Rule-Based Error Detection and Correction to Operationalize Movement Trajectory Classification

Bowen Xi [1], Kevin Scaria[1], Divyagna Bavikadi[2] and Paulo Shakarian[2]

[1]Arizona State University, Tempe, Arizona
[2]Syracuse University, Syracuse, New York

Abstract

Classification of movement trajectories has many applications in transportation and is a key component for large-scale movement trajectory generation and anomaly detection which has key safety applications in environments with unseen movement types. However, the current state-of-the-art (SOTA) are based on supervised deep learning - which leads to challenges when they encounter novel unseen classes. We provide a neuro-symbolic rule-based framework to conduct error correction and detection of these models to integrate into our movement trajectory platform. We provide a suite of experiments on several recent SOTA models where we show highly accurate error detection, the ability to improve accuracy on test data that includes novel movement types not seen in training set, and accuracy improvement for the base use case in addition to a suite of theoretical properties that informed algorithm development. Specifically, we show an F1 scores for predicting errors of up to 0.984, significant performance increase for unseen movement accuracy (8.51% improvement over SOTA for zero-shot accuracy), and accuracy improvement over the SOTA model.

1. Introduction

The identification of a mode of travel for a time-stamped sequence of global position system (GPS) known as "movement trajectories" has important applications in travel demand analysis [1], transport planning [2], and analysis of sea vessel movement [3]. More recently this problem has been of interest for security applications such as leading to efforts such as the IARPA HAYSTAC program[1] for which we have created and deployed a platform for trajectory analysis. A key facet of this problem is the proper classification of trajectories by movement type - particularly in the aftermath of an external shock like a natural disaster. However, the current state-of-the-art has relied on supervised neural models [4, 5] which have been shown to perform well but can experience failure when exposed to previously unseen data, specifically previously unidentified movement types. In this paper, we extend the current supervised neural methods with a lightweight error detection and correction rule (EDCR) framework providing an overall neurosymbolic system. This framework further enables critical technologies, like Artificial Intelligence for Transport, where it's typical to encounter unseen data and require models to not misidentify it.

The key intuition is that training and operation data can be used to learn rules that predict and correct errors in the supervised model. Once trained, the rules are employed operationally in two phases: first detection rules identify potentially misclassified movement trajectories. A second type of rule to re-classify the trajectories ("correction rules") is then used to re-assign the sample to a new class. We present a strong theoretical framework for EDCR rooted in both logic and rule mining. We formally prove how quantities related to learned rules (e.g., confidence and support) are related to changes in class-level machine learning metrics such as precision and recall. To demonstrate effectiveness empirically, we provide a suite of experiments that show this framework is highly effective in detecting errors (F1 of detecting errors of 0.875 for the SOTA model, and as high as 0.984 based on the examined models), unseen movement accuracy of 8.51% over SOTA for zero-shot tuning, and standard classification accuracy improvement over the SOTA.

In what follows, we provide further background on our domain problem and our current trajectory analysis platform (some of which is a review of [6]), introduce the algorithmic framework for EDCR including it's theoretical properties, and provide our suite of experimental results before concluding with our findings and future work.

2. Background

Overall concept and deployed system. Movement types not typically included in the ground truth data emerge with certain target environments (e.g, paid scooters in certain urban areas, auto-rickshaws in South Asia, or boats in Venice). As a result, IARPA (Intelligence Advanced Research Projects Activity) has identified problems relating to the characterization and generation of normal movement as a key problem of study in the HAYSTAC program. Here, the goal is to establish models of normal human movement at a fine-grain level and operationalize those models and techniques in a system deployed to a government environment for evaluation. As a performer on the program, [6] examine the problem of generating realistic movement trajectories.

Initial government tests for trajectory generation involved movement trajectories consisting of only a single mode of transportation. However, in preparation for the transition to operational use, the government has set requirements to analyze trajectories from various movement types - where the mode of transportation is not known. As such, we look to operationalize a movement trajectory classification module, which we have depicted in the context of our deployed cloud-based architecture shown in Figure 1.

This pipeline interfaces with the government system to access the raw geospatial data with related knowledge for various geolocations as well as historical agent trajectories and their corresponding objective files. Our initial ingest and containerized processes are held in a directed acyclic graph (DAG) as nodes. Our ingest mechanism first parses for the historical trajectories associated with a given agent to stage them in the S3 bucket. Then, geospatial data stored in Neo4j is consolidated into a knowledge graph and staged into the S3 bucket. We instantiate pods on the Amazon

STRL'25: Fourth International Workshop on Spatio-Temporal Reasoning and Learning, 16 August 2025, Montreal, Canada
bowenxi@asu.edu (B. X.); kscaria@asu.edu (K. Scaria); dbavikad@syr.edu (. D. Bavikadi); pshakar@syr.edu (P. Shakarian)
https://divyagnab.github.io/home/ (. D. Bavikadi)

[1]https://www.iarpa.gov/research-programs/haystac

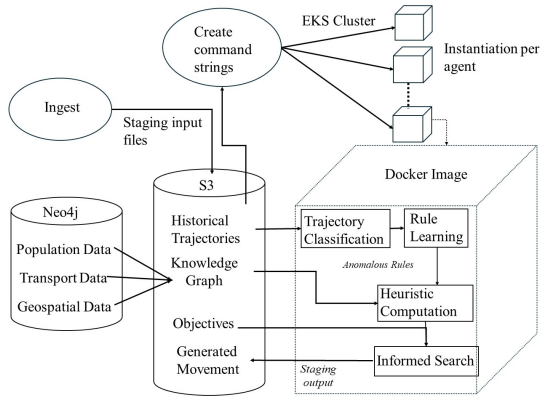

Figure 1: Overall system deployed for government testing

Elastic Kubernetes Service (EKS) cluster for all agents with a Docker image to analyze the staging folders and create the respective string commands specific to each agent. The trajectory classification module identifies and tags the modes of transportation in the corresponding trajectory, which is further used to learn rules while considering different types of movements. These rules along with the knowledge graph are used to compute the heuristic value for an informed search method (A* search) to generate movement trajectories [6]. As the container runs, generated movement instruction files are pushed to the appropriate output directory as seen in Figure 1. Additionally, the generated movement abides by predefined spatiotemporal constraints (objectives).

Movement Trajectory Classification Problem. The problem of classifying movement trajectories has been studied in the literature [7, 8, 9, 5, 4] and we shall refer to it as the movement trajectory classification problem (MTCP). We also note that this line of work differs from and is complementary to trajectory generation [6, 10, 11, 12] which does not seek to identify the mode of transportation. An MTCP instance is defined as given a sequence of GPS points, ω, assign one of n movement class from $\mathcal{C}$ which is often defined [4, 5] as $\mathcal{C} = \{\mathsf{walk}, \mathsf{bike}, \mathsf{bus}, \mathsf{drive}, \mathsf{train}\}$.

The current paradigm for the MTCP problem is to create a neural model f_θ that maps sequences to movement classes using a set of weights, θ. In this approach traditional methods (i.e., gradient descent) are used to find a set of parameters such that a loss function is minimized based on some training set $\mathcal{T}$ (where each sample $\omega \in \mathcal{T}$ is associated with a ground truth class $gt(\omega)$). Formally: $\arg\min_\theta \mathbb{E}_{\omega \in \mathcal{T}} Loss(f_\theta(\omega), gt(\omega))$. Within this paradigm, several approaches have been proposed. Most notably a CNN-based architecture [5] and the current state-of-the-art approach known as Long-term Recurrent Convolutional Network (LRCN) [4] which combines lower CNN layers and upper LSTM layers - both of which we use as baselines in addition to an extension of LRCN that uses an additional attention head (LRCNa).

Limitations of Current SOTA. However, there are several limitations to these approaches that are problematic in the context of the IARPA HAYSTAC use case.

- *Not designed for unseen movements.* Any supervised MTCP model requires a data set whose movement classes match the target environment. To address the more dynamic needs of our government customer, we require approaches that can identify when they are likely to give incorrect results to adapt to novel environments.
- *All classes known a-priori.* In the prior work, set $\mathcal{C}$ is treated as static and complete meaning that novel movement types not in training set will not be properly classified and not identified as being different than a movement type in class $\mathcal{C}$.
- *Previous Results Evaluated on Overlapping Training and Testing Sets.* As noted in [13], the standard evaluation of MTCP approaches has been on datasets that experience leakage between train and test. Due to the operational nature of this work, we must examine other splits.

Deploying movement trajectory classification models to a certain environments can lead to movements not seen in training (e.g., paid scooters are not seen in training but prevalent in certain urban areas). Hence, these "novel movements" will inherently be classified incorrectly. The common element in all of these limitations is an understanding of when such classifiers are likely wrong. However, this goes beyond retraining or selecting from different training data as the government customer envisions use-cases with unseen movement types- hence training data would be limited. This generally precludes meta-learning and domain generalization [14, 15, 16, 17] which attempt to account for changes in the distribution of data and/or selection of a model that was trained on data similar to the current problem. This work also differs from approached like One-Class Support Vector Machine [18] because of the inherent rule-based method in EDCR that can be leveraged for explainability and can further be built upon machine learning models.

Additionally, these problems must also be addressed in the context of our existing system (Figure 1), which employs symbolic reasoning to generate movement trajectories - ensuring they attain a degree of normalcy [6]. As a result, we examined approaches for characterizing failures in machine learning models such as introspection [19, 20], however, these approaches only predict model failure and do not attempt to explain or correct it. Another area of related work is machine learning verification that [21, 22, 23]) that looks to ensure the output of an ML model meets a logical specification - however to-date this work has not been applied to correct the output of a machine learning model and generally depend on the logical specifications being known a-priori (not an assumption we could make for our use case). In recent studies on abductive learning [24, 25] and neural symbolic reasoning [26], incorporate error correction mechanisms rooted in inconsistency with domain knowledge as logical rules - but as with verification, we do not have this symbolic knowledge a-priori.

3. Error Detection and Correction Rules

To address the issues of the previous section, we are employing a rule-based approach to correcting MTCP model f_θ. The intuition is that using limited data, we will learn a set of rules (denoted Π) that will be able to detect and correct errors of f_θ by logical reasoning [27]. Then, upon deployment for some new sequence ω, we would first compute the class $f_\theta(\omega)$ and then use the rules in set Π to conclude if the result of f_θ should be accepted and if not, provide an alternate class in an attempt to correct the mistake. In this

section, we formalize the error correcting framework with a simple first order logic (FOL) and provide analytical results relating aspects of learned rules that inform our analytical approach to learning such error detecting and correcting rules. We complete the section with a discussion on how various potential "failure conditions" are extracted to create the rules to correct errors.

In this paper, we shall assume a set $\mathcal{O}$ of operational sequences for which there is ground truth available after model training. This set can be the set of training data, a subset, or a superset. We notate the set of training data with $\mathcal{T}$. Later, in our experiments, we look at cases where $\mathcal{O} = \mathcal{T}$ and $\mathcal{T} \subseteq \mathcal{O}$ - however these are not requirements as our results are based on model performance on $\mathcal{O}$ - and we envision use-cases where $\mathcal{O}$ is significantly different from $\mathcal{T}$. On these samples, for each class i, the model (f_θ) returns class i for N_i of the samples, and for each class i we have the number of true positives TP_i, false positives FP_i, true negatives TN_i, and false negatives FN_i. We have precision $P_i = TP_i/N_i = TP_i/(TP_i + FP_i)$, recall $R_i = TP_i/(TP_i + FN_i)$, and prior of predicting class i: $\mathcal{P}_i = N_i/N$.

Language. We use a simple first-order language where samples are represented by constant symbols (ω). We define set C of m "condition" unary predicates $cond_1, \ldots, cond_m$ associated with each sample that can be either true or false - these are conditions that can be thought of as potentially leading to failure (but our learning algorithms will identify which ones lead to failure for a given prediction). These predicates can also be features related to a sample in the dataset. We also define unary predicates for each class i: $pred_i$, $corr_i$, and $error$ defined below.

- $pred_i$: True if and only if the model predicts class i i.e., $pred_i(\omega)$ is true iff $f_\theta(\omega) = i$.
- $corr_i$: This predicate is true if and only if the correct movement class for ω is i, i.e., $corr_i(\omega)$ is true iff $gt(\omega) = i$.
- $error$: This predicate is true if and only if an EDCR rule concludes there is an error in the model's prediction.

Rules. The set of rules Π will consist of two rules for each class: one "error detecting" and one "error correcting." Error detecting rules which will determine if a prediction by f_θ is not valid. In essence, we can think of such a rule as changing the movement class assigned by f_θ to some sample ω from i to "unknown." For a given class i, we will have an associated set of detection conditions DC_i that is a subset of conditions, the disjunction of which is used to determine if f_θ gave an incorrect classification.

$$error(\omega) \leftarrow pred_i(\omega) \wedge \bigvee_{j \in DC_i} cond_j(\omega) \tag{1}$$

After the application of the error detection rules for each class, we may consider re-assigning the samples to another class using a second type of rule called the "corrective rule." Such rules are formed based on a subset of conditions-class pairs $CC_i \subseteq C \times \mathcal{C}$. The disjunction of such condition-class pairs are used to correct the class of a given sample.

$$corr_i(\omega) \leftarrow \bigvee_{q,r \in CC_i} (cond_q(\omega) \wedge pred_r(\omega)) \tag{2}$$

Associated with the rules of both types are the following values - both are defined as zero if there are no conditions.

Support (s): fraction of samples in $\mathcal{O}$ where the body is true.

Support w.r.t. class i (s_i): given the subset of samples where the model predicts class i, the fraction of those samples where the body is true (note the denominator is N_i).

Confidence (c): the number of times the body and head are true together divided by the number of times the body is true.

Now we present some analytical results that inform our learning algorithms. Our strategy for learning involves first learning detection rules (which establish conditions for which a given classification decision by f_θ is deemed incorrect) and then learning correction rules (which then correct the detected errors by assigning a new movement class to the sample). We formalize these two tasks as follows.

Improvement by error detecting rule. For a given class i, find a set of conditions DC_i such that precision is maximized and recall decreases by, at most ϵ.

Improvement by error correcting rule. For a given class i, find a subset CC_i of $C \times \mathcal{C}$ such that both precision and recall are maximized.

Properties of Detection Rules. First, we examine the effect on precision and recall when an error detecting rule is used. Our first result shows a bound on precision improvement. If class support (s_i) is less than $1 - P_i$, which we would expect (as the rule would be designed to detect the $1 - P_i$ portion of results that failed), then we can also show that the quantity $c \cdot s_i$ gives us an upper bound on the improvement in precision.[2]

Theorem 1. *Consider an error detecting rule with support s_i and confidence c, initial precision P_i of model f_θ for class i, then under the condition $s_i \leq 1 - P_i$, the precision of model f_θ for class i, after applying the error detecting rule increases by a function of both s_i and c. The increase is no greater than $c \cdot s_i$ and this quantity is a normalized polymatroid submodular function with respect to the set of conditions in the rule DC_i.*

The error detecting rules can cause the recall to stay the same or decrease. Our next result tells us precisely how much recall will decrease.

Theorem 2. *After applying the rule to detect errors, the recall will decrease by $(1 - c)s_i \frac{R_i}{P_i}$ and this quantity is a normalized polymatroid submodular function with respect to the set of conditions in the rule DC_i.*

Algorithm 1 DetRuleLearn

Require: Class i, Recall reduction threshold ϵ, Condition set C
Ensure: Subset of conditions DC_i
$DC_i := \emptyset$
$DC^* := \{c \in C \text{ s.t. } NEG_{\{c\}} \leq \epsilon \cdot \frac{N_i P_i}{R_i}\}$
while $DC^* \neq \emptyset$ **do**
  $c_{best} = \arg\max_{c \in DC^*} POS_{DC_i \cup \{c\}}$
  Add c_{best} to DC_i
  $DC^* := \{c \in C \setminus DC_i \text{ s.t. } NEG_{DC_i \cup \{c\}} \leq \epsilon \cdot \frac{N_i P_i}{R_i}\}$
end while
return DC_i

As the quantities identified Theorems 1 and 2 are submodular and monotonic, we can see that the selection of a set of rules to maximize $c \cdot s_i$ subject to the constraint that $(1 - c)s_i \frac{R_i}{P_i} \leq \epsilon$ is a special case of the "Submodular

[2]Complete proofs for all formal results can be found at https://arxiv.org/abs/2308.14250.

Cost Submodular Knapsack" (SCSK) problem and can be approximated with a simple greedy algorithm [28] with approximation guarantee of polynomial run time (Theorem 4.7 of [28]). Our algorithm DetRuleLearn is an instantiation of such an approach to creating an error detecting rule for a given class that maximize precision while not reducing recall more than ϵ. Here, ϵ is treated as a hyperparameter. Also, POS_{DC} and NEG_{DC} are simply the number of samples that satisfy the conditions for some set DC and are true errors (for POS_{DC}) and non-errors (for NEG_{DC}). In other words, given a set of condition class pairs and the rule of interest, BOD here is the number of examples that satisfy the body (class-condition pair) of the error detection rules, and POS here is the number of examples that satisfy the body (class-condition pair) and the head of the error detection rules. P_i, R_i are precision and recall for class i while N_i is the number of samples that the model classifies as class i.

Properties of Corrective Rules. In what follows, we shall examine the results for corrective rules. Here, the error correcting rule with predicate $corr_j$ in the head will have a disjunction of elements of set $CC_i \subseteq C \times \mathcal{C}$. Also, note that here the support s is used instead of class support (s_i). Here we find that both precision and recall increase with rule confidence (Theorem 3).

Theorem 3. *For the application of error correcting rules, both precision and recall increase if and only if rule confidence (c) increases.*

This result suggests that optimizing confidence will optimize both precision and recall. However, this is not a monotonic function over CC_i, so we adopt a fast, heuristic approach for non-monotonic optimization based on [29], presented by CorrRuleLearn in this paper. Here, we will consider an initial set of condition-class pairs CC_{all} that is a subset of $C \times \mathcal{C}$. For a given class for which we create an error correcting rule, we select CC_i from this larger set using our approach. Note here that POS_{CC} is the number of samples that satisfy the rule body and head ($corr_i(\omega)$ in this case) given a set of condition-class pairs CC while BOD_{CC} is the number of samples that satisfy the body formed with set CC.

Learning Detection and Correction Rules Together. Error correcting rules created using CorrRuleLearn will provide optimal improvement to precision and recall for the rule in the target class, but in the case of multi-class problems, it will cause recall to drop for some other classes. However, we can combine error detecting and correcting rules to overcome this difficulty. The intuition is first to create error detecting rules for each class, which effectively re-assigns any sample into an "unknown" class. Then, we create a set CC_{all} (used as input for CorrRuleLearn) based on the conditions selected by the error detecting rules. In this way, we will not decrease recall beyond what occurs in the application of error detecting rules.

Algorithmic Efficiency. We note that these algorithms are quite efficient. For example, DetRuleLearn is quadratic in the number of conditions and linear in the number of samples. However, in practice it actually performs better, as the outer loop iterates significantly less than the total number of conditions and the number of selected conditions is reduced with each iteration. Likewise, the algorithm CorRuleLearn is linear in the number of samples and linear in the number of condition-class pairs.

Algorithm 2 CorrRuleLearn

Require: Class i, Set of condition-class pairs CC_{all}
Ensure: Subset of condition-class pairs CC_i
$CC_i := \emptyset$
$CC_i' := CC_{all}$
Sort each $(c, j) \in CC_{all}$ from greatest to least by $\frac{POS_{\{(c,j)\}}}{BOD_{\{(c,j)\}}}$
and remove $\frac{POS_{\{(c,j)\}}}{BOD_{\{(c,j)\}}} \leq P_i$
for $(c, j) \in CC_{all}$ selected in order of the sorted list **do**
  $a := \frac{POS_{CC_i \cup \{(c,j)\}}}{BOD_{CC_i \cup \{(c,j)\}}} - \frac{POS_{CC_i}}{BOD_{CC_i}}$
  $b := \frac{POS_{CC_i' \setminus \{(c,j)\}}}{BOD_{CC_i' \setminus \{(c,j)\}}} - \frac{POS_{CC_i'}}{BOD_{CC_i'}}$
  if $a \geq b$ **then**
    $CC_i := CC_i \cup \{(c, j)\}$
  else
    $CC_i' := CC_i' \setminus \{(c, j)\}$
  end if
end for

if $\frac{POS_{CC_i}}{BOD_{CC_i}} \leq P_i$ **then**
  $CC_i := \emptyset$
end if
return CC_i

Algorithm 3 DetCorrRuleLearn

Require: Recall reduction threshold ϵ, Condition set C
Ensure: Set of rules Π
$\Pi := \emptyset$
$CC_{all} := \emptyset$
for Each class i **do**
  $DC_i :=$ DetRuleLearn(i, ϵ, C)
  if $DC_i \neq \emptyset$ **then**
    $\Pi := \Pi \cup$
    $\{error(\omega) \leftarrow pred_i(\omega) \wedge \bigvee_{j \in DC_i} cond_j(\omega)\}$
  end if
  for $cond \in DC_i$ **do**
    $CC_{all} := CC_{all} \cup \{(cond, i)\}$
  end for
end for
for Each class i **do**
  $CC_i :=$ CorrRuleLearn(i, CC_{all})
  if $CC_i \neq \emptyset$ **then**
    $\Pi := \Pi \cup$
    $\{corr_i(\omega) \leftarrow \bigvee_{q,r \in CC_i} (cond_q(\omega) \wedge pred_r(\omega))\}$
  end if
end for
return Π

Conditions for Error Detection and Correction. Practically, the source of the conditions from which our algorithms create EDCR rules (set C) needs to be instantiated. We adopt two straightforward approaches to this. First, we use a binary version of the classifier – for given class i, we have a binary classifier g_i which returns "true" for sample ω if g_i assigns it as i and "false" otherwise. In this way, for each sample ω we have a $g_i(\omega)$ condition for each of the classes. The second way we create conditions is based on outlier behavior based on the velocity of the vehicle in the sample. Here, if the velocity of a given sample is above a threshold (based on the maximum value for ground truth in the training data) this velocity condition is true - and it is false otherwise.

Evaluated Model	Error Precision (EDCR)	Error Recall (EDCR)	Error F1 (EDCR)
LRANa	0.999	0.941	0.969
LRCN	0.996	0.780	0.875
CNN	0.987	0.982	0.984

Table 1
EDCR Error Detection Results - this table shows EDCR's ability to detect error for three different models.

4. Experimental Evaluation

Experimental Setup. Previous work such as [4] is known to have data leakage based on the split between training and test primarily due to segments of a movement sequence existing in both training and test sets [13]. In this paper, we examine a training-test split with no overlap between the two avoiding this error and more closely resembling our target use-case. The assessments in this paper used GPS trajectories obtained from the GeoLife project [7] which include ground truth (note that ground truth data for our target application was unavailable at the time of this writing). All experiments were conducted on an NVIDIA A100 GPU using Python 3.10, with an 80/20 train–test split. Source code is available via https://github.com/lab-v2/Error-Detection-and-Correction.

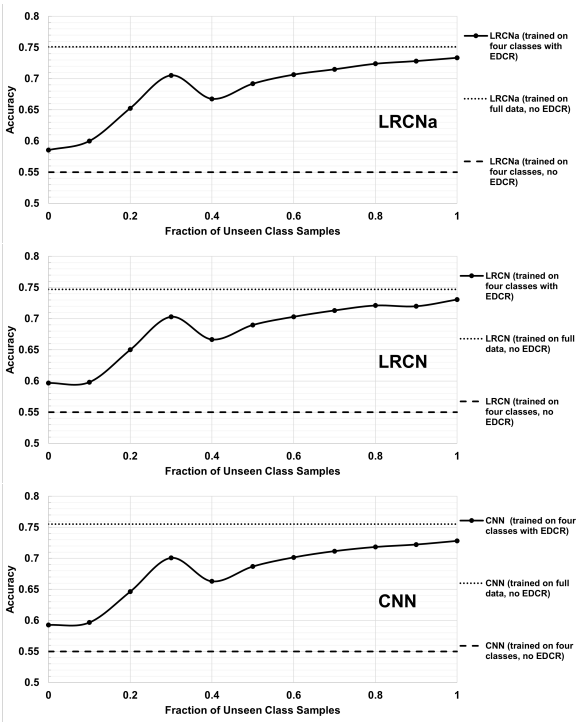

Figure 2: Results for experiments with two movement classes removed from training for the LRCNa, LRCN, and CNN models.

Error Detection Experiments. First we examined the ability of learned error detection rules to detect errors in the underlying model. Here we examined three base model architectures CNN [5], LRCN [4], and our version of LRCN with an additional attention head (LRCNa). In this experiment, error detection rules were trained from the same training data as the model. Similar to previous work on examining the ability to detect errors in a machine learning model [19] we evaluated precision, recall, and F1 of the ability of rules to identify errors. These can be thought of as the fraction of results where our learned error detection rules correctly return an error (error precision), the fraction of errors identified (error recall), and the harmonic mean of the two (error F1). The results shown in Table 1 demonstrate consistently high precision and recall for detecting errors across all model types - specifically obtaining a 0.875 F1 for errors in the SOTA model (LRCN) and a top F1 of 0.984 (for CNN).

Test Data with Additional Classes. A key set of concerns for our use-case was the ability to deploy movement trajectory classification in an environment where the data differs from the training data - specifically containing previously unidentified classes. To examine this, we trained CNN, LRCN, and LRCNa models without incorporating the walk and drive classes (Figure 2). We note here both detection and correction are used. We initially learned the EDCR rules with the same training data in the model - which results in no sample being corrected to a class unseen in training data and effectively is zero-shot tuning of the base model by EDCR. However, due to detection, this still resulted in accuracy improvements of 6.41%, 8.51%, and 7.76% for LRCNa, LRCN, and CNN respectively. We then added few-shot samples from the unseen data (the x-axis of Figure 2) giving us few-shot tuning of the base model. Here with only 20% of the samples with the unseen classes, we obtained an overall accuracy of 0.65 on all three models representing a $17-18\%$ improvement. We note these results are obtained without direct access to the underlying model, which may indicate that EDCR has the potential for adaptation of arbitrary f_θ models to novel scenarios - a key use case for our government customer.

Precision-Recall Trade-off. A key intuition in our algorithmic design with the ability for the hyperparameter to ϵ to trade-off precision and recall. Hence, we examined the effect in varying ϵ on test data that resembled training data (results for LRCN are shown in Figure 3). Recall that ϵ is interpreted as the maximum decrease in recall. We observed and validated the theoretical reduction (TR) in recall empirically and the experiments show us that in all cases, recall was no lower than the threshold specified by the hyperparameter ϵ though recall decreases as ϵ increases. In many cases, the experimental evaluation reduced recall significantly less than expected. We also see a clear relationship between ϵ, precision, and recall: increasing ϵ leads to increased precision and decreased recall - which also aligns with our analytical results. We also note that while DetCorrRuleLearn calls for a single ϵ hyperparameter, it is possible to set it differently for each class (e.g., lower values for classes where recall is important, higher values for classes where false positives are expensive). This may be beneficial as F1 for different classes seemed to peak for different values of ϵ. We leave the study of heterogeneous ϵ settings to future work.

Evaluated Model	No EDCR (baseline)	With EDCR (ours)
LRCNa	0.751	**0.763** (+1.6%)
LRCN	0.747	**0.760** (+1.7%)
CNN	0.755	0.755 ($\pm$ 0%)

Table 2
Overall accuracy when all classes are represented with and without EDCR.

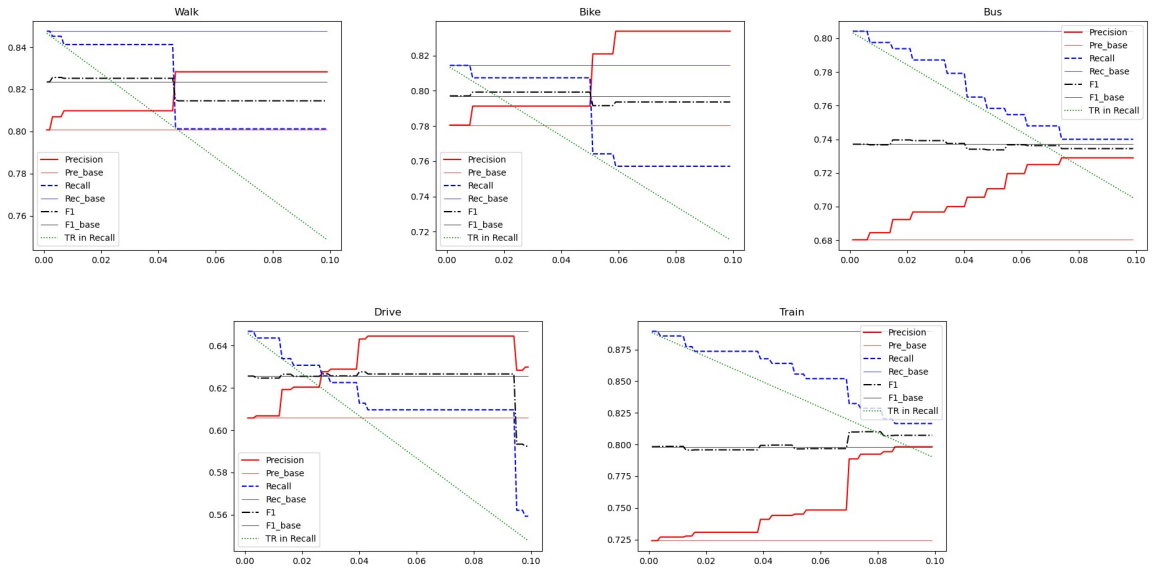

Figure 3: LRCN Results for application of error detection and correction rules as a function of ϵ. TR in Recall is the theoretical reduction in recall based on analytic results.

Accuracy Improvement via EDCR. We also investigated EDCR's ability to provide overall accuracy improvement to the base model. Here we trained each of the three models (LRCNa, LRCN, CNN) and associated EDCR rules (on the same training data as the model) and evaluated the overall accuracy on the test set both with and without applying rules (see Table 3). We found that that EDCR provided a noticeable improvement in both LRCN and LRCNa models - effectively establishing a new SOTA when evaluated with no overlap between training and testing. We also examined other splits between training and testing (not depicted) and obtained comparable results.

5. Conclusion

We propose a rule-based framework for the error detection and correction of supervised neural models for classification of movement trajectories. Our framework uses the training data to learn rules to be employed in the testing phase. Firstly, we use the detection rules to identify the movement trajectories that are misclassified by the supervised model and then we use the correction rules to re-classify the movement. Further, we formally prove the relation of confidence and support of the learned rules to the changes in the classification metrics like precision and recall. To show EDCR's emperical validation, we first report the framework's ability to identify errors with the F1 scores going up to 0.984. We also show overall accuracy imporvement over the SOTA model by employing the EDCR framework. Our framework is specifically useful in cases of encountering novel classes not seen in training data as shown by a 8.51% improvement of unseen movement accuracy over SOTA for zero-shot tuning. Additionally, we discuss operationalizing our trajectory classification method in our deployed system. There are several directions for future work. First, we look to explore other methods to create the conditions, in particular leveraging ideas from conformal prediction [30]. Another direction is to look at alternative solutions to learn the rules allowing for more complicated rule structures. Human validation of the rules responsible for a corrected label can be conducted for further evaluation. Finally, the use of rules for error detection and correction of machine learning models presented here may be useful in domains such as vision. To reliably incorporate vision models in real-world applications for tasks like object detection, image classification, and motion tracking, etc., EDCR framework can be leveraged to improve the overall system's accuracy and robustness by identifying and correcting it's misclassification.

Ethical Statement

There are no ethical issues.

6. Acknowledgments

This research is supported by the Intelligence Advanced Research Projects Activity (IARPA) via the Department of Interior/ Interior Business Center (DOI/IBC) contract number 140D0423C0032. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DOI/IBC, or the U.S. Government. Additionally, some of the authors are supported by ONR grant N00014-23-1-2580 and ARO grant W911NF-24-1-0007.

References

[1] H. Huang, Y. Cheng, R. Weibel, Transport mode detection based on mobile phone network data: A systematic review, Transportation Research Part C: Emerging Technologies 101 (2019) 297–312.

[2] M. Lin, W.-J. Hsu, Mining gps data for mobility patterns: A survey, Pervasive and mobile computing 12 (2014) 1–16.

[3] G. Fikioris, K. Patroumpas, A. Artikis, M. Pitsikalis, G. Paliouras, Optimizing vessel trajectory compression for maritime situational awareness, GeoInformatica 27 (2023) 565–591.

[4] J. Kim, J. H. Kim, G. Lee, Gps data-based mobility mode inference model using long-term recurrent convolutional networks, Transportation Research Part C: Emerging Technologies 135 (2022) 103523.

[5] S. Dabiri, K. Heaslip, Inferring transportation modes from gps trajectories using a convolutional neural network, Transportation research part C: emerging technologies 86 (2018) 360–371.

[6] D. Bavikadi, D. Aditya, D. Parkar, P. Shakarian, G. Mueller, C. Parvis, G. I. Simari, Geospatial trajectory generation via efficient abduction: Deployment for independent testing, in: Proceedings of the 40th International Conference on Logic Programming (ICLP 2024), 2024.

[7] Y. Zheng, Q. Li, Y. Chen, X. Xie, W.-Y. Ma, Understanding mobility based on gps data, in: Proceedings of the 10th international conference on Ubiquitous computing, 2008, pp. 312–321.

[8] H. Wang, G. Liu, J. Duan, L. Zhang, Detecting transportation modes using deep neural network, IEICE TRANSACTIONS on Information and Systems 100 (2017) 1132–1135.

[9] M. Simoncini, L. Taccari, F. Sambo, L. Bravi, S. Salti, A. Lori, Vehicle classification from low-frequency gps data with recurrent neural networks, Transportation Research Part C: Emerging Technologies 91 (2018) 176–191.

[10] M. Janner, Q. Li, S. Levine, Offline reinforcement learning as one big sequence modeling problem, in: Advances in Neural Information Processing Systems, 2021.

[11] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, I. Mordatch, Decision transformer: Reinforcement learning via sequence modeling, CoRR abs/2106.01345 (2021). URL: https://arxiv.org/abs/2106.01345. arXiv:2106.01345.

[12] M. Itkina, M. J. Kochenderfer, Interpretable self-aware neural networks for robust trajectory prediction, 2022. arXiv:2211.08701.

[13] J. Zeng, Y. Yu, Y. Chen, D. Yang, L. Zhang, D. Wang, Trajectory-as-a-sequence: A novel travel mode identification framework 146 (2023) 103957. URL: https://www.sciencedirect.com/science/article/pii/S0968090X22003709. doi:https://doi.org/10.1016/j.trc.2022.103957.

[14] T. Hospedales, A. Antoniou, P. Micaelli, A. Storkey, Meta-learning in neural networks: A survey, IEEE transactions on pattern analysis and machine intelligence 44 (2021) 5149–5169.

[15] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, C. C. Loy, Domain generalization: A survey, IEEE Transactions on Pattern Analysis and Machine Intelligence (2022).

[16] J. Vanschoren, Meta-learning: A survey, arXiv preprint arXiv:1810.03548 (2018).

[17] P. Maes, D. Nardi, Meta-level architectures and reflection (1988).

[18] H. J. Shin, D.-H. Eom, S.-S. Kim, One-class support vector machines,Äîan application in machine fault detection and classification, Computers Industrial Engineering 48 (2005) 395–408. URL: https://www.sciencedirect.com/science/article/pii/S0360835205000100. doi:https://doi.org/10.1016/j.cie.2005.01.009.

[19] S. Daftry, S. Zeng, J. A. Bagnell, M. Hebert, Introspective perception: Learning to predict failures in vision systems, 2016. URL: http://arxiv.org/abs/1607.08665. doi:10.48550/arXiv.1607.08665. arXiv:1607.08665 [cs].

[20] M. S. Ramanagopal, C. Anderson, R. Vasudevan, M. Johnson-Roberson, Failing to learn: Autonomously identifying perception failures for self-driving cars 3 (2018) 3860–3867. URL: http://arxiv.org/abs/1707.00051. doi:10.1109/LRA.2018.2857402. arXiv:1707.00051 [cs].

[21] R. Ivanov, T. Carpenter, J. Weimer, R. Alur, G. Pappas, I. Lee, Verisig 2.0: Verification of neural network controllers using taylor model preconditioning, in: Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I, Springer-Verlag, 2021, pp. 249–262. URL: https://doi.org/10.1007/978-3-030-81685-8_11. doi:10.1007/978-3-030-81685-8_11.

[22] K. Jothimurugan, S. Bansal, O. Bastani, R. Alur, Compositional reinforcement learning from logical specifications, in: Advances in Neural Information Processing Systems, 2021.

[23] M. Ma, J. Gao, L. Feng, J. Stankovic, Stlnet: Signal temporal logic enforced multivariate recurrent neural networks, Advances in Neural Information Processing Systems 33 (2020) 14604–14614.

[24] Y.-X. Huang, W.-Z. Dai, Y. Jiang, Z.-H. Zhou, Enabling knowledge refinement upon new concepts in abductive learning (2023).

[25] W.-Z. Dai, Q. Xu, Y. Yu, Z.-H. Zhou, Bridging machine learning and logical reasoning by abductive learning, NeurIPS 32 (2019).

[26] C. Cornelio, J. Stuehmer, S. X. Hu, T. Hospedales, Learning where and when to reason in neuro-symbolic inference, in: The Eleventh International Conference on Learning Representations, 2022.

[27] D. Aditya, K. Mukherji, S. Balasubramanian, A. Chaudhary, P. Shakarian, PyReason: Software for open world temporal logic, in: AAAI Spring Symposium, 2023.

[28] R. Iyer, J. Bilmes, Submodular optimization with submodular cover and submodular knapsack constraints, in: Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2, NIPS'13, Curran Associates Inc., Red Hook, NY, USA, 2013, p. 2436–2444.

[29] N. Buchbinder, M. Feldman, J. Naor, R. Schwartz, A tight linear time (1/2)-approximation for unconstrained submodular maximization, in: 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science, 2012, pp. 649–658. doi:10.1109/FOCS.2012.73.

[30] J. Sun, Y. Jiang, J. Qiu, P. Nobel, M. J. Kochenderfer, M. Schwager, Conformal prediction for uncertainty-aware planning with diffusion dynamics model, in: NeurIPS, volume 36, 2023, pp. 80324–80337. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/fe318a2b6c699808019a456b706cd845-Paper-Conference.pdf.

A. Appendix

A.1. Proof of Theorem 1

Under the condition $s_i \leq 1 - P_i$, the precision of model f_θ for class i, with initial precision P_i, after applying an error detecting rule with support s_i and confidence c increases by a function of s_i and c and is no greater than $c \cdot s_i$ and this quantity a normalized polymatroid submodular function with respect to the set of conditions in the rule DC_i.

Proof. CLAIM 1: The precision of model f_θ for class i, with initial precision P_i, after applying an error detecting rule with support s_i and confidence c increases by:

$$\frac{s_i}{1 - s_i}(c + P_i - 1) \tag{3}$$

The total number of items that f_θ will attempt to classify as i before error detection is $N_i = TP_i + FP_i$. Out of those, $s_i \cdot N_i$ will be detected by the rule. However, a fraction of $(1 - c)$ will be samples that would have been true positives if not detected. Hence, the new precision can be written as follows:

$$\frac{TP_i - (1 - c)s_i \cdot N_i}{N_i - s_i \cdot N_i} \tag{4}$$

As $P_i \cdot N_i = TP_i$, we have:

$$\frac{P_i \cdot N_i - (1 - c)s_i \cdot N_i}{N_i(1 - s_i)} \tag{5}$$

$$= \frac{P_i - (1 - c)s_i}{(1 - s_i)} \tag{6}$$

Now we subtract from that quantity the initial precision.

$$\frac{P_i - (1 - c)s_i}{(1 - s_i)} - P_i \tag{7}$$

$$= \frac{P_i - (1 - c)s_i}{(1 - s_i)} - \frac{(1 - s_i)P_i}{1 - s_i} \tag{8}$$

$$= \frac{-s_i + s_i c + P_i s_i}{1 - s_i} \tag{9}$$

$$= \frac{s_i}{1 - s_i}(c + P_i - 1) \tag{10}$$

CLAIM 2: If $s_i \leq 1 - P_i$ then $c \cdot s_i$ is a upper bound on the improvement in precision.

BWOC, then by Claim 1 we have.

$$\frac{s_i}{1 - s_i}(c + P_i - 1) > c \cdot s_i \tag{11}$$

$$c + P_i - 1 > c(1 - s_i) \tag{12}$$

$$c + P_i - 1 > c - c \cdot s_i \tag{13}$$

$$c \cdot s_i > 1 - P_i \tag{14}$$

$$c \cdot s_i > s_i \tag{15}$$

However, as $c \leq 1$ this is a contradiction.

CLAIM 3: $c \cdot s_i = POS/N_i$ where POS is the number of samples where both the rule body and head are satisfied.

Let BOD be the number of samples that the body of the rule is true. This gives us $c \cdot s_i = \frac{POS}{BOD}\frac{BOD}{N_i}$ which is equivalent to the statement of the claim.

CLAIM 4: The quantity $c \cdot s_i$ is submodular w.r.t. set DC.

We show this by the submodularity of POS as N_i is a constant as well as the result of Claim 3. BWOC, POS is not submodular for some set DC. We use the symbol $POS(DC)$ to denote this and assume the existence of two sets of conditions DC_1, DC_2. Then, the following must be true:

$$POS(DC_1) + POS(DC_2) < \tag{16}$$

$$POS(DC_1 \cup DC_2) + POS(DC_1 \cap DC_2) \tag{17}$$

We can re-write $POS(DC_1 \cup DC_1)$ as:

$$|\bigcup_{cond \in DC_1 \cup DC_2}\{x|cond(\omega) \wedge pred_x\}| \tag{18}$$

$$= |\bigcup_{cond \in DC_1}\{x|cond(\omega) \wedge pred_x\} \cup \tag{19}$$

$$\bigcup_{cond \in DC_2}\{x|cond(\omega) \wedge pred_x\}| \tag{20}$$

$$= POS(DC_1) + POS(DC_2) - \tag{21}$$

$$POS(DC_1 \cap DC_2) \tag{22}$$

Substituting this back into inequality 16, we can re-write the right-hand side as:

$$POS(DC_1) + POS(DC_2) - \tag{23}$$

$$POS(DC_1 \cap DC_2) + POS(DC_1 \cap DC_2) \tag{24}$$

$$= OS(DC_1) + POS(DC_2) \tag{25}$$

Which give us our contradiction.

CLAIM 5: $c \cdot s_i$ monotonically increases with DC.

By claim 1, as the quantity equals POS/N_i and N_i is a constant, we just need to show monotonicity of POS. Clearly POS increases monotonically as additional elements in DC can only make it increase.

CLAIM 6: When $DC = \emptyset$, $c \cdot s_i = 0$.

Follows directly from the fact that we define s_i as zero is no conditions are used.

Proof of theorem. Follows directly from claims 1-6. □

A.2. Proof of Theorem 2

After applying the rule to detect errors, the recall will decrease by $(1 - c)s_i\frac{R_i}{P_i}$ and this quantity is a normalized polymatroid submodular function with respect to the set of conditions in the rule DC_i.

Proof. CLAIM 1: After applying the rule to detect errors, the recall will decrease by $(1 - c)s_i\frac{R_i}{P_i}$.

The number of corrections made by the rule is $s_i(TP_i + FP_i)$ with $(1 - c)$ fraction of these being incorrect (so the false negatives increases by $s_i(TP_i + FP_i)1 - c)$). Note that the sum $TP_i + FN_i$ does not change after error detection, as any true positive "detected" as being incorrect becomes a false negative, and false negatives do not otherwise change from error detection. Therefore, the new recall is:

$$\frac{TP_i - s_i(1 - c)(TP_i + FP_i)}{TP_i + FN_i} \tag{26}$$

When this quantity is subtracted from the original recall (R_i), we obtain:

$$R_i - \frac{TP_i - s_i(1 - c)(TP_i + FP_i)}{TP_i + FN_i} \tag{27}$$

$$= \frac{TP_i - (TP_i - s_i(1 - c)(TP_i + FP_i))}{TP_i + FN_i} \tag{28}$$

$$= \frac{s_i(1 - c)(TP_i + FP_i))}{TP_i + FN_i} \tag{29}$$

$$= s_i(1 - c)\left(\frac{TP_i}{TP_i + FN_i} + \frac{FP_i}{TP_i + FN_i}\right) \tag{30}$$

$$= s_i(1 - c)\left(R_i + \frac{FP_i}{TP_i + FN_i}\right) \tag{31}$$

We note that $FP_i = \frac{TP_i}{P_i} - TP_i = \frac{TP_i - P \cdot TP_i}{P_i}$ which gives us:

$$s_i(1-c)\left(R_i + \frac{TP_i}{P(TP_i+FN_i)} - \frac{TP_i \cdot P_i}{P_i(TP_i+FN_i)}\right) \quad (32)$$

$$= s_i(1-c)\left(R_i + \frac{R_i}{P_i} - R_i\right) \quad (33)$$

$$= (1-c)s_i\frac{R_i}{P_i} \quad (34)$$

CLAIM 2: $(1-c)s_i\frac{R_i}{P_i}$ is a normalized polymatroid submodular function with respect to the set of conditions in the rule DC_i. Note that BOD is the number of samples that satisfy the body, while POS is the number of samples that satisfy the body and head, $NEG = POS - BOD$.

$$(1-c)s_i\frac{R_i}{P_i} = \left(1 - \tfrac{POS}{BOD}\right)\tfrac{BOD}{N_i}\tfrac{R_i}{P_i} \quad (35)$$

$$= \tfrac{NEG}{BOD}\tfrac{BOD}{N}\tfrac{R_i}{P_i} \quad (36)$$

$$= NEG\tfrac{1}{N_i}\tfrac{R_i}{P_i} \quad (37)$$

As $\frac{1}{N_i}\frac{R_i}{P_i}$ is a constant, we need to show the submodularity of NEG which follows the same argument for POS as per Claim 4 of Theorem 1. Likewise, NEG is monotonic (mirroring the argument of Claim 5 of Theorem 1) and normalized by the definition of s_i in the case where there are no conditions. The statement of the theorem follows. □

A.3. Proof of Theorem 3

For the application of error correcting rules, both precision and recall increase if and only if rule confidence (c) increases.

Proof. CLAIM 1: Precision increases by $\frac{cs-P_is}{\mathcal{P}_i+s}$.

The new precision is equal to the following:

$$\frac{TP_i + csN}{M_i + sN} \quad (38)$$

The improvement of the precision can be derived as follows.

$$\frac{TP_i + csN}{M_i + sN} - P_i = \quad (39)$$

$$= \tfrac{TP_i+csN-P_iM_i-P_isN}{M_i+sN} \quad (40)$$

$$= \tfrac{TP_i+csN-TP_i-P_isN}{M_i+sN} \quad (41)$$

$$= \tfrac{csN-P_isN}{M_i+sN} \quad (42)$$

$$= \tfrac{cs-P_is}{\mathcal{P}_i+s} \quad (43)$$

CLAIM 2: If count of samples satisfying both rule body and head (the numerator of confidence) increases, then precision increases.

Suppose BWOC the claim is not true. Then for some value of POS for which the improvement in precision is greater than $POS' = POS + 1$. Note that, in this case, the number of samples satisfying the body also increases by 1. First, we know that we can re-write the result of claim 1 as follows.

$$\frac{POS - P_iBOD}{M_i + BOD} \quad (44)$$

Therefore, using the result from Claim 1, the following relationship must hold.

$$\frac{POS - P_iBOD}{M_i + BOD} > \frac{POS + 1 - P_iBOD - P_i}{M_i + BOD + 1} \quad (45)$$

$$POS - P_iBOD > M_i(1-P_i) + BOD(1-P_i) \quad (46)$$

$$POS > M(1-P_i) + BOD \quad (47)$$

This gives us a contradiction, as $M(1-P_i) \geq 0$ and $POS \leq BOD$ by definition.

CLAIM 3: If the difference in precision increases, the number of samples satisfying both rule body and head must increase.

By definition, the only way for this to occur is if BOD increases and POS does not - as they can both increase or only BOD increase. If neither there is no change, and it is not possible for POS to increase without BOD. Therefore the following must be true.

$$\frac{POS - P_iBOD}{M_i + BOD} < \frac{POS - P_iBOD - P_i}{M_i + BOD + 1} \quad (48)$$

However, this is clearly a contradiction the expression on the right is clearly smaller (the numerator is smaller as P_i is positive, and the denominator is larger).

CLAIM 4: Precision increases if and only if c increases.

Follows directly from claims 1-3.

CLAIM 5: When adding more samples that satisfy the body of the rule, confidence increases if and only if POS increases.

Note that confidence is defined as POS/BOD. Clearly, there confidence decreases if BOD increases but not POS and it is not possible for POS to increase alone. Therefore, BWOC, the following must hold true.

$$\frac{POS + k}{BOD + k} < \frac{POS}{BOD} \quad (49)$$

$$BODk < POSk \quad (50)$$

$$BOD < POS \quad (51)$$

This is a contradiction as $BOD \geq POS$.

Going other way, suppose BWOC confidence increases but POS does not. We get:

$$c_2 > c_1 \quad (52)$$

$$\frac{POS}{BOD_2} > \frac{POS}{BOD_1} \quad (53)$$

$$BOD_1 > BOD_2 \quad (54)$$

However, by the statement, as we add more samples that satisfy the body of the rule, we must have $BOD_1 \leq BOD_2$. Hence a cotnradiction.

CLAIM 6: Recall increases if and only if POS increases.

As we can write the new recall in this case simply as the following, the claim immediately follows.

$$\frac{TP_i + POS}{TP_i + FN_i} \quad (55)$$

CLAIM 7: Recall increases if and only if c increases.

Follows directly from claims 5-6.

Proof of theorem.

Follows directly from claims 4 and 7. □

A.4. Overall Accuracy Results for Other Data Splits

Previous work such as [4] is known to have data leakage based on the split between training and test primarily due to segments of a movement sequence existing in both training

	No Overlap		Segment Overlap		Data point Overlap	
	Random	Sequential (least leakage)	Random (known leakage, prev. studies)	Sequential	Random	Sequential
LRCNa (ours)	0.747	0.751	**0.971**	0.758	**0.921**	0.760
LRCNa+EDCR (ours)	0.759 (+1.6%)	**0.763** (+1.6%)	**0.971** ($\pm$ 0%)	**0.769** (+1.5%)	**0.921** ($\pm$ 0%)	**0.780** (+2.6%)
LRCN (prev. SOTA)	0.749	0.747	0.952	0.767	0.887	0.774
LRCN+EDCR (ours)	**0.761** (+1.6%)	0.760 (+1.7%)	0.952 ($\pm$ 0%)	0.768 (+0.1%)	0.889 (+0.2%)	0.783 (+1.1%)
CNN	0.742	0.755	0.851	0.763	0.853	0.779
CNN+EDCR (ours)	0.743 (+0.1%)	0.755 ($\pm$ 0%)	0.866 (+1.8%)	0.763 ($\pm$ 0%)	0.862 (+1.0%)	0.779 ($\pm$ 0%)

Table 3
Accuracy when all classes are represented in training and test sets under various data leakage cases. EDCR means "error detecting and correcting rules" were used on the model output and numbers in parens show the percent change in accuracy from EDCR over the base model. Bold numbers are the best in each case.

and test sets resulting from ransom assignment to each. To address this data leakage issue, we examine our algorithms under various conditions based on ordering and overlap. For ordering, we examine random (which can allow previous behavior of the same agent in the training set, as in previous work) and sequential (which orders the agents to avoid this issue). For overlap, we examine no overlap between the training and test sets, segment overlap that allows training and test samples to overlap each other(as in previous work), and data point overlap (that allows for data points of a trajectory to span both training and test). In Table 3 we examine the accuracy of each model, both with and without EDCR. Models enabled with EDCR performed the same or better with improvement most noticeable when samples are sequential (which has less data leakage between training and test). In terms of overall performance, LRCNa with EDCR performed the best in five of six cases with LRCN with EDCR performing the best in the sixth. Of particular importance, in the "no overlap - sequential" case - the least likely to exhibit data leakage - EDCR improves the performance of both LRCNa and LRCN, 1.6% and 1.7% respectively.