Machine Learning (2025) 114:250
https://doi.org/10.1007/s10994-025-06879-x

Integrating human knowledge for explainable AI

Eleonora Cappuccio[1] · Bahavathy Kathirgamanathan[2] · Salvatore Rinzivillo[1] · Gennady Andrienko[2,3] · Natalia Andrienko[2,3]

Received: 3 April 2025 / Revised: 7 July 2025 / Accepted: 25 August 2025

Abstract
This paper presents a methodology for integrating human expert knowledge into machine learning (ML) workflows to improve both model interpretability and the quality of explanations produced by explainable AI (XAI) techniques. We strive to enhance standard ML and XAI pipelines without modifying underlying algorithms, focusing instead on embedding domain knowledge at two stages: (1) during model development through expert-guided data structuring and feature engineering, and (2) during explanation generation via domain-aware synthetic neighbourhoods. Visual analytics is used to support experts in transforming raw data into semantically richer representations. We validate the methodology in two case studies: predicting COVID-19 incidence and classifying vessel movement patterns. The studies demonstrated improved alignment of models with expert reasoning and better quality of synthetic neighbourhoods. We also explore using large language models (LLMs) to assist experts in developing domain-compliant data generators. Our findings highlight both the benefits and limitations of existing XAI methods and point to a research direction for addressing these gaps.

Keywords Knowledge-guided explainable AI (XAI) · Visual analytics · Trustworthy AI

1 Introduction

Explainable AI (XAI) systems often produce explanations that, while technically accurate, fail to align with domain experts' understanding and reasoning processes (Liao & Varshney, 2021; Abdul et al., 2018). This misalignment occurs because conventional machine learning (ML) and XAI approaches lack systematic mechanisms for incorporating domain expertise, relying instead solely on data-driven methods that often fail to capture meaningful domain concepts and relationships (von Rueden et al., 2023).

We hypothesize that systematic integration of domain knowledge can make ML models more understandable to humans by ensuring that both model behaviour and explanations

Editors: Riccardo Guidotti, Anna Monreale, Dino Pedreschi.

Extended author information available on the last page of the article

align with expert mental models (Simkute et al., 2022; Kulesza et al., 2013). However, achieving this integration remains a critical challenge. While there exist some approaches to incorporating knowledge, e.g., physics-informed neural networks (Raissi et al., 2019) or domain-constrained architectures (Muralidhar et al., 2018), they require customized algorithms and substantial manual engineering, limiting their applicability in general-purpose ML pipelines.

This paper addresses the following research question: *How can domain expert knowledge be incorporated into both ML models and their explanations without modifying existing, widely used ML and XAI algorithms?*

We tackle this question by addressing two technical challenges. First, standard ML pipelines rely on statistical feature extraction from raw data and do not reflect how experts conceptualize data structures. For instance, time series data are typically processed as sequences of points, whereas experts often interpret them through higher-level episodes or patterns. We argue that the standard ML workflow `Raw Data` → `Automated Feature Extraction` → `Model Training` should be enhanced with expert-guided data structuring: `Raw Data` → `Expert Knowledge Integration` → `Structured Representations` → `Model Training`.

Second, many XAI methods work by creating synthetic neighbours around a data point to understand how the model behaves in that local area (Guidotti et al., 2019b; Adadi & Berrada, 2018). However, these synthetic examples often violate domain-specific constraints, resulting in implausible or misleading explanations. We propose to modify the current XAI approach `Query Instance` → `Random Synthetic Neighbours` → `Model Predictions` → `Explanations` by embedding domain knowledge into neighbourhood generation: `Query Instance` → `Domain-Aware Synthetic Neighbours` → `Model Predictions` → `Realistic Explanations`. The resulting explanations are expected to be logically consistent with expert knowledge.

In summary, our proposed framework (outlined in Section 3 following the review of related work in Section 2) incorporates domain knowledge at two stages: (1) through visual analytics-supported data structuring and feature engineering during model development (Section 3.1), and (2) through domain-constrained neighbourhood generation during explanation (Section 3.2). This approach enables domain experts to meaningfully shape the model and its interpretability, while remaining compatible with standard, widely used ML and XAI tools.

We validate the framework through two case studies (Section 4): COVID-19 incidence prediction (Section 4.1) and vessel movement pattern recognition (Section 4.2). Results are evaluated and discussed in Section 6.

Additionally, we explore the use of large language models (LLMs) as collaborative agents in this process (Section 5). Our experiments show that LLMs can acquire domain knowledge through interactive prompting and apply it to generate realistic, constraint-respecting synthetic data. In Section 6.2.2, we discuss the potential for extending this capability to enhance explanations generated by algorithmic XAI methods. Section 7 concludes the paper and outlines directions for future research.

2 Related work

We review related work in three interconnected areas: incorporating expert knowledge into ML models, integrating prior knowledge in explanations, and interactive ML as a foundation for human-AI collaboration.

2.1 Incorporating expert knowledge in ML development

Informed machine learning (von Rueden et al., 2023; Karpatne et al., 2017) aims to integrate prior knowledge into model training through formalized structures such as logic rules or physical equations. Knowledge graphs (KGs) have emerged as a powerful approach to embed structured domain knowledge into machine learning workflows. By representing entities, relations, and domain constraints, KGs can improve model interpretability and support tasks such as feature extraction, concept inference, and information retrieval, making them particularly relevant in explainable AI systems (Rajabi & Etminani, 2024). However, most KG-based approaches assume that such a graph is already available. In many practical domains, this is not the case. Creating a KG from scratch requires the elicitation of expert knowledge, followed by formalization into machine-interpretable structures. This process typically demands significant effort from skilled knowledge engineers (Kidd, 1987). As past experiences with development of expert systems (Buchanan et al., 2006) have shown, this bottleneck often limits the deployment of knowledge-intensive AI systems in practice.

Instead of requiring formal knowledge engineering beforehand, we propose a human-in-the-loop workflow in which domain experts directly shape and test their knowledge through feature engineering. We support this process with interactive visual analytics tools that help experts formulate meaningful features, inspect their impact on model behaviour, and iteratively refine their representations. In this way, we lower the barrier to incorporating expert knowledge into the machine learning pipeline without requiring formal ontology design or structured knowledge encoding.

Visual analytics offers a range of tools for knowledge-driven data structuring. Existing VA systems have demonstrated the value of expert interaction in model development workflows, for instance, via interactive feature selection, hyperparameter tuning, or comparative model assessment (Sacha et al., 2019; Endert et al., 2017; Lu et al., 2017; Wang et al., 2024). However, most focus on supporting user feedback rather than capturing tacit domain expertise for downstream explainability. Earlier work on conceptual model development through VA (Giabbanelli & Jackson, 2015; Zhao et al., 2018) shares our goal of knowledge externalization but lacks direct connections to ML contexts.

Our work builds on these foundations by enabling experts to encode domain semantics through data representations and by evaluating model behaviour iteratively, thereby preparing a basis for human-aligned explanations without altering the learning process itself.

2.2 Incorporating domain knowledge into XAI methods

A wide range of local explainable AI (XAI) techniques have been developed. Widely used are feature attribution methods, such as `LIME` (Ribeiro et al., 2016) and `SHAP` (Lundberg & Lee, 2017), which quantify the contribution of each feature to a specific prediction. However, feature attribution scores are limited in their ability to convey semantic structure or

domain-relevant logic. A list of feature weights does not tell the user how features interact or why a prediction was made.

In contrast, explanation methods based on rules or decision trees, such as ANCHOR (Ribeiro et al., 2018) and LORE (Guidotti et al., 2019a), offer richer semantics through IF-THEN statements that express local decision logic. These structures better align with how humans conceptualize categories. ANCHOR produces high-precision rules focused on sufficiency conditions, but often omits broader feature context, making its explanations potentially brittle. LORE learns a local surrogate decision tree that captures both primary and alternative outcomes by generating a synthetic neighbourhood through controlled perturbation.

Although widely used for their model-agnostic nature, such post-hoc methods often overlook domain-specific feature relationships, resulting in implausible or misleading explanations. We address this by constraining the perturbation process using domain-aware instance generators, thereby improving plausibility without modifying the XAI method itself. We build on LORE due to its extensible architecture, which supports externally defined neighbourhood generation, which is an essential capability for integrating domain knowledge into explanation.

Domain knowledge could, in principle, also be integrated into the rule extraction process of ANCHOR by enforcing such constraints during the growth of explanation rules. However, this would likely require a substantial re-implementation of the algorithm, along with optimizations to its rule induction mechanism. Similarly to LORE, LIME could also benefit from a custom neighbourhood generation process that integrates domain knowledge. However, implementing such functionality would require extending or re-engineering the library, as its current design does not natively support this level of customization.

In addition to improving the generation of explanations, we also focus on how they are presented and validated by users. Visual analytics (VA) supports this by enabling interactive exploration of explanation structures and helping users relate model behaviour to domain knowledge. For example, systems like RuleMatrix (Ming et al., 2019) and explAIner (Spinner et al., 2020) allow users to inspect and manipulate explanation rules, while others facilitate exploration of instance clusters, latent spaces (Hohman et al., 2019), and model errors (Cheng et al., 2021). VA enhances cognitive alignment by contextualizing explanations and supporting iterative refinement, making it a natural complement to rule-based explanation methods.

2.3 Interactive machine learning and human-AI alignment

Our methodology aligns with principles of explanatory interactive machine learning (IML), where human input is integrated iteratively not only during training but also during explanation refinement. Teso and Kersting (2019) formalize this approach, showing how explanatory feedback can guide models toward more interpretable behaviour. Bhattacharya et al. (2024) extend this line of work by combining multifaceted explanations and data transformations within interactive VA environments closely matching our dual-loop interaction pattern.

Moreover, Schramowski et al. (2020) demonstrate that expert feedback on explanations can steer models away from spurious correlations, reinforcing the importance of incorporating domain knowledge during both training and explanation. These studies empirically vali-

Table 1 Comparison of related approaches to our framework

Approach	Focus	Limitations addressed by our work
Informed ML von Rueden et al., (2023)	Formal knowledge integration	Requires structured prior knowledge
VA for ML Sacha et al., (2019); Endert et al., (2017)	Interactive model refinement	Limited support for knowledge encoding
XAI methods (e.g., LIME, SHAP, ANCHOR)	Instance-level explanations	Synthetic data may violate domain constraints
LORE Guidotti et al., (2019a)	Rule-based local modeling	Lacks expert-driven neighborhood control
Explanatory IML Teso & Kersting, (2019); Bhattacharya et al., (2024)	Iterative explanation refinement	Our work applies this in model-agnostic, VA-driven settings

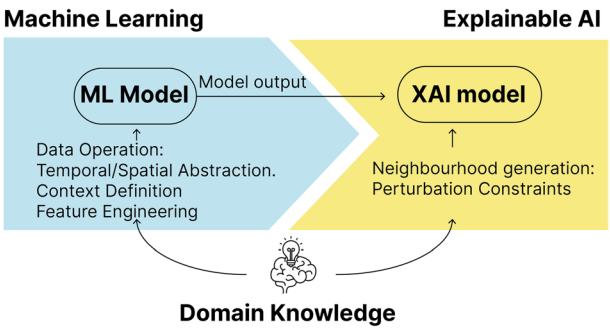

Fig. 1 Integrating human expertise into the ML pipeline. Knowledge is embedded through data structuring and realistic neighbourhood generation, supporting model interpretability and explanation plausibility

date the broader intuition behind our approach: explanation is not just a post-hoc step but part of an iterative human-AI dialogue. Our framework applies the principles of explanatory IML while preserving the modularity of standard ML and XAI pipelines.

Table 1 summarizes the related work and highlights the specific limitations our approach addresses.

3 Methods

Figure 1 illustrates the proposed pipeline for integration of human expert knowledge in ML and XAI: (1) during the ML process, domain knowledge is used to prepare appropriate training data and refine the model; (2) in the explanation phase, this knowledge is further leveraged to ensure explanations align with user perspectives.

3.1 Incorporating human knowledge in models

As stated earlier, we propose a human-in-the-loop workflow where domain experts incorporate their knowledge into ML through feature engineering, supported by interactive interfaces and visual representations. While feature engineering is a standard part of ML workflows

(Dong & Liu, 2018), our contribution lies in offering UI components and visualization techniques that enable experts to (a) apply and manage domain-informed data transformations, (b) assess the representational adequacy of engineered features for the modelling task, and (c) explore and compare model outputs across iterations to identify and address performance issues. The framework is organized around a set of modular operations:

Step 1: Data abstraction – defining analytical units

Raw data often consist of elementary observations tied to individual objects, timestamps, or locations. The first step is to define higher-level units of analysis suitable for the modelling task, e.g., grouping time points into intervals, spatial points into regions, or individuals into groups. The framework provides interactive interfaces to configure such aggregations and view their effects on the data distribution (see, for example, Andrienko et al. (2016, 2023)).

Step 2: Context encoding

Many predictive tasks depend not only on the internal properties of a unit but also on the context in which the unit occurs. Experts can define context windows, e.g., preceding time intervals, neighbouring regions, or connected nodes of a network, and derive features that summarise relevant surrounding conditions. Our tools support this through parametrised transformations and visual representations allowing users to inspect the relevance and informativeness of contextual features (e.g., Peng et al. (2025)).

Step 3: Synoptic feature construction

To capture the internal structure or dynamics of a unit, the system provides functions for computing statistical summaries, trends, variation measures, or custom aggregates. For example, experts can summarise speed variations over a time interval or density distributions within a region. Visualisations help users to verify whether the derived features align with domain-relevant behaviour patterns (e.g., Andrienko et al. (2024)).

Step 4: Iterative model refinement

After model training, experts can compare predictions across model versions using side-by-side views and performance overlays. This feedback loop allows users to detect where features are insufficient or misleading, and return to previous steps to refine transformations or add new features.

This iterative process allows domain knowledge to shape the model indirectly through improved data representation. Rather than encoding knowledge explicitly in rules or ontologies, experts interactively refine how data are structured, contextualised, and summarised to better capture relevant behaviour patterns. Our framework supports this process through interfaces drsigned for expert-guided data transformation and responsive visual feedback. Many of these tools have been introduced in our prior work. Some papers are cited above, but a comprehensive overview is beyond the scope of this paper. Instead, we focus on two case studies where temporal abstraction was used to craft meaningful features for interpretable model development.

3.2 Incorporating human knowledge in explanations

In our work, we focus on generating *local explanations* telling how a model obtains a specific prediction for a given instance. We build on the rule-based explanation method `LORE` (LOcal Rule-based Explanation) (Guidotti et al., 2019a), which approximates the model's

decision boundary in the local neighbourhood of the instance via a transparent surrogate model. The explanation takes the form of logical rules:

(1) a **factual rule**, which identifies the key factors leading to the prediction, and
(2) a set of **counterfactual rules** describing minimal changes in feature values that would alter the prediction.

Both rule types are represented as logical conjunctions in the antecedent with a predicted class as the consequent. Counterfactual rules are particularly aligned with human reasoning styles, as they provide contrastive examples encouraging the users to explore alternative scenarios that could arise by adjusting input features. Presentation of counterfactual rules is widely recognised in human-centred XAI research (Miller, 2019; Riveiro & Thill, 2021).

Explanations are more effective when they reference concepts familiar to the user. By relying on features engineered through expert-driven abstraction and contextualisation (see Section 3.1), we expect that our explanations align closely with domain knowledge and thus can be well understood by users.

Neighbourhood generation in LORE

To derive local rules, `LORE` generates a synthetic neighbourhood around the instance of interest. These synthetic instances are created by perturbing feature values and then querying the black-box model to obtain predictions. The resulting labelled instances serve as training data for a transparent surrogate model (e.g., a decision tree), which learns to approximate the black-box model's local decision boundary.

Neighbourhood generation in `LORE` uses a genetic algorithm with the following steps:

(1) Initialization: A seed population is formed by replicating the input instance x.
(2) Mutation and Crossover: At each generation, feature values are randomly modified (mutation) or recombined from pairs of instances (crossover).
(3) Selection: Individuals with the highest fitness are selected for the next generation. Fitness is computed based on (a) feature similarity (closeness to x in input space) and (b) target similarity (closeness in predicted class).
(4) Dual Objective Optimization: Two sub-populations are evolved separately:

- Instances similar to x with the same predicted class.
- Instances similar to x with a different class.

These sub-populations are merged to form the local training dataset for the surrogate model.***Incorporating human knowledge via constraints***

To ensure that generated explanations are meaningful to domain experts, we extend `LORE` with domain knowledge through neighbourhood constraints, guiding the synthetic data generation process:

- **Hard Constraints:** Following the **data abstraction** and **contextualization** steps, the synthetic instances need to adhere to the same structural properties. Hard constraints represent these properties. Any synthetic instance violating these constraints (e.g., temporal orderings) is discarded.

- **Soft Constraints:** These constraints serve as guidance for the generation process, providing weights and priorities for the features. For example, if a feature is considered more important by the expert, it will be more likely to be altered in the synthetic instances.

This constraint-driven extension allows the generated neighbourhood to better align with the expert's mental model and domain expectations.

Alternative neighbourhood generators

To assess the effect of constraint-driven generation, we compare against two alternative generators:

- **Random Generator**: A baseline implemented in the `LORE` Python library that perturbs features independently within their value ranges until the neighbourhood reaches the desired size.
- **kNN Generator**: A new baseline introduced by us, which selects the k nearest neighbours from the original training set based on feature distance. This method does not generate synthetic data but instead reuses existing instances. It is applicable only when suitable labelled data (e.g., the model training data) are available.

4 Case studies

To evaluate the practical utility and generalizability of our proposed methodology, we conducted two case studies in distinct domains: (1) prediction of COVID-19 incidence level changes using temporal context features derived from prior disease trends and mobility data, and (2) classification of vessel movement patterns based on trajectory features. The case studies serve three primary purposes: (1) to demonstrate how expert knowledge can be integrated into the ML pipeline through visual analytics-supported data preparation and synthetic neighbourhood generation, (2) to assess the impact of knowledge-informed instance generation on explanation quality, and (3) to reveal possible limitations of our approach. Importantly, we conduct these investigations without modifying the core ML or XAI algorithms, thereby revealing how much improvement can be achieved within existing frameworks by embedding domain knowledge at the data and representation levels.

For both case studies, we employed Random Forest as our classification method due to its robust performance characteristics and practical advantages. This ensemble technique combines multiple decision trees to deliver high accuracy while efficiently capturing complex non-linear relationships and feature interactions. However, our framework is general and can be applied with any ML method.

4.1 Case study 1: COVID-19 and population mobility in Spanish provinces

This case study uses data collected in Spain during the COVID-19 pandemic to predict upcoming COVID-19 incidence levels based on trends in disease and population mobility in preceding weeks. The dataset includes daily counts of COVID-19 cases and intra-provincial mobility across 52 provinces over 64 weeks, derived from open sources (Ponce-de Leon et al., 2021a, b, c, d).

4.1.1 Incorporating domain knowledge in model development

Step 1: Data abstraction – defining analytical units

Raw daily time series were smoothed with a 7-day moving average to reduce noise and weekly cyclicity. The smoothed data were segmented into weekly episodes, which were categorized into four ordinal levels for COVID-19 severity (c1 to c4) and mobility (m1 to m4). Consecutive episodes with the same level were grouped into *events* defined as coherent intervals of high or low incidence or mobility (Fig. 2).

Step 2: Context encoding

Each COVID-19 event was assigned a temporal context defined by the COVID-19 and mobility levels over the six preceding weeks. This window length was chosen based on epidemiological reasoning: the effects of mobility changes on disease incidence typically manifest with a delay of about three weeks, corresponding to the incubation period and reporting lag. An additional three weeks were included to capture trends and make directional changes in incidence or mobility more detectable. The resulting context encodes short-term temporal dynamics relevant for predicting the severity of upcoming COVID-19 events (Fig. 3).

Step 3: Synoptic feature construction

The context window was converted into 12 categorical features (6 weeks $\times$ 2 variables), describing the sequence of COVID-19 and mobility levels. An additional continuous feature, "days since pandemic onset", was added to account for broader temporal effects.

Step 4: Iterative model refinement

A random forest classifier was trained to predict the level (c1–c4) of each COVID-19 event from its context. Visual analytics was used to inspect model predictions, identify temporal or regional misclassifications, and guide refinement. Through human-in-the-loop iteration, we introduced targeted improvements:

- Temporal and spatial decomposition (e.g., isolating early pandemic phases, separating island provinces)
- Filtering outliers and non-representative periods
- Model comparison via visualization of misclassification patterns across time and provinces (Fig. 4)

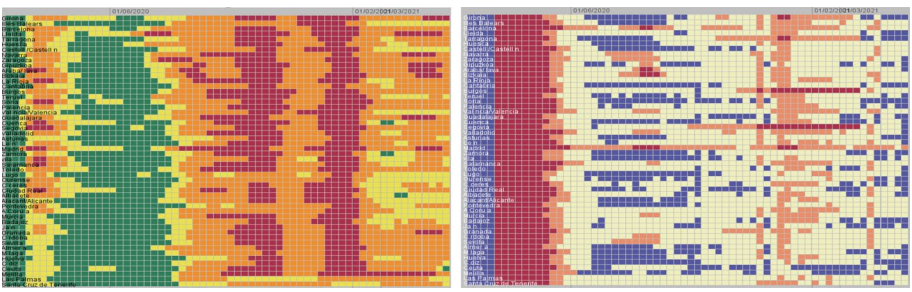

Fig. 2 Visualisation of weekly episodes derived from time series of COVID-19 incidence and population mobility in provinces of Spain. The horizontal dimension represents time, the rows correspond to the provinces, and the cells represent the episodes coloured by the levels of disease incidence (left, from green for low to red for high) and mobility (right, from red for very low to blue for normal)

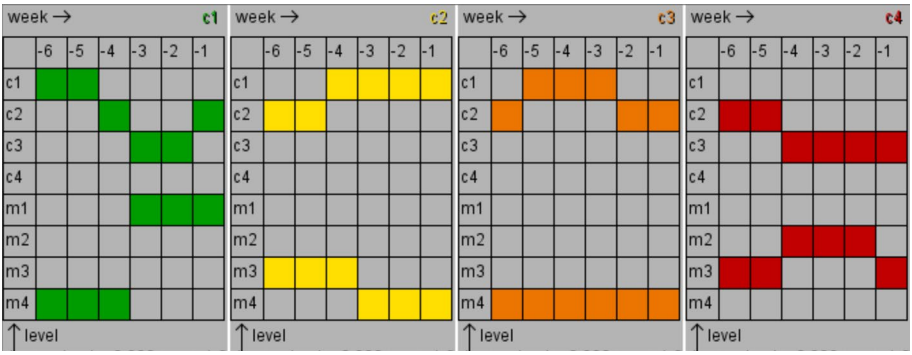

Fig. 3 Visual representation of temporal contexts of four selected examples of pandemic events from different classes by disease incidence. The columns of the matrices correspond to the weeks from -6 to -1 preceding the events. Each row corresponds to one level of either COVID-19 (top 4 rows) or mobility (lower 4 rows). The filled cells signify which levels of COVID-19 and mobility were attained before the events started. The colours of the filled rectangles reflect the classes (levels) of the represented events

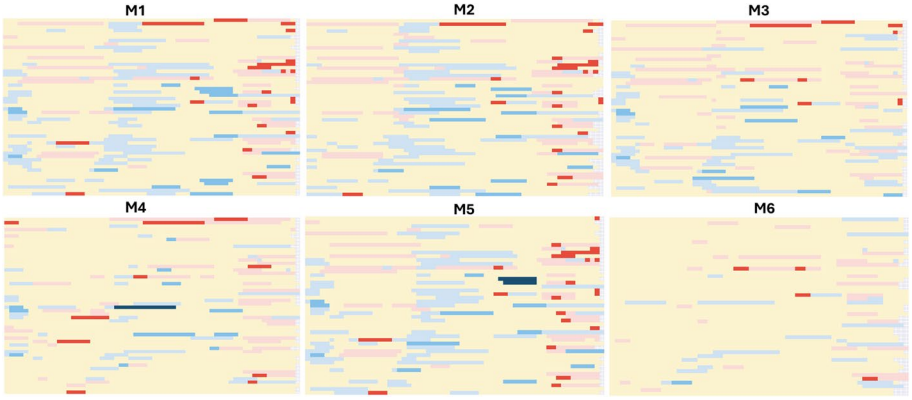

Fig. 4 Comparison of different versions of the pandemic level model by visual inspection of the distributions of model errors over time (horizontal dimension) and provinces (vertical dimension). The events are represented by horizontal bars with model errors encoded by the colours. Shades of blue and red signify, respectively, under and over-estimation of the pandemic event level; the darkness is proportional to the difference between the predicted and actual levels

This process allowed domain experts to iteratively inject knowledge into both data representation and model structure, improving both interpretability and performance beyond what could be achieved with raw data and automatic learning alone.

4.1.2 Incorporating domain knowledge into neighbourhood generation

During the data preparation phase, we identified internal constraints and relationships within our data that could not be captured by traditional neighbourhood generators. These included time-related patterns and the prevalence of gradual rather than abrupt changes over time. We developed a customized neighbourhood generator designed to account for these dependencies and constraints. Each event record comprises three sections: (1) attribute describing

temporal positioning, (2) epidemic severity classes for COVID-19, and (3) mobility intensity classes. While the legacy neighbourhood generator used in LORE randomly selects feature values to perturb within a predefined radius, we want to restrict this perturbation based on the internal properties of our data, ensuring that feature values for COVID-19 and mobility remain consistent with domain-specific constraints. For example, abrupt transitions from a low epidemic level to the maximum level are avoided, as are inconsistencies like abrupt level drops in sequences of high-incidence epidemic classes. Within the LORE pipeline, this neighbourhood generation step is where domain-specific rules can be incorporated.

We defined a set of **soft constraints** to guide the perturbation process, ensuring that generated data remain realistic and domain-consistent. Our approach is based on the observation that COVID-19 and mobility levels from previous weeks strongly influence subsequent levels. For instance, a high epidemic level is more likely to persist or transition to another high level in the following week. To capture these dependencies, we modelled transitions using probability distributions derived from training data. These transitions are encoded in a transition matrix that governs how feature values change over time. Specifically, we define a 5×5 matrix A, where each element $a_{i,j}$ represents the probability of transitioning from state s_i to state s_j. Figure 5 illustrates examples of these matrices for COVID-19 and mobility features, highlighting differences in transition patterns. Bipartite graphs further visualize weighted transition probabilities, offering an intuitive representation of these relationships. We exclude class 0 (missing or undefined values) to ensure valid record generation.

To estimate the transition probabilities, we trained a Bayesian network on the real-world data. This network models dependencies between consecutive weeks as a directed graph, where each state influences the next (e.g., $P(\text{week}^n = s_l \mid \text{week}^{n-1} = s_k)$, with l and k representing COVID-19 or mobility levels). Conditional probabilities were averaged across weeks to obtain stable transition probabilities for each level pair. The resulting transition matrices serve as the basis for perturbations, preserving domain knowledge and generating consistent neighbourhoods.

This approach is computationally efficient: since transitions rely on precomputed probabilities stored in index-based data structures, perturbation lookup operates in constant time, regardless of the number of generated instances. This ensures that the method integrates seamlessly with the legacy algorithm without increasing its computational complexity. The quality of the generated neighbourhoods is evaluated in Section 6.

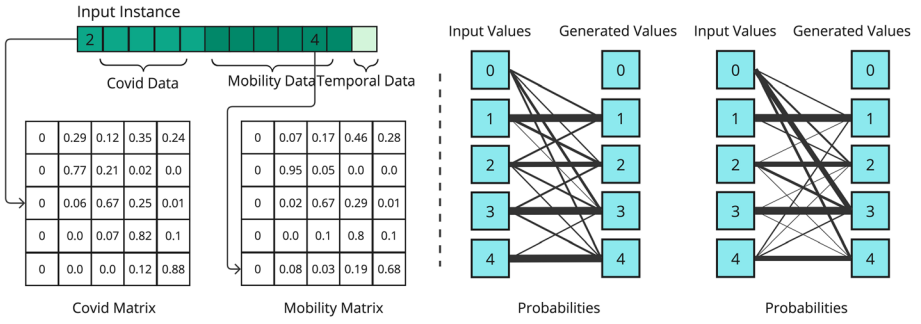

Fig. 5 On the left: The COVID-19 and Mobility transition matrices used as constraints to generate neighbourhoods. On the right: Bipartite graphs with the probabilities for COVID-19 (Left) and Mobility (Right) state transitions. The thickness of each arc is proportional to the probability of the transition

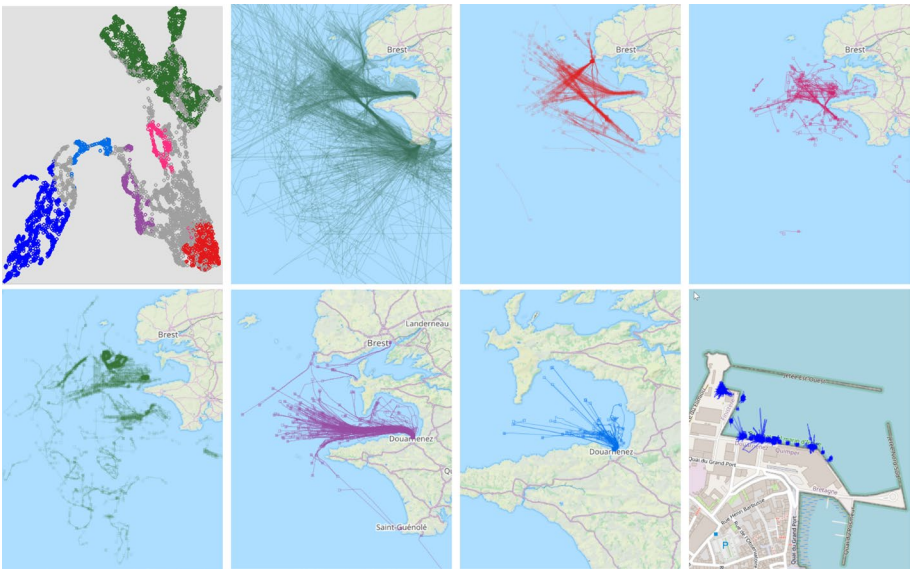

Fig. 6 Visually supported preparation of data for recognition of vessel movement patterns. Upper row, from left to right: 1) UMAP projection of episodes based on the generated features; 2) complete original trajectories; 3) episodes of straight movement; 4) episodes of curved movement. Lower row, from left to right: 5) trawling; 6) port-connected; 7) near port; 8) anchored

4.2 Case study 2: vessel movement pattern recognition

This case study addresses the classification of short vessel trajectory segments into six types of movement behaviour: 1 - straight movement, 2 - curved movement, 3 - trawling, 4 - port-connected (entering or exiting a port), 5 - near port (manoeuvring near a port), and 6 - anchored. The dataset comprises trajectories of 71 fishing vessels operating north-west of France between October 1, 2015, and March 31, 2016, sourced from an openly available repository[1]. The six movement types are illustrated in Fig. 6.

4.2.1 Incorporating domain knowledge in model development

Step 1: Data abstraction – defining analytical units

The raw vessel trajectories were segmented into overlapping 3-hour episodes with a 1-hour shift between consecutive segments. This window length balances temporal context and segment resolution: three hours is typically sufficient to capture operational patterns such as trawling or anchoring. Overlaps ensure that important patterns are not lost at segment boundaries, enabling better continuity and behavioural interpretation. Each episode becomes a single data instance for subsequent classification.

Step 2: Context encoding

The key contextual factor in this domain is proximity to port, as it strongly influences vessel behaviour. To incorporate this context, the minimum and maximum distance from the nearest port were computed for each episode using an external dataset of port coordinates.

[1] https://zenodo.org/records/1167595

These two features – `Log10MinDistPort` (log-transformed) and `MaxDistPort` – capture whether the vessel is performing port-related manoeuvres or operating farther offshore.

Step 3: Synoptic feature construction

We generated 7 synoptic features characterising the vessel movement during the lifetime of each episode. They summarize speed distribution, movement direction, and trajectory complexity:

- `SpeedMinimum`, `SpeedQ1`, `SpeedMedian`, `SpeedQ3`: Statistics of speed smoothed over a 5-minute window, representing the distribution from minimum to upper quartile.
- `Log10Curvature`: Logarithm of the curvature of the vessel's distance-from-start time series, computed as the ratio between the sum of absolute consecutive changes and the amplitude of values. A value close to 1 indicates straight-line movement; higher values indicate loops or turns.
- `DistStartTrendAngle`: Angle of the fitted linear trend to the distance-from-start time series, capturing the directional tendency of movement away from the starting point.
- `Log10DistStartTrendDevAmplitude`: Logarithm of the amplitude of deviations from the fitted trend line, quantifying local path tortuosity or zigzagging.

Logarithmic transformations were applied to features with skewed distributions (namely curvature, deviation amplitude, and minimum port distance) to improve separability and model performance.

`Log10Curvature` and `Log10DistStartTrendDevAmplitude` jointly describe the shape complexity of a movement episode: the former reflects global movement geometry, while the latter captures local irregularities. Together, they support fine-grained discrimination between behaviours such as steady outward navigation, trawling, and in-place manoeuvring.

Feature effectiveness was evaluated using dimensionality reduction (UMAP; see Fig. 6, top left), which revealed distinct clusters of movement patterns. These clusters were examined through interactive map-based visualization, and representative episodes were labelled by domain experts.

Step 4: Iterative model refinement

The process of feature design and evaluation was iterative: a k-nearest neighbours (kNN) classifier was used as a rapid probe to assess the discriminative power of evolving feature sets. When results were inconclusive, new features were designed or refined to enhance behavioural distinction. After achieving good results with kNN, a random forest classifier was trained using 5-fold cross-validation with a fixed random seed (0) to ensure reproducibility. The final model achieved 98% accuracy, confirming that the constructed synoptic and contextual features were sufficient to accurately distinguish movement patterns without additional post-hoc tuning.

The process of defining movement episodes and feature engineering in this case study is described in more detail in our earlier publication (Andrienko et al., 2024).

4.2.2 Incorporating domain knowledge into neighbourhood generation

Following the feature engineering described in the previous section, we define two sets of constraints to guide the neighbourhood generation process:

- Hard Constraints: the speed features should be consistent with the internal dependencies. We discard or correct those instances that do not satisfy these internal consistencies.
- Soft Constraints: We introduce a preferential selection of features to be perturbed based on their importance for each target class.***DT-based generator***

This generator designs a mechanism to select the features to be perturbed based on a set of domain-specific relationships among the feature values. Since these relationships are too complex to be expressed through a set of precise statements by the expert, we introduce a method based on decision trees (DT) by leveraging the knowledge embedded in the training data. For each class c, we construct a DT to distinguish instances of this class from all others. The training is performed on a modified version of the training dataset, where instances belonging to class c are labelled as positive examples (1), while all others as negative examples (0). The resulting tree serves as a representation of the relationships that discriminate class c from the rest.

Given the forest of DTs, when an instance x is processed, it is first classified by the black box model. The predicted class determines the appropriate binary DT in the forest, which is then used to classify x and retrieve the decision rules along its classification path. Figure 7 provides a schematic representation of this process. The features involved in the decision path are prioritized for perturbation to generate the neighbourhood. In the toy example shown in Fig. 7, the three attributes used by the tree are given a higher probability of being perturbed. This **selective perturbation** process is used to incorporate the **soft constraints** into the perturbation process. The set of **hard constraints** ensures the correct relationships between the values of the speed features, namely: `SpeedMinimum` $\leq$ `SpeedQ1` $\leq$ `SpeedMedian` $\leq$ `SpeedQ3`. Features not selected by the binary DTs can still be perturbed with a lower probability. The computational cost of this selective perturbation process remains comparable to the baseline random generator, with a slight one-time overhead due to training the forest of DTs. The process involves a single classification by a DT, followed by selective perturbation of the identified features.

Genetic optimization of DT-based generator

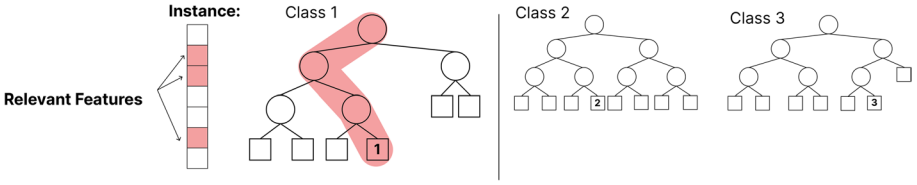

Fig. 7 Example of a forest of decision trees (DT) used to define the probability of perturbation of relevant features for an instance. Given an input instance x, classified as 'Class 1' (Left), the corresponding decision tree provides the features that contribute to its classification. These features, highlighted in red, are then used to perturb x

We improved our DT-based generator using a genetic optimization process to improve performance. While retaining the fitness functions from LORE, we introduced custom mutation and crossover operators. During the genetic optimization, the operation of *mutation* and *crossover* are limited by the same set of **hard constraints** and **soft constraints** as the DT-based generator. The genetic optimization improves the generation of opposite classes and the quality of the generated neighbourhoods.

5 LLM-supported generation of synthetic neighbours

Domain experts may struggle to incorporate their specialised knowledge into synthetic data generation without support from skilled data engineers. We have investigated how large language models (LLMs) can bridge this gap by enabling non-technical domain experts to collaboratively create, refine, and validate synthetic data generation methods. Through this approach, LLMs serve as intelligent intermediaries that translate domain expertise into actionable data generation processes, making synthetic data creation easier and more efficient for experts.

5.1 COVID-19 and mobility: an experiment with ChatGPT

We conducted an experiment with using GPT-4 Turbo (ChatGPT-4o) for generating synthetic neighbourhoods in a COVID-19 and mobility prediction task. Here we provide a brief summary. A more detailed description is available in Appendix A.

Through an iterative dialogue with the LLM, we specified requirements for the generator, emphasizing the need for both randomness and domain consistency. The LLM produced Python code that integrated key domain knowledge in the form of transition probabilities between COVID-19 and mobility levels. After assessing the first result, the LLM was further requested to increase the diversity of the generated instances to ensure predictions to cover all target classes.

The final Python code was tested using 15 real input instances. For each, 100 synthetic neighbours were generated. These were evaluated using the predictive model and visualised using UMAP, which produced a 2D projection of the instances according to their similarity. Visualisations revealed that synthetic neighbours clustered around their original instances while also exhibiting sufficient diversity to yield varied model predictions. This confirmed the generator's ability to produce plausible and informative data variations.

The experiment was judged successful, showing the potential of LLMs to act as interactive coding assistants for building data generators guided by expert knowledge. However, some challenges were revealed. Translating informal domain insights into precise prompts often required reformulation. The generated code needed validation and debugging. These tasks demand coding skills, which domain experts may lack. Visual analytics proved helpful for evaluating neighbourhood quality, but further tools are needed to assess the realism of feature combinations more systematically.

5.2 Vessel activities: an experiment with Google Gemini

For the vessel movement case study, we conducted an experiment using Google Gemini version 2.5 Flash. Using a textual prompt describing domain-specific constraints and feature characteristics, the LLM generated a Python implementation with methods to produce class-targeted, constraint-compliant synthetic neighbours. The generator integrates expert knowledge into the perturbation process through the following mechanisms:

- Feature-constrained perturbation: Feature values are perturbed within reasonable ranges while preserving known structural relationships:
  - Speed statistics (`SpeedMinimum`, `SpeedQ1`, `SpeedMedian`, `SpeedQ3`) must maintain ascending order and remain within the empirical upper bound (22.0 knots).
  - Curvature-related features such as `Log10Curvature`, `DistStartTrendAngle`, and `Log10DistStartTrendDevAmplitude` are constrained to lie within historically observed ranges.
  - A domain rule is enforced: high speed cannot co-occur with high curvature.
  - Speed- and distance-related features are kept within valid, non-negative domains, while `Log10MinDistPort` is kept above a known minimum of -3.05.
- Class-tendency-guided generation: The generator strives to produce synthetic instances biased toward each of the six behaviour classes. For example, instances simulating "anchored" behaviour are generated with very low speed and high curvature. This mechanism supports counterfactual exploration and class separation analysis.
- Balanced neighbourhood construction: For a given input instance, the generator produces a user-specified number of synthetic neighbours. It balances the output across all six class tendencies, ensuring that the resulting neighbourhood exhibits class diversity.

A more detailed description, including evaluation of generated synthetic instances by means of visual analytics techniques, is available in Appendix B.

This experiment again illustrates that LLMs can translate high-level expert instructions into executable, constraint-aware code for synthetic data generation. Similar to the COVID-19 case study, the generated neighbourhood supports downstream tasks such as explanation generation and model testing. Both experiments demonstrate the high potential of LLMs in supporting human-in-the-loop model testing and explanation by enabling domain-aware synthetic data generation.

6 Evaluation and discussion

The evaluation of our proposed methodology is structured into two parts. First, we compare the properties of synthetic neighborhoods generated by standard domain-agnostic methods with those produced by our domain-informed approaches. The comparison is guided by three key criteria: realism, locality, and diversity. Our primary research question in this part is whether the incorporation of domain knowledge leads to the generation of higher-quality synthetic data.

In the second part of the evaluation, we assess the explanatory rules generated using synthetic neighbourhoods created by different methods. Our initial hypothesis was that higher-quality synthetic data would enable the derivation of more accurate, meaningful, and trustworthy explanations. However, as we discuss later, this hypothesis was not confirmed. The observed limitations point to more fundamental deficiencies in current algorithmic XAI techniques that derive explanations purely from data.

6.1 Evaluation of synthetic neighbourhoods

To evaluate the impact of the proposed methodology, we conducted comparative assessments of the outcomes of different generators for the two use cases. The goal was to compare each generator's performance in terms of alignment with the expert's ground truth. As a reference, we used the respective training data of both domains based on the intuition that if the generated neighbourhoods aligned with expert knowledge, they should also align with the training data distribution. Consequently, the derived explanation rules would be consistent with both.

We apply two methods of comparison. (1) **Analytical comparison**, where we used a combination of measures to assess the locality, compactness, and plausibility of the produced instances.

(2) **Visualization-based qualitative comparison of features**, comparing for each neighbourhood generator the marginal distribution of features of the synthetic instances with the training data. This comparison has the goal to assess how much the generated instances are consistent with the domain knowledge and how each generator is able to diversify the feature values.

Analytical comparison

Neighbourhood generators are designed to produce a set of points as perturbation of a given instance x, with two primary objectives: (i) ensuring that the generated points exhibit realistic values, i.e., that they are likely to belong to the original training distribution; and (ii) maintaining proximity between the generated instances and x.

To quantitatively assess the compactness criterion, we use the measure as defined in Guidotti et al. (2019a), that we named **instance distance**, computed as the average distance of all generated instances from the original instance x. Formally:

$$D(x, N) = \frac{1}{|N|} \sum_{z_i \in N} d(x, z_i),$$

where $d(x, z_i)$ is the distance between the original instance x and each generated instance z_i in the neighbourhood N. In our implementation, we use the Euclidean distance for this measure, although the choice of distance metric is configurable.

To evaluate the realism of the generated neighbourhoods, we use the Energy Distance (Székely & Rizzo, 2013) and Maximum Mean Discrepancy (MMD) (Gretton et al., 2012), which are commonly employed to compare distributions. The Energy Distance is a statistical metric that quantifies the discrepancy between two probability distributions, while MMD measures the distance between the means of two distributions in a reproducing kernel Hilbert space. Both metrics provide insight into how well the generated neighbourhoods resemble the training data distribution.

We compared the neighbourhoods produced by each generator to the training data distribution. For each instance x, we generated a neighbourhood N of size $n = 2000$ using each generator. This process was repeated multiple times, and for each repetition and each generator, we computed the aforementioned metrics. In all cases, the tests consistently rejected the null hypothesis that the distributions of the generated neighbourhoods and the training data are the same (p-value < 0.001).

Table 2 summarizes the results of the analytical comparison for the Vessel case study. Among the evaluated generators, the *Custom* and *Custom Genetic* generators achieved the lowest instance distance, indicating that they are able to produce compact neighbourhoods that are close to the original instance. The *LLM* generator also performed well, producing a compact neighbourhood and the lowest divergence across the Energy and MMD measures, indicating that it successfully captured the training data distribution. The *Genetic*, *Custom Genetic*, and *Custom* generators have comparable distributions. The *Random* generator, as expected, produced the largest instance distance, indicating that it generated data that is very spread out. The *Baseline* generator, which simply extracts instances from the training data, achieved the lowest Energy distance and it does not produce MMD, as the two distributions are identical. In summary, the *Custom* and *LLM* generators yielded the best results, demonstrating their efficacy in capturing internal data characteristics. This also demonstrates that the constraints defined by the LLM were effective in producing a generator that match the requirements of the domain. From a computational perspective, the *LLM* generator was the most efficient. The *Custom* generator, while slightly slower, is the generator that best balances performance and output quality. The *Custom Genetic* generator, while producing the most compact neighbourhoods, was significantly slower due to the genetic optimization process.

Table 3 summarizes the results of the analytical comparison for the COVID-19 and mobility case study. Since the majority of the features of this dataset are categorical, potential perturbations are limited. This generates a similar performance across all generators, in particular for the *Instance distance* measure. Given the limited variability of the feature space there is no a clear winning generator. For this reason, the analytical comparison provides only a partial view of the performance of the generators. In the next section, we complement this analysis with a qualitative comparison of the feature distributions of the generated neighbourhoods.

Visualization-based qualitative comparison of feature distributions

To better understand the quality of the generated neighbourhoods, we conducted a qualitative comparison of the feature distributions of the synthetic instances produced by each generator. We implemented a visual-based dashboard that provides a comprehensive overview of the generated neighbourhoods. For each feature and for each generator, we display

Table 2 Vessel movements: Comparison of neighborhood methods across Instance distance, Energy, MMD, and Time (mean ± standard deviation)

Generator	Instance distance	Energy	MMD	Time (s)
Random	119.1400 ± 12.0031	111.8990 ± 13.7530	0.0178 ± 0.0014	0.0358 ± 0.0042
Baseline	9.6637 ± 4.8497	5.0910 ± 0.0000	N/A	0.0024 ± 0.0001
Custom	2.2196 ± 0.2118	13.5748 ± 0.0430	0.0399 ± 0.0004	6.5345 ± 0.1702
Genetic	6.0275 ± 4.0885	12.9352 ± 0.1566	0.0575 ± 0.0009	13.4913 ± 0.2668
Custom genetic	1.2714 ± 0.9461	15.0236 ± 0.6921	0.0587 ± 0.0053	52.7897 ± 0.7902
LLM	11.9267 ± 1.5013	8.1266 ± 0.0198	0.0083 ± 0.0002	0.0500 ± 0.0021

Table 3 COVID-19 and mobility

Generator	Instance distance	Energy	MMD	Time
Random	6.2837 ± 0.1386	0.3761 ± 0.0375	0.0052 ± 0.0003	0.3078 ± 0.0400
Baseline	6.2837 ± 0.1386	0.0000 ± 0.0000	N/A	0.0002 ± 0.0000
Custom	6.5898 ± 0.0760	1.7483 ± 0.0187	0.0203 ± 0.0003	0.0482 ± 0.0039
Genetic	6.7421 ± 0.0567	3.2248 ± 0.0500	0.0336 ± 0.0005	20.2679 ± 0.3779
Custom genetic	5.8902 ± 0.0929	4.3411 ± 0.0647	0.0391 ± 0.0004	41.5833 ± 1.0018
GPT	6.4904 ± 0.0657	2.0216 ± 0.0204	0.0253 ± 0.0004	0.4225 ± 0.0440

Comparison of generator variants across Instance distance, Energy, MMD, and Time (mean ± standard deviation)

a row of histograms with the marginal distribution of the feature values. This allows us to visually assess how well the generators are capable of capturing the internal structure of the domain. All the distributions are directly compared with the training data distribution, which serves as a reference for the expected feature values.

The visualization starts from an instance x. For example the charts in Figure 8 show the distributions of the features for a specific instance of the Trawling class from the Vessel dataset. The instance has the following feature values: *SpeedMinimum*=0.91, *SpeedQ1*=2.37, *SpeedMedian*=2.49, *SpeedQ3*=2.78, *Log10DistanceStartShapeCurvature*=0.39, *DistanceStartTrendAngle*=-0.01, *LogDistStartTrendDevAmplitude*=0.70, *MaxDistPort*=27.55, *LogMinDistPort*=1.30.

The columns on the right show the distribution of the training data. However, the generators aim to produce local instances. Thus, it is important to consider the distribution of the *Baseline* generator, which contains elements of the original training data restricted to the local neighbourhood of the instance x.

We observe that the *Random* generator produces a distribution very different from the training data, as it samples randomly from the entire feature space. In contrast, all other generators produce instances that remain closer to the training data distribution. Starting from the distribution of classes in each neighbourhood, we can observe that the *Random* generator produces instances across all possible classes. The *Custom*, *Genetic*, and *Custom Genetic* generators focus on a limited number of classes near x. The *LLM* generator produces a broader distribution of classes, designed to diversify the neighbourhoods.

The speed-related features (*SpeedMinimum*, *SpeedQ1*, *SpeedMedian*, and *SpeedQ3*) show that customized generators reproduce internal relationships and constraints effectively. The *Custom*, *Genetic*, and *Custom Genetic* generators generate narrow value ranges, whereas the *LLM* generator produces distributions closer to the training data.

Similarly, for shape-based features (*Log10DistanceStartShapeCurvature*, *DistanceStartTrendAngle*, and *LogDistStartTrendDevAmplitude*), the *LLM* generator mimics the local distribution of the *Baseline* data effectively. For port proximity features (*MaxDistPort* and *LogMinDistPort*), the *Custom*, *Custom Genetic*, and *Genetic* generators focus on the local neighbourhood of x, while the *LLM* generator explores a wider feature space.

Figure 9 shows the dashboard for the COVID-19 and mobility case study. The instance x used to generate the neighbourhoods has the following feature values: *week6_covid*=c3, *week5_covid*=c3, *week4_covid*=c3, *week3_covid*=c3, *week2_covid*=c2, *week6_mobility*=c4, *week5_mobility*=c3, *week4_mobility*=c4, *week3_mobility*=c4, *week2_mobility*=c3, *week1_mobility*=c4, *week_passed*=63. The *Random* generator produces instances with feature

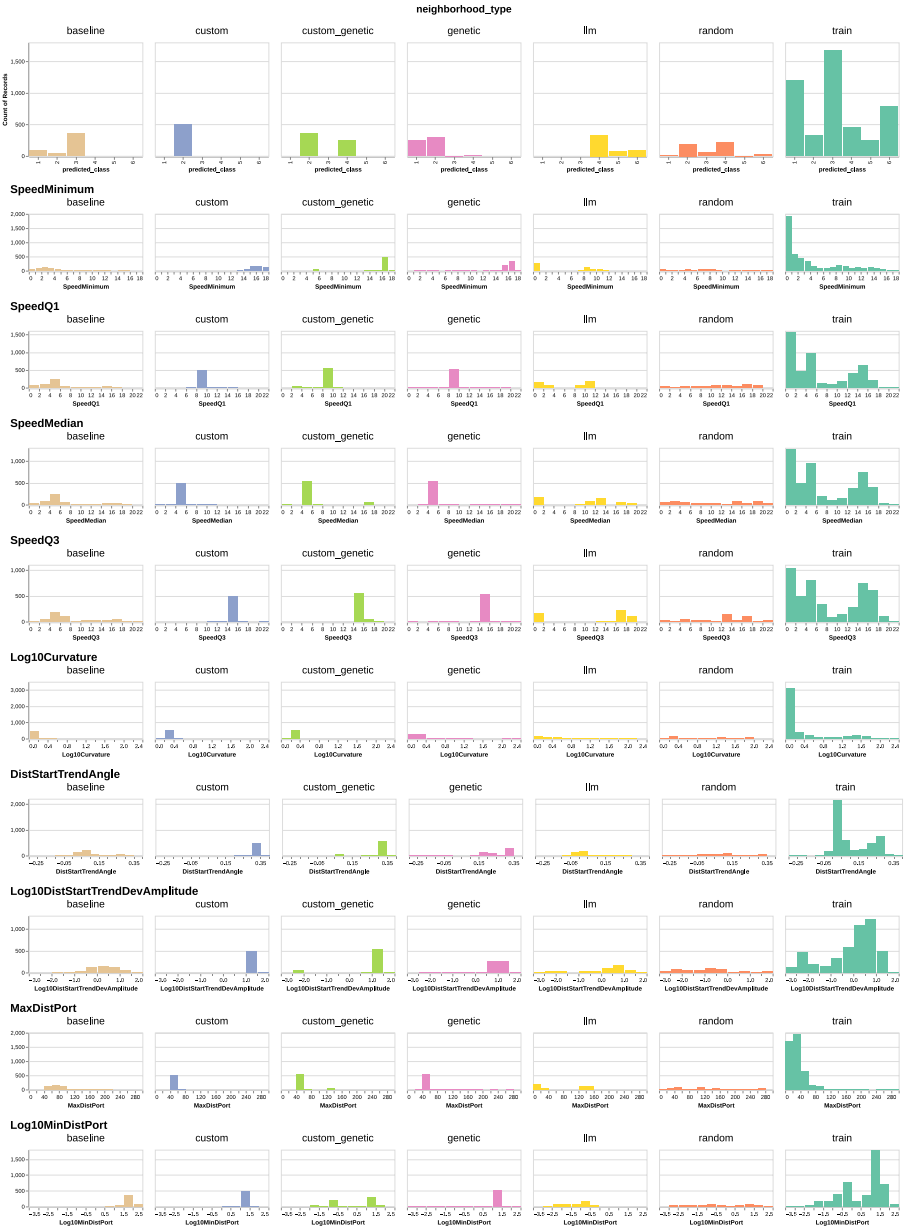

Fig. 8 A visual dashboard for the comparison and exploration of neighbourhoods generated around an instance x for the vessels dataset

values uniformly distributed across the entire feature space, resulting in a set of data distributions that do not align with the training data. In contrast, the *Custom* and *GPT* generators produce instances that are very close to each other, reflecting the constraints imposed by the domain knowledge. They are characterize by a narrow range of feature values. The two

Fig. 9 A visual dashboard for the comparison and exploration of neighbourhoods generated around an instance x for the COVID dataset

generators based on genetic optimization (*Genetic* and *Custom Genetic*) produce instances whose feature values are even more narrowly distributed. Given the size of the neighborhoods, the *Baseline* generator is limited in the number of instances available in the training data, thus producing a smaller number of instances.

In summary, the analytical comparison confirms that the domain-informed generators outperform standard methods in terms of locality and compactness. The measures of instance distance, Energy, and MMD indicate that the domain-informed generators produce neighbourhoods that are closer to the original instance and more aligned with the training data distribution. The qualitative comparison through visualizations further supports these findings, showing that the domain-informed generators produce instances that are more realistic and diverse, while also preserving the internal structure of the domain. The customized generators and those based on LLMs balance performance and output quality.

6.2 Evaluation of the rules

To assess the effectiveness of different synthetic neighbourhood generators for producing interpretable explanations, we extracted factual and counterfactual rules using several methods and displayed them side-by-side for expert evaluation. The rules were presented in a textual format to allow for direct comparison of their structure, complexity, and interpretability.

Figure 10 shows example rule sets for a specific instance from the vessel movement dataset (previously shown in Figure 8). The rule sets, each generated using a different method, reveal several insights:

- The *Baseline Training* generator, which samples from the training dataset, often produces verbose rules with many predicates. This is likely due to the inclusion of non-local instances in large neighbourhoods, causing the local surrogate to resemble the global model and weakening its local explanatory power.
- The *Custom* generator yields more concise rules by restricting perturbations based on expert-defined constraints. The surrogate model, influenced by a forest of decision trees, favours features that are semantically meaningful and better aligned with domain knowledge.
- The *Genetic Custom* generator behaves similarly to the Custom generator but introduces more diverse counterfactuals. The use of genetic optimization allows the search to explore a broader portion of the feature space while maintaining constraints.
- The *Genetic* generator, though unconstrained, occasionally produces compact rules resembling those from the Custom generator, but lacks domain grounding.

Figure 11 shows the rules extracted for the COVID-19 case study. As in the vessel example, the *Custom* and *GPT* generators produce the most concise and semantically focused rules. This conciseness results directly from the domain constraints embedded in the instance generation process. In contrast, the generic generators tend to yield rules with longer predicate lists and limited interpretability.

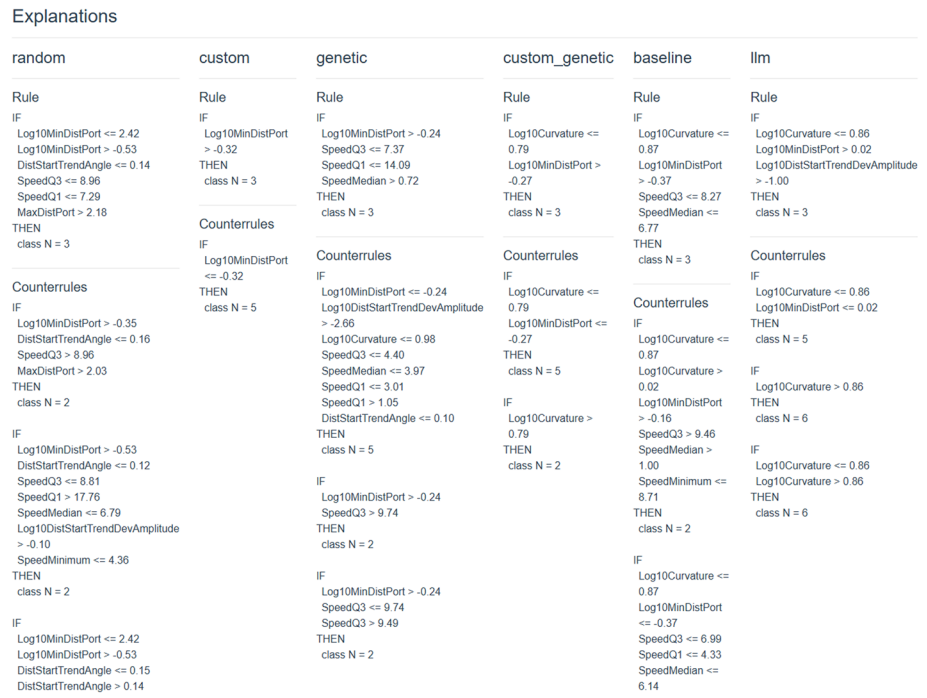

Fig. 10 Comparison of factual and counterfactual rules extracted from different generators for the Vessels model. The class labels correspond to: 1 - Straight, 2 - Curved, 3 - Trawling, 4 - Port-Connected, 5 - Near Port, 6 - Anchored

6.2.1 Limitations of current rule-based XAI methods

Despite these observations, our study ultimately reveals a fundamental limitation: the overall quality of the extracted rules across all generators is often inadequate for interpretability. For example, a rule such as: *"if week2_covid* $\neq$ c3 *and week6_covid* $\neq$ *c4 then class_label = c3"* fails to offer an exhaustive explanation of the model's behaviour involving all the relevant features. The logical link between the premises and conclusion may not be aligned with human reasoning about temporal dynamics or causality in epidemiological data.

This issue likely stems from how rule-based XAI methods like LORE operate. These methods assume independence among features and aim to identify minimal discriminative predicates to separate classes locally. As a result, they prioritise compactness over semantic coherence and are unable to leverage the complex interdependencies embedded in synthetic instances created by domain-informed generators.

As a further investigation, we computed the statistics of the decision trees used to extract the rules. Table 4 summarizes some of the key structural characteristics of the decision trees generated by each method when generating the explanations for the vessel movement dataset presented in Fig. 10. We can observe that the *Baseline* and *Random* generators produce more complex trees, with a larger depth and diameter. The *Genetic* generator also produces a tree with a complex structure. The two customized models (*Custom* and *LLM*) produce

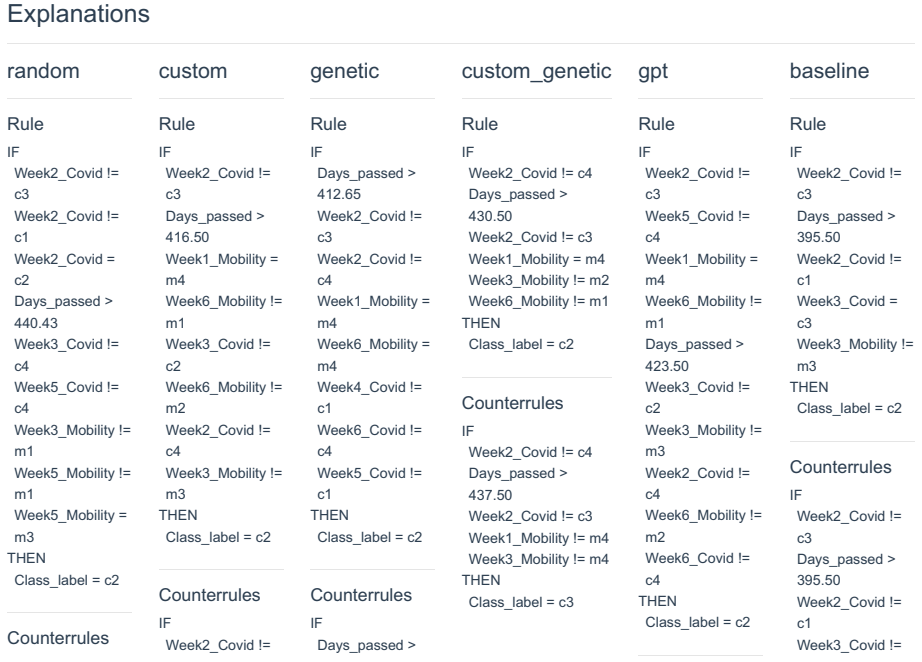

Fig. 11 Comparison between factual and counterfactual rules extracted from different generators for the COVID-19 model. The labels c1–c4 represent the scale from the lowest to the highest level of COVID-19 incidence

Table 4 Comparison of decision tree statistics across the geneators for the vessel movement dataset. Leaf depth is reported as mean ± standard deviation

Tree	# Leaves	Max depth	Leaf depth	Diameter	# Features used
Random	153	17	9.09 ± 2.99	34	9
Baseline	