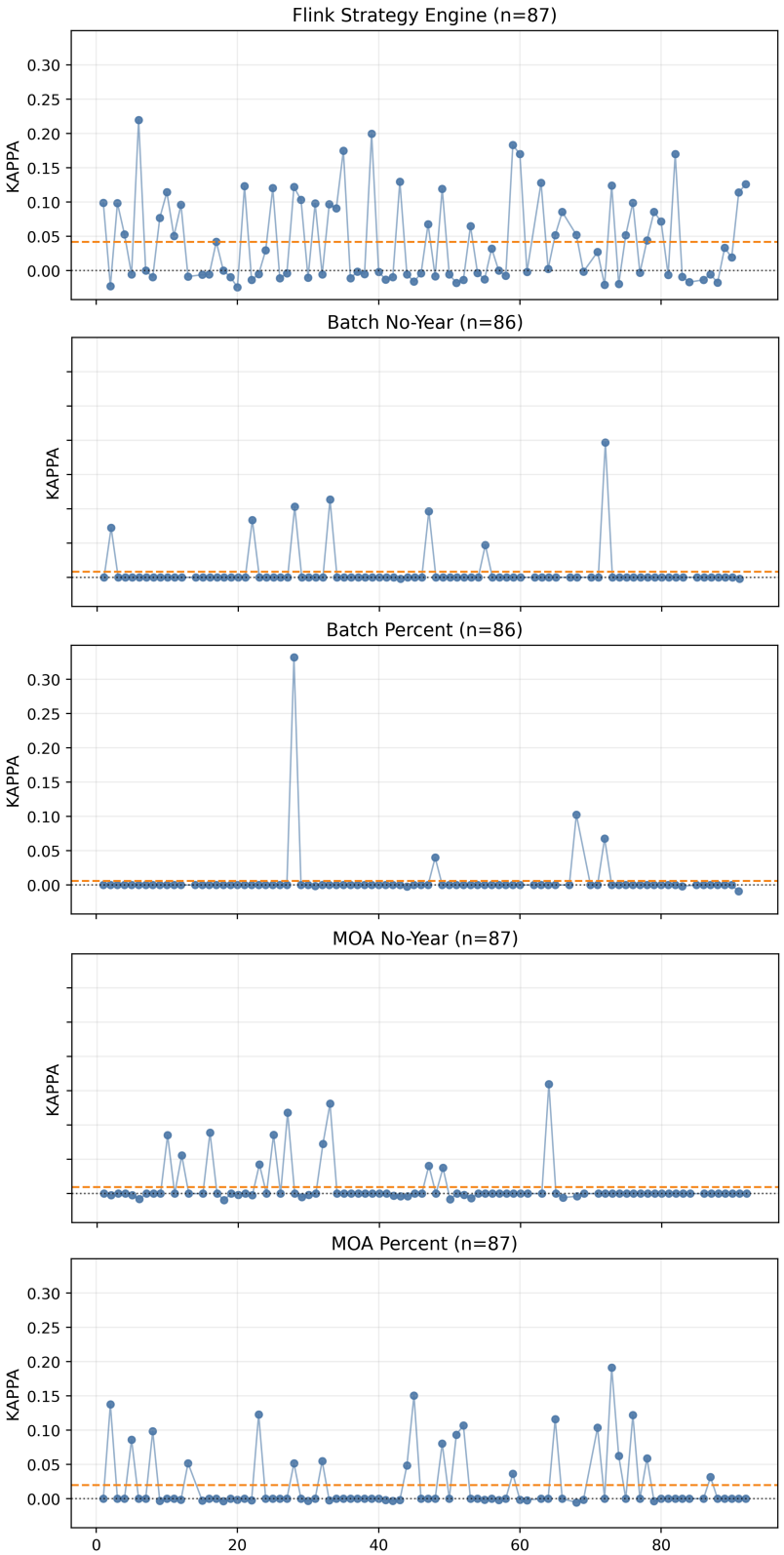

Figure 5.9: Race-level Kappa diagnostic for `pit_success_h2` under the support filter of at least three positives per race.

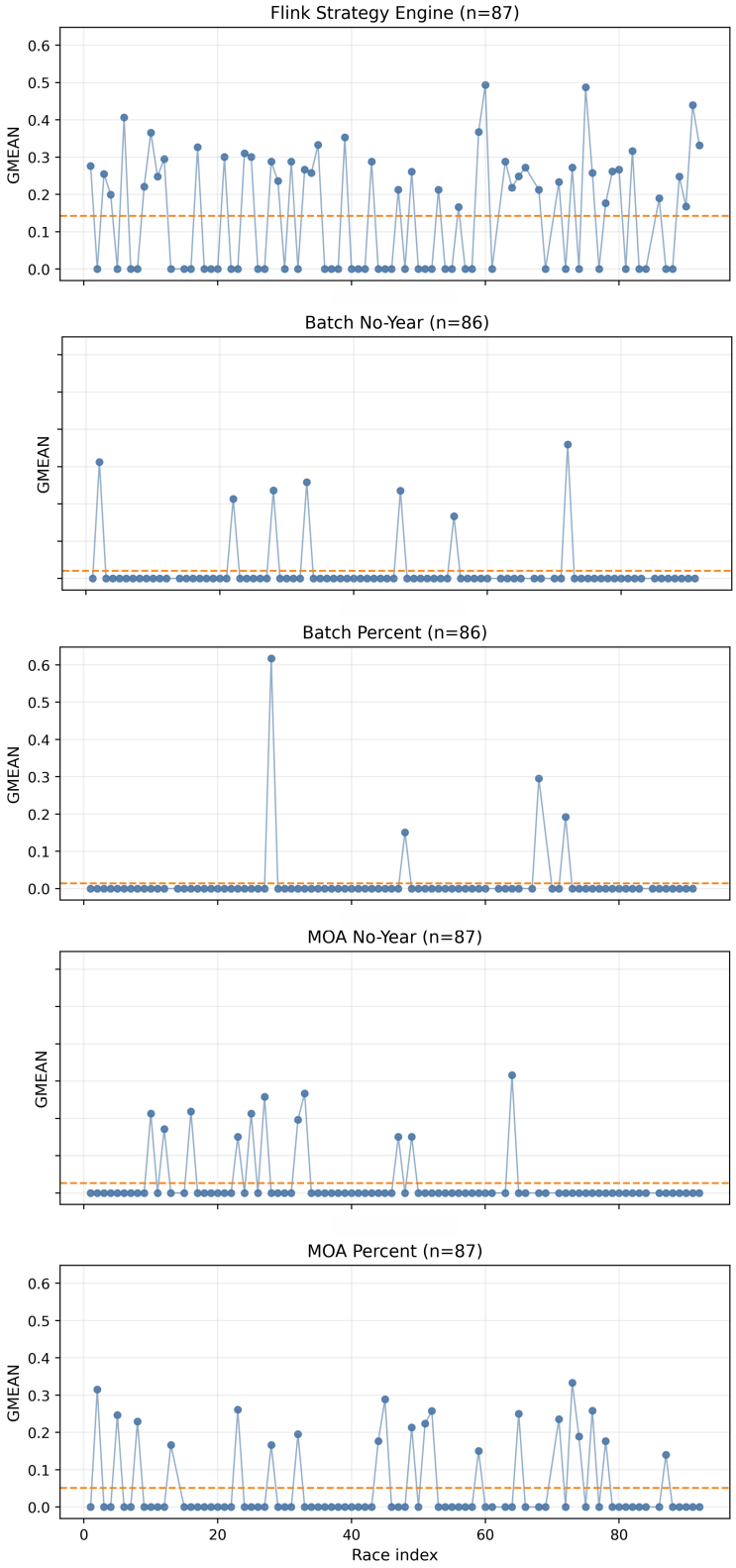

Figure 5.10: Race-level G-Mean diagnostic for `pit_success_h2` under the support filter of at least three positives per race.

5.10. Explainability as Methodology Debugging

Explainability is used in this evaluation as a supporting diagnostic where the objective is to inspect whether the systems rely on plausible race context signals, and to detect possible shortcut features that could make the comparison less reliable.

5.10.1. Explainability role across systems

The three decision systems have different explainability profiles; the Flink Strategy Engine is interpretable by construction, because its recommendations come from explicit score components, gates, and promotion rules, its behaviour can be inspected by following the rule path that led to a `MONITOR`, `GOOD_PIT`, or `PIT_NOW` label.

Batch XGBoost is inspected through SHAP and TreeSHAP on the trained tree models, this gives feature attribution values for the learned scores and helps identify which variables contribute most to the model output [10, 24, 25]. The analysis is direct in this case, because TreeSHAP is applied to the same model family used in the Batch branch.

The MOA Online Learner is harder to inspect directly since the model is executed as a streaming learner and the implementation does not expose the same type of native feature attribution available for XGBoost. A surrogate SHAP analysis was explored as a possible proxy, but it was not retained in the reported explainability results because the surrogate model is a different implementation and its attributions cannot be verified as faithful explanations of the internal MOA learner; for this reason the explainability discussion reports MOA behaviour through its scores, threshold diagnostics, and race-level metrics, rather than through feature attribution.

Explainability is therefore used mainly for the components where the explanation path is directly traceable, it helps check whether the feature matrix is reasonable, whether some variables behave as shortcuts, and whether the engineered profiles change the information available to the learner.

5.10.2. `source_year` shortcut-risk and No-Year correction

One of the main findings from the Batch explainability analysis concerned `source_year`, this field was useful as metadata for ordering races, reconstructing out-of-fold predictions, and tracing results back to seasons. As a model input, however, it created a shortcut risk; if a learner assigns high importance to `source_year`, part of the prediction may depend on season identity rather than on race context signals such as tyre life, gaps, pace, and

race progress [6].

This observation motivated the No-Year correction described in Chapter 4. In the No-Year profile, `source_year` is kept for metadata and splitting logic, but it is removed from the learner feature matrix, this makes the comparison cleaner because the model cannot use the season identifier directly.

The Percent profile goes one step further, it also removes selected shortcut-prone fields and adds race-relative variables, such as race progress and relative lap-time signals. This profile is useful for testing whether a more normalized representation improves learning, but it is not a fully neutral baseline since it is an engineered profile, No-Year remains the cleaner reference for comparing the effect of removing season identity.

Figure 5.13 show the Batch explainability evidence used during this correction phase. The beeswarm plots in figure 5.11 and 5.12 compare the attribution patterns of the original and Percent style Batch configurations. The feature comparison plot summarizes how the most relevant variables change after the feature contract is adjusted, these figures support the methodological decision to remove `source_year` from the final learner input and to keep No-Year as the clean reference profile.

The explainability figures therefore close the loop with the implementation corrections introduced earlier, they explain why the final protocol distinguishes No-Year from Percent and why the season identifier is excluded from the learner feature matrix.

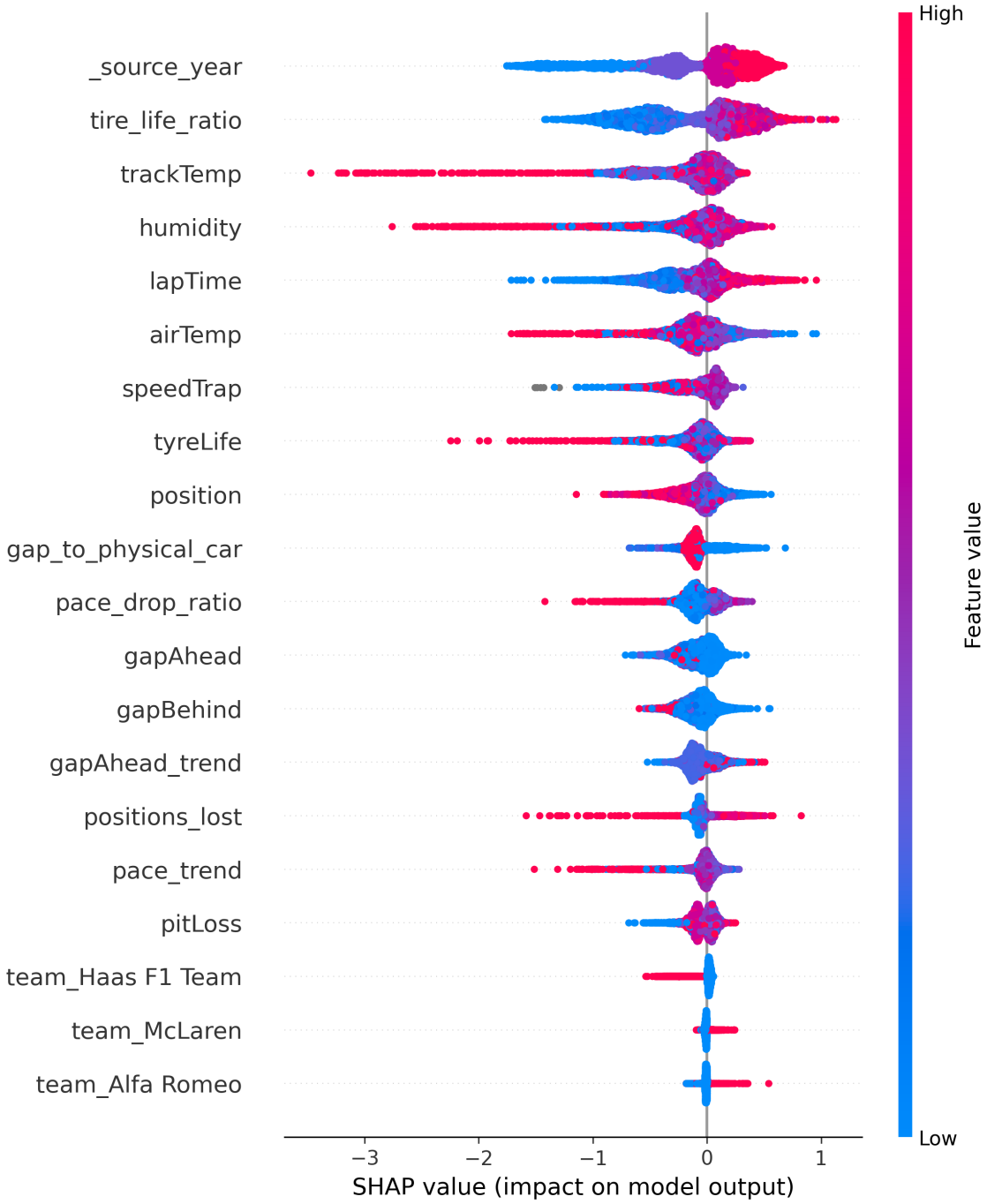

Figure 5.11: Batch original feature view.

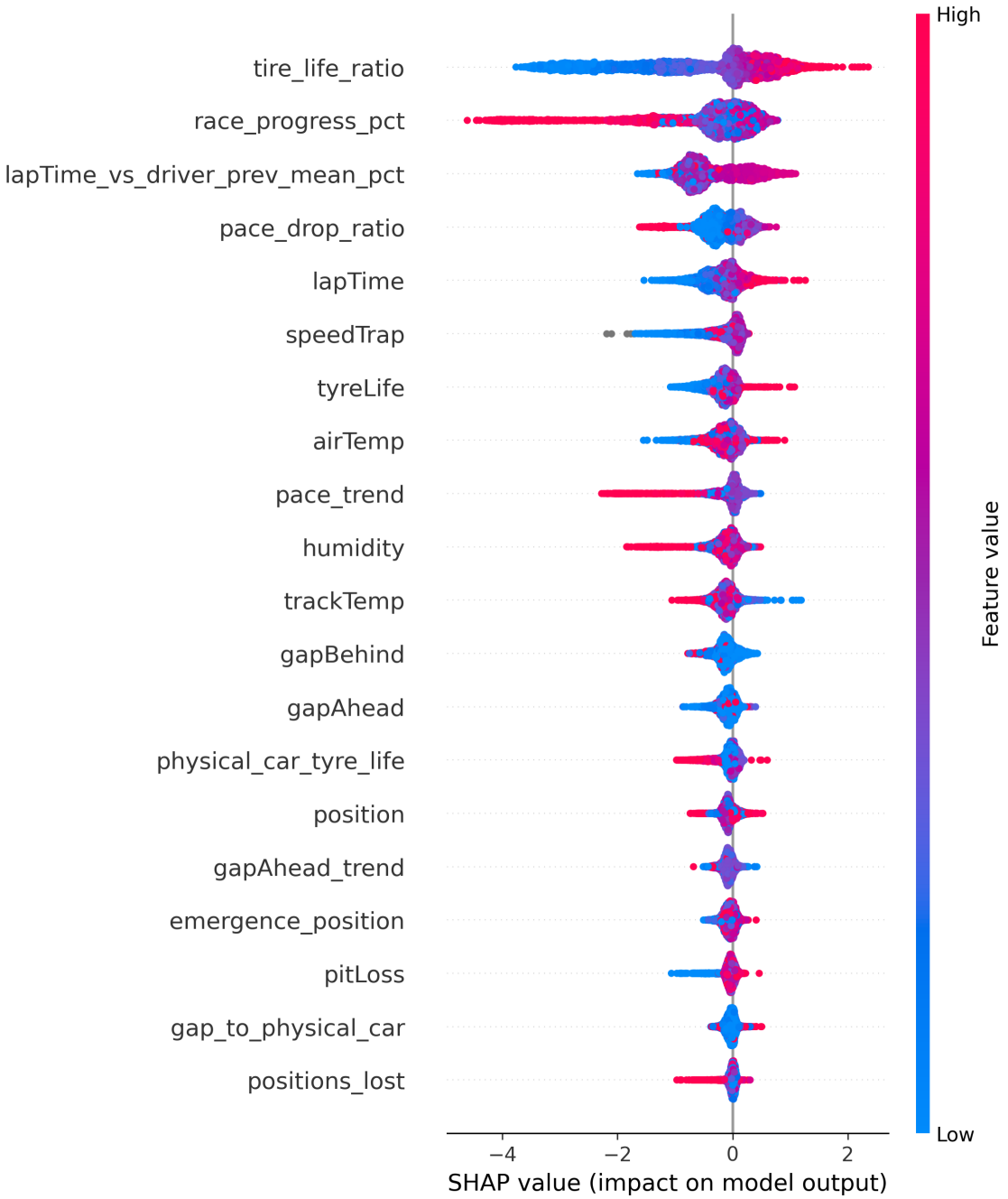

Figure 5.12: Batch Percent feature view.

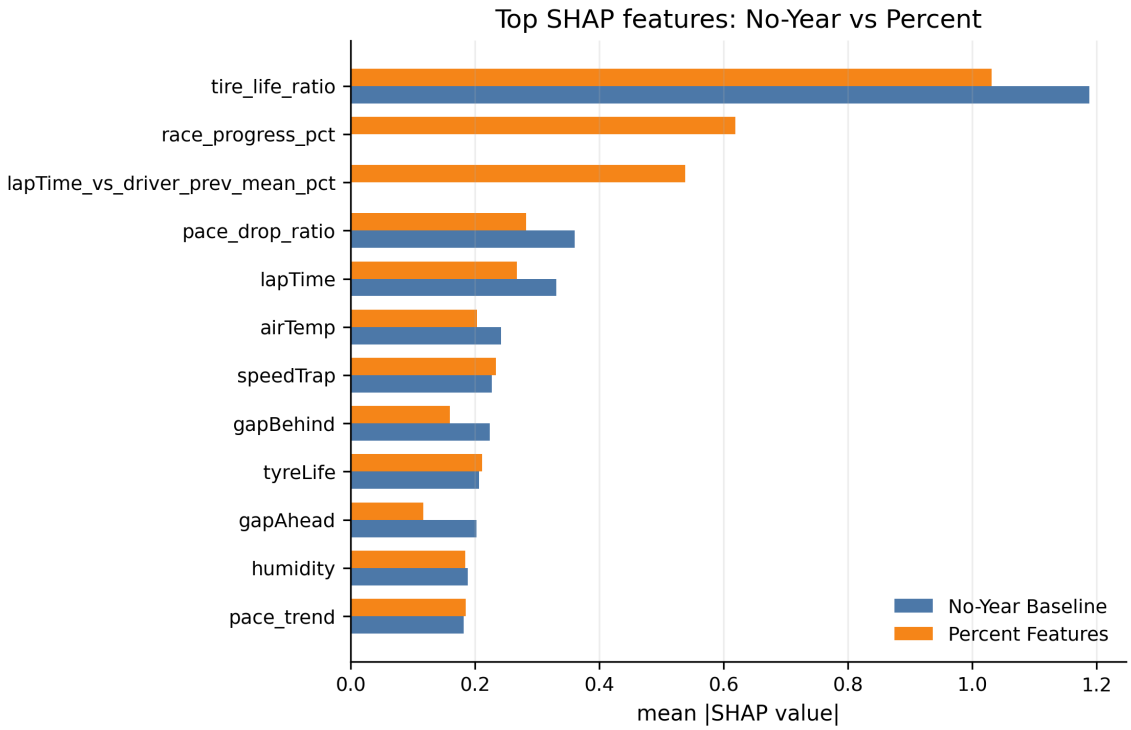

Figure 5.13: Top Batch SHAP features before and after the Percent profile correction.

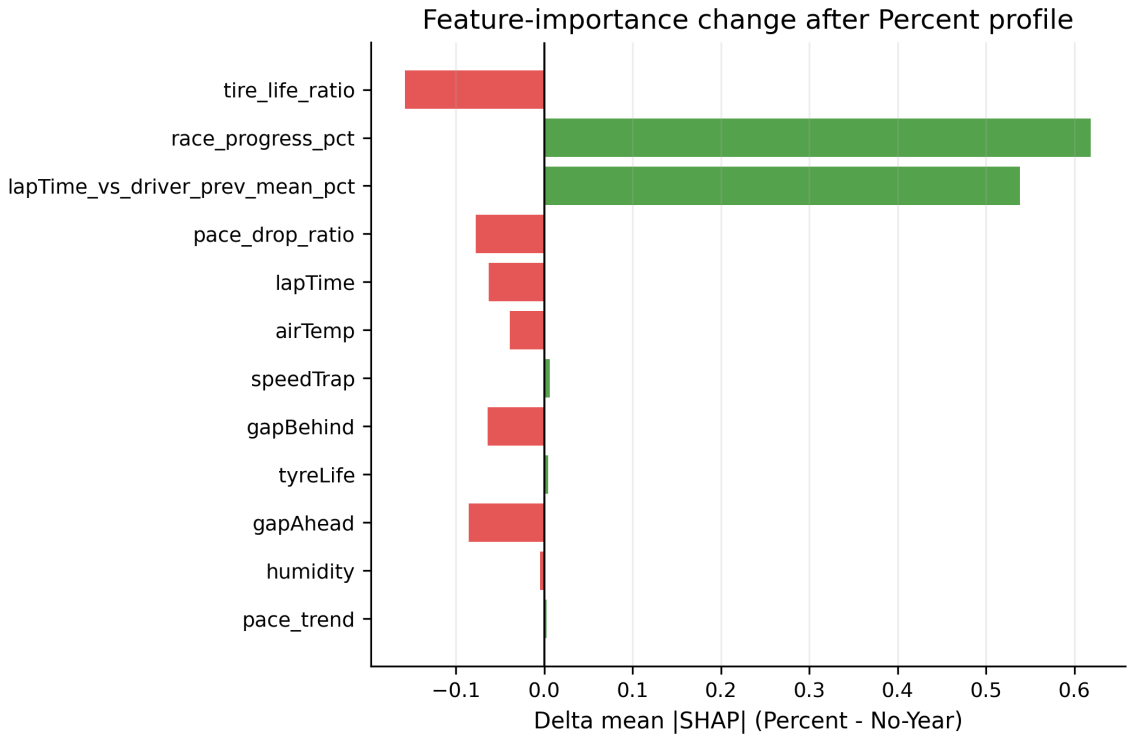

Figure 5.14: Change in top Batch SHAP feature importance after the Percent profile correction.

5.11. Conclusive Summary

This chapter evaluated the three decision paradigms under the same frozen protocol, the comparison used the `canonical_sde_truth` universe, the `clean_actionable` lens, and the two contracts defined earlier: `pit_any_h2` for pit timing and `pit_success_h2` for successful pit-call prediction.

The runtime comparison showed the expected difference between the learning branches, Batch XGBoost is slower because it is evaluated through race-sequential out-of-fold training, while MOA processes the stream in a single prequential pass, this clarifies the practical cost of the implemented evaluation protocols.

For `pit_any_h2`, the results show that pit timing is the cleaner and more learnable task, the learning-based systems improve over the deterministic Flink Strategy Engine, with Batch Percent reaching the largest event recall and MOA providing the strongest precision-oriented balance. The row-level imbalance explains why absolute precision remains moderate, but the selected operating points are still above the naive positive-rate baseline.

For `pit_success_h2`, the comparison becomes stricter, a correct call must match a future eligible pit and the pit must later be classified as successful, under this contract MOA Percent gives the cleanest selected successful-pit calls, but coverage remains low. The relaxed threshold analysis confirms that more coverage is possible, although it comes with lower strict precision.

The Flink Strategy Engine remains useful as an interpretable deterministic baseline, its matched-pit success rate shows that many matched calls correspond to successful outcomes, but strict operational precision is much lower once no-match calls and failed matched pits are counted as false positives. This confirms why matched-pit success rate has to remain a diagnostic quantity rather than the main precision metric.

The diagnostic sections complete the interpretation, PR curves and Average Precision inspect row-level ranking quality, race-level Kappa and G-Mean show support-sensitive variability across races, and SHAP-based explainability helped identify the `source_year` shortcut risk that motivated the No-Year correction. These diagnostics explain why the final protocol separates selected operating points, relaxed threshold sensitivity, and methodological debugging.

Overall, the evaluation shows that the proposed pipeline can compare deterministic stream rules, Batch learning, and online learning under shared contracts. Pit timing benefits more clearly from learning, while successful pit-call prediction remains sparse, threshold-

sensitive, and harder to cover reliably. The next chapter uses these results to answer the thesis objectives and to discuss the main limitations and future extensions of the work.

6 | Conclusions and Future Work

6.1. Conclusions

This thesis addressed the problem of evaluating pit decision support systems from historical Formula 1 data under live-like temporal constraints, the implemented system starts from historical race sessions, converts them into replayable event streams, processes them through a Flink based strategy pipeline, and then compares three decision paradigms under shared evaluation contracts: the Flink Strategy Engine, Batch XGBoost, and the MOA Online Learner.

A pit recommendation is emitted at a specific driver-lap, while its correctness can only be assessed later, after the race has evolved and the pit event, if any, has occurred. The evaluation therefore had to connect decision rows, future pit events, and post pit outcome labels without using future information as model input. This led to the two contracts used throughout the thesis: `pit_any_h2`, for pit timing, and `pit_success_h2`, for successful pit-call prediction.

The final evaluation showed that these two contracts describe different levels of difficulty, where Pit timing can be learned from replayed race context with useful precision and coverage trade-offs, while Successful pit-call prediction can also be evaluated under a strict operational view, but it remains sparse, threshold sensitive, and harder to cover reliably. The following findings summarize the main outcomes of the work.

6.1.1. Principal Findings

1. **Aligned contracts are necessary for a fair comparison.**

   The experiments showed that the three systems cannot be compared directly from their native outputs. The Flink Strategy Engine emits rule based labels, Batch XGBoost emits out-of-fold scores, and MOA emits vote based scores in prequential order. A meaningful comparison required a shared truth universe, explicit truth lenses, leakage filtering, and a clear separation between row-level decisions and event-

level coverage. These elements are part of the contribution of the thesis, because they define the conditions under which deterministic rules, Batch learning, and online learning can be judged on the same evidence.

2. `pit_any_h2` **is the cleaner learnable task.**

   The pit timing contract gave the clearest learning signal. It checks whether an eligible pit stop happens within the evaluation horizon, without asking whether that pit is later classified as strategically successful. Under this contract, the learning based systems improved over the deterministic Flink Strategy Engine. Batch Percent reached the largest event recall, while MOA gave the strongest precision oriented balance. The task remains imbalanced, but the results indicate that race context features such as pace, tyre life, gaps, and race progress contain usable signal for predicting that a pit window is approaching.

3. `pit_success_h2` **is stricter and policy-sensitive.**

   The success contract was substantially harder. A positive call is correct only when it matches a future eligible pit and the post pit evaluator classifies that pit as successful. This reduces the amount of clean positive evidence and makes the selected operating threshold much more influential. MOA Percent produced the cleanest selected successful pit calls, but its successful event coverage remained low. The relaxed threshold analysis confirmed the expected trade-off: coverage can be increased, but only by accepting more false positives under the strict operational view.

4. **The Flink Strategy Engine remains useful as an interpretable baseline.**

   The rule based system did not dominate the strict precision results, but it provided an interpretable reference point. Its decisions can be followed through explicit score terms, gates, and promotion rules. The evaluation also showed why diagnostic matched pit quality must be separated from operational precision. When a Flink call matches a known pit, many of those matched pits correspond to successful outcomes. Once no-match calls and failed matched pits are counted as false positives, strict precision becomes much lower. This distinction prevents the deterministic baseline from being overstated and keeps the comparison with Batch and MOA fair.

Taken together, these findings answer the two main contract questions of the thesis; pit timing is a learnable decision support task in the implemented replay setting, successful pit-call prediction can be evaluated under a stricter contract, but the results show a more limited and fragile signal. The strongest conclusion is that each paradigm occupies a different position in the trade-off between interpretability, coverage, precision, and temporal

evaluation.

6.2. Scope and Limitations

The results should be read within the scope of the implemented pipeline; the first boundary is the use of historical replay, replaying FastF1 sessions through Kafka and Flink makes the experiments reproducible and gives the system live-like temporal semantics, the same race can be replayed multiple times, the produced artifacts can be inspected, and the comparison can be regenerated. A live race feed would introduce additional issues, such as data corrections, missing fields, feed latency, and operational failure modes. The current system is therefore a controlled experimental environment rather than a deployed pit-wall tool usable in production for real world applications.

A second boundary concerns the definition of successful pit outcomes, the `pit_success_h2` target depends on the `PitStopEvaluator`, which classifies completed pit cycles from post pit evidence. This gives the thesis a reproducible truth contract, but it does not represent the full utility function of a racing team because a real strategy decision may also depend on tyre allocation, risk tolerance, team objectives, opponent response, long-term stint plans, or championship context. The evaluator captures part of the strategic outcome, mainly through gap and rival evolution, but it remains an heuristic approximation.

The strict success contract also creates a support problem, successful pit events are rarer than pit timing events, and some races contain only a small number of clean positive cases. Race-level Kappa and G-Mean diagnostics therefore become noisy for `pit_success_h2`, even after support filtering. This sparsity limits how strongly per race behaviour can be interpreted.

Threshold choice is another limitation since Batch XGBoost and MOA produce scores, and the final operational behaviour depends on where the threshold is placed. The selected thresholds define the headline policies, while the relaxed rows show how quickly the balance between precision and coverage can change. A fixed threshold is useful for reproducibility, but it cannot represent every possible operational preference.

The MOA branch adds two further boundaries; its positive scores are derived from class votes and are treated as ranking scores, they support threshold sweeps and precision–recall diagnostics, but they should not be read as direct probabilities of a pit event. The same caution applies to explainability, Batch XGBoost can be inspected directly through SHAP and TreeSHAP, while the MOA Online Learner does not expose an equally direct attribution path. A surrogate SHAP analysis was explored, but it was not retained

because the surrogate model could not be verified as a faithful explanation of the internal online learner.

Finally, the model comparison is deliberately scoped; the Batch branch uses XGBoost and the online branch uses the selected MOA configuration. This is enough to evaluate the proposed framework across deterministic, Batch, and online paradigms, but it does not exhaust the space of possible learners. Other Batch models, online learners, calibration methods, and drift-aware algorithms may produce different trade-offs under the same contracts.

These limitations define what the comparison actually supports: a reproducible evaluation of three decision paradigms on historical replay data, under explicit contracts and controlled accounting rules. They also indicate where future work should extend the system.

6.3. Future Work

The implemented pipeline provides a foundation for several extensions, the most relevant ones follow directly from the limitations described above.

6.3.1. From Historical Replay to Live Ingestion

The first extension is the transition from replay to live ingestion, the current architecture already separates event sourcing, Kafka streaming, Flink processing, persisted artifacts, and downstream evaluation. A live version would replace the historical replay source with a real timing or telemetry feed, while keeping the same stateful processing logic where possible.

This step would test a different part of the system, in replay the input sequence is frozen and can be regenerated while in a live setting, events may arrive late, be corrected, or be partially unavailable. The system would need stronger monitoring, latency checks, and failure handling, also the evaluation would also have to distinguish between decisions made with incomplete live data and decisions made after corrected data become available.

6.3.2. Richer Truth Models for Successful Pit Outcomes

The second direction concerns the successful pit truth model. The current `PitStopEvaluator` provides a deterministic classification of completed pit cycles, which was necessary for this thesis, a richer evaluator could use longer post pit windows, better rival tracking,

tyre compound plans, and more detailed traffic evolution. It could also include sampled human validation, especially for ambiguous pit cycles where the rule based classification is less clear.

Another possible development is a probabilistic success label, instead of assigning only success, failure, or unknown categories, the evaluator could estimate the confidence of the outcome; this would help in cases where the gap change is small, the rival context changes after the stop, or safety car and weather conditions distort the comparison.

6.3.3. Event Level and Sequence Aware Learning

The current learning setup uses driver lap rows, this representation is simple and works well with tabular learners, but part of the evaluation is event level. One future direction is to train models more directly for event coverage, rather than treating each driver lap row as an independent binary example.

Several formulations are possible, for instance the model could predict time to pit, rank pit opportunities within a short horizon, or optimize event coverage under a false positive budget. Sequence aware models could also use recent lap history directly, instead of relying only on the engineered state available in the current row. This direction is especially relevant for `pit_success_h2`, where the signal is sparse and isolated row predictions may not capture the full strategic episode.

6.3.4. Adaptive Threshold and Risk Aware Policies

The threshold frontier showed that the same score model can behave differently depending on the selected operating point, future work could replace fixed thresholds with adaptive policies. A conservative threshold may be appropriate during stable race phases, while a lower threshold may be useful when a rival has just pitted, when a safety car appears, or when tyre degradation is rapidly increasing. Risk-aware policies could expose different operating modes, for instance a monitoring mode could favour coverage and emit early warnings, a recommendation mode could require higher precision before producing an actionable call. This would make the output more flexible and closer to the way a strategy engineer might use a decision support tool.

6.3.5. Broader Online Learning and Calibration Aware Evaluation

The MOA Online Learner was evaluated through one selected configuration, future work could compare additional online learners, including Hoeffding Trees, Adaptive Random Forests, and other drift aware ensembles. This would clarify whether the observed MOA behaviour depends on the selected learner or whether the same pattern appears across different online methods.

6.3.6. Faithful Explainability for Online Learners

The final direction concerns explainability for online models, since the Batch branch has a direct explanation path through tree based attribution, while the MOA branch does not; a future version could investigate explanation methods designed for streaming learners, or log additional interpretable signals during prequential prediction. The goal would be to explain online decisions without replacing the learner with a separate surrogate model, possible approaches include local perturbation diagnostics, model specific explanations when available, or rule summaries extracted from online behaviour. This would make the online branch easier to inspect and more comparable with the interpretability of the Flink Strategy Engine and Batch XGBoost.

6.4. Final Remarks

The thesis developed a pipeline for comparing pit decision systems from historical race replay to stream processing, learning, and evaluation. The main outcome is a structured comparison where pit timing showed the clearest learning gains. Successful pit call prediction remained more difficult, mainly because success is rarer, evaluator dependent, and sensitive to the operating threshold.

The work also showed that evaluation design matters as much as model choice in this setting, a matched pit diagnostic can suggest good conditional behaviour while strict operational precision remains low. A score can rank rows well while still requiring a conservative threshold. A feature can improve apparent performance while acting as a shortcut. These effects are part of the pit decision problem and have to be exposed.

The final contribution is therefore both practical and methodological; practically, the system provides an executable path from historical races to comparable decision artifacts, methodologically, it defines contracts and accounting rules that make deterministic stream

rules, Batch learning, and online learning comparable under the same evidence.

Bibliography

[1] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. Fernandez-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, 8(12):1792–1803, 2015. doi: 10.14778/2824032.2824076.

[2] S. Arora, R. Rani, and N. Saxena. A systematic review on detection and adaptation of concept drift in streaming data using machine learning techniques. *WIREs Data Mining and Knowledge Discovery*, 14(1):e1536, 2024. doi: 10.1002/widm.1536.

[3] A. Bifet and R. Gavalda. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 443–448, 2007. doi: 10.1137/1.9781611972771.42.

[4] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. Moa: Massive online analysis. *Journal of Machine Learning Research*, 11:1601–1604, 2010.

[5] G. W. Brier. Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78(1):1–3, 1950. doi: 10.1175/1520-0493(1950)078<0001:VOFEIT>2.0.CO;2.

[6] G. Brookshire, J. Kasper, N. M. Blauch, Y. C. Wu, R. Glatt, D. A. Merrill, S. Gerrol, K. J. Yoder, C. Quirk, and C. Lucero. Data leakage in deep learning studies of translational eeg. *Frontiers in Neuroscience*, 18:1373515, 2024. doi: 10.3389/fnins.2024.1373515.

[7] F. Cappello and A. Hoegh. A state-space approach to modeling tire degradation in f1 racing. arXiv preprint arXiv:2512.00640, 2025. URL `https://arxiv.org/abs/2512.00640`.

[8] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache flink: Stream and batch processing in a single engine. *IEEE Data Engineering Bulletin*, 38(4):28–38, 2015.

[9] O. F. Carrasco Heine and C. Thraves. On the optimization of pit stop strategies via dynamic programming. *Central European Journal of Operations Research*, 31: 239–268, 2023. doi: 10.1007/s10100-022-00806-4.

[10] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016. doi: 10.1145/2939672.2939785.

[11] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960. doi: 10.1177/001316446002000104.

[12] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 233–240, 2006. doi: 10.1145/1143844.1143874.

[13] D. H. Dinh, T. M. Tran, T. Reichert, and T. Ametsbichler. A gradient boosting analytics framework for predictive pit stop decision making in formula one racing. SSRN preprint, 2025. URL `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=6271405`. SSRN 6271405.

[14] C. Elkan. The foundations of cost-sensitive learning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 973–978, 2001. URL `http://cseweb.ucsd.edu/~elkan/rescale.pdf`.

[15] M. Fowler. Event sourcing. Online article, 2005. URL `https://martinfowler.com/eaaDev/EventSourcing.html`.

[16] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):44, 2014. doi: 10.1145/2523813.

[17] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106:1469–1495, 2017. doi: 10.1007/s10994-017-5642-8.

[18] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *PMLR*, pages 1321–1330, 2017.

[19] H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009. doi: 10.1109/TKDE.2008.239.

[20] A. Heilmeier, M. Graf, M. Lienkamp, J. Betz, et al. A virtual strategy engineer for racing strategy decision support in circuit motorsports. *Applied Sciences*, 10(12): 4229, 2020. doi: 10.3390/app10124229.

[21] V. Hettmann. From data to podium: A machine learning model for predicting formula 1 pit stop timing. Master's thesis, Nova School of Business and Economics, 2024. URL `https://run.unl.pt/bitstream/10362/175111/1/FROM_DATA_TO_PODIUM_A_MACHINE_LEARNING_MODEL_FOR_PREDICTING_FORMULA_1_PIT_STOP_TIMING.pdf`.

[22] J. Kreps, N. Narkhede, and J. Rao. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB Workshop*, 2011.

[23] M. Kull, T. M. Silva Filho, and P. Flach. Beta calibration: A well-founded and easily implemented improvement on logistic calibration for binary classifiers. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *PMLR*, pages 623–631, 2017.

[24] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[25] S. M. Lundberg, G. G. Erion, and S.-I. Lee. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888*, 2018. URL `https://arxiv.org/abs/1802.03888`.

[26] A. Niculescu-Mizil and R. Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 625–632, 2005. doi: 10.1145/1102351.1102430.

[27] T. Oehrly. Fastf1: A python package for accessing and analyzing formula 1 results, schedules, timing data, and telemetry, 2025. URL `https://github.com/theOehrly/Fast-F1`. Python package.

[28] N. C. Oza and S. J. Russell. Online bagging and boosting. In *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics*, pages 229–236, 2001.

[29] D. Piccinotti. Open loop planning for formula 1 race strategy identification. Master's thesis, Politecnico di Milano, 2020. URL `https://www.politesi.polimi.it/handle/10589/175624`.

[30] J. C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In A. J. Smola, P. Bartlett, B. Scholkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 1999.

[31] M. P. Quiroga Garcia. Optimization of pit stop strategies in formula 1 racing: A data-driven approach. Master's thesis, Universidad de Chile, 2024. URL `https://repositorio.uchile.cl/handle/2250/199664`.

[32] D. R. Roberts, V. Bahn, S. Ciuti, M. S. Boyce, J. Elith, G. Guillera-Arroita, S. Hauenstein, J. J. Lahoz-Monfort, B. Schroder, W. Thuiller, D. I. Warton, B. A. Wintle, F. Hartig, and C. F. Dormann. Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. *Ecography*, 40(8):913–929, 2017. doi: 10.1111/ecog.02881.

[33] T. Saito and M. Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PLOS ONE*, 10 (3):e0118432, 2015. doi: 10.1371/journal.pone.0118432.

[34] R. Sasikumar, A. Leema, and N. Balakrishnan. Data-driven pit stop decision support for formula 1 using deep learning models. *Frontiers in Artificial Intelligence*, 8: 1673148, 2025. doi: 10.3389/frai.2025.1673148.

[35] P. Schäfer. Fastf1 documentation. Online documentation, 2026. URL `https://docs.fastf1.dev/`.

[36] C. Widanage, C. Li, A. Tyagi, K. Teja, B. Peng, S. Kamburugamuve, J. Koskey, D. Baum, P. Smith, and J. Qiu. Anomaly detection over streaming data: Indy500 case study. In *2019 IEEE International Conference on Cloud Engineering Workshops*, 2019. doi: 10.1109/IC2EW.2019.00044.

List of Figures

List of Tables

List of Symbols

Variable	Description	Unit
k	Current lap at which a decision or prediction is emitted	laps
H	Prediction horizon used to match positive calls with future pit events	laps
P	Current race position of a driver in the drop-zone computation	position
j, m	Index variables used when walking the position ladder behind a driver	–
$lapTime$	Lap time of the current driver-lap observation	s
$stintBestLap$	Best lap time observed in the current stint for the driver	s
$thresholdPct$	Dynamic tyre-drop threshold used to compare the current lap with the stint-best lap	ratio
$compoundBase$	Base tyre-degradation threshold associated with the tyre compound	ratio
$fuelAdjust$	Fuel-related adjustment applied to the tyre-drop threshold	ratio
$hotTrackAdjust$	Track-temperature adjustment applied to the tyre-drop threshold	ratio
$thresholdFloor$	Minimum allowed value for the dynamic tyre-drop threshold	ratio
$gapToCarAhead_j$	Gap from car j to the car immediately ahead in the race order	s

Variable	Description	Unit
$pitLoss$	Estimated time loss caused by a pit stop under the current track-status condition	s
S	Total pit desirability score produced by the Flink Strategy Engine	score
S_{pace}	Score contribution associated with stint-relative pace degradation	score
S_{track}	Score contribution associated with track status, such as SC or VSC conditions	score
$S_{traffic}$	Score contribution associated with the expected traffic or clean-air condition after the pit stop	score
$P_{strategy}$	Strategy penalty applied when the inferred next compound cannot cover the remaining laps	score
S_{action}	Action urgency score combining tyre-life urgency and rival reaction pressure	score
S_{timing}	Timing-pressure score associated with local decision-window signals	score
P_{eor}	End-of-race suppression term used to reduce late unnecessary calls	score
$B_{competitive}$	Competitive pressure bonus derived from gaps and recent rival pit activity	score
$P_{uncertainty}$	Penalty associated with missing or low-confidence race-context signals	score
g	Estimated emergence gap after applying pit loss in the drop-zone computation	s
$deficit$	Fractional deficit between remaining laps and the estimated maximum stint length of the next compound	ratio
$lapsRemaining$	Number of laps remaining in the race at the current observation	laps
$maxStint_{next}$	Estimated maximum stint length for the inferred next tyre compound	laps
$tyreRatio$	Tyre-life ratio used to compute action urgency	ratio

Variable	Description	Unit
$racePrograss$	Fraction of the scheduled race already completed	ratio
$signalConfidence$	Confidence score derived from the availability and quality of race-context signals	ratio
$\Delta_{gap}\%$	Normalized change in gap before and after a pit stop	%
$preGap$	Gap to the relevant rival before the pit cycle	s
$postGap$	Gap to the relevant rival after the pit cycle	s
$baselineLapTime$	Reference lap time used to normalize the post-pit gap change	s
TP	True positives: positive calls matched to the required target condition	count
FP	False positives: positive calls that do not satisfy the target condition	count
FN	False negatives: target events or rows not covered by a positive call	count
$F_{0.5}$	Precision-weighted F-score used to summarize selected operating points	–
AP	Average Precision, used as a row-level ranking diagnostic	–
κ	Cohen's Kappa, used as a race-level agreement diagnostic	–
G-Mean	Geometric mean of sensitivity and specificity, used as an imbalance diagnostic	–

Acknowledgements

1. English

My academic journey would've been impossible without the support of many people, first I would like to thank my supervisor, Prof. Emanuele Della Valle, for his invaluable guidance and support throughout this thesis, for giving me freedom and trusting me to work on a thesis topic that I am passionate about.

Thanks to my girlfriend Giulia, for her love and encouragement, for being my rock during the challenging moments of this journey, and for always believing in me; believe in yourself as much as you believe in me and you will achieve anything you set your mind to, because you are truly amazing. I am grateful for your presence in my life and for the joy you bring to every day, I love you.

Thanks to my mum for her unconditional love and support, for always being there for me, and for teaching me the value of life and perseverance; words cannot express my gratitude enough.

Thanks to my dad, even if I wasn't lucky enough to get to know you, I can feel your passion and stubbornness in my veins which has been fuel for my ambitions and dreams, I hope I made you proud, a lot more has to come.

Thanks to my uncle Pepe for his support through my journey, as a fellow Politecnico student you knew how hard it can be sometimes, and your help and advice especially through my first years were immense.

Thanks to my grandma Carolina and to my grandpa Gigi, for their love and continuous support, I wish I could show grandpa Gigi how much I have achieved and how much I am still going to achieve, I know you are looking at me from above and cheering for me, thank you.

Thanks to the Tedesi family, for being a second family to me, for their love and support, and for always making me feel welcome and at home, I am grateful to have you in my life.

Thanks to all my family, we are too many to mention but I love you all and I am grateful for having you in my life, I couldn't have been more lucky.

Thanks to all my friends, for the ones at home and the ones I made along my academic journey, for the fun times and the support and strength you all gave me directly or indirectly, I am glad to have you in my life and I hope we will keep sharing great moments together for many years to come.

Here at Section 3 of the acknowledgements I want to share a link, it is a video of Ayrton Senna's speech on life and achievements, it is a great source of motivation and inspiration for me, I hope it can be for someone else as well.

Thanks to everyone from the bottom of my heart, this thesis is also yours.

2. Italian

Il mio percorso accademico sarebbe stato impossibile senza il supporto di tante persone, per prima cosa vorrei ringraziare il mio relatore, il Prof. Emanuele Della Valle, per la sua preziosa guida e il suo supporto durante questa tesi, per avermi lasciato libertà e per essersi fidato di me nel lavorare su un argomento che mi appassiona.

Grazie alla mia ragazza Giulia, per il suo amore e il suo incoraggiamento, per essere stata la mia roccia nei momenti più difficili di questo percorso, e per aver sempre creduto in me; credi in te stessa tanto quanto credi in me e riuscirai a raggiungere qualsiasi cosa tu voglia, perché sei davvero una persona incredibile. Sono grato di averti nella mia vita e per la gioia che porti in ogni giorno, ti amo.

Grazie a mia mamma per il suo amore e il suo supporto incondizionati, per esserci sempre stata per me, e per avermi insegnato il valore della vita e della perseveranza; le parole non bastano per esprimere tutta la mia gratitudine.

Grazie a mio papà, anche se non ho avuto la fortuna di conoscerti, sento la tua passione e la tua testardaggine scorrere nelle mie vene e alimentare le mie ambizioni e i miei sogni, spero di averti reso orgoglioso, ma questo è solo l'inizio.

Grazie a mio zio Pepe per il suo supporto durante il mio percorso, da ex studente del Politecnico sapevi quanto potesse essere difficile a volte, e il tuo aiuto e i tuoi consigli soprattutto durante i miei primi anni sono stati immensi.

Grazie a mia nonna Carolina e a mio nonno Gigi, per il loro amore e il loro supporto costante, vorrei poter mostrare a nonno Gigi quanta strada ho fatto e quanta ancora ne voglio fare, so che mi stai guardando da lassù e che fai il tifo per me, grazie.

Grazie alla famiglia Tedesi, per essere stata una seconda famiglia per me, per il loro amore e supporto, e per avermi sempre fatto sentire accolto e a casa, sono grato di avervi nella mia vita.

Grazie a tutta la mia famiglia, siamo troppi per nominarvi tutti ma vi voglio bene e sono grato di avervi nella mia vita, non avrei potuto essere più fortunato.

Grazie a tutti i miei amici, sia quelli di casa sia quelli che ho incontrato durante il mio percorso accademico, per i momenti belli e divertenti e per il supporto e la forza che mi avete dato direttamente o indirettamente, sono felice di avervi nella mia vita e spero che continueremo a condividere bei momenti insieme per molti anni ancora.

Qui nella Sezione 3 dei ringraziamenti voglio condividere un link, è un video del discorso di Ayrton Senna sulla vita e sui traguardi, per me è una grande fonte di motivazione e ispirazione e spero possa esserlo anche per qualcun altro.

Grazie a tutti dal profondo del cuore, questa tesi è anche vostra.

3. Link

Ayrton Senna's speech on life and achievements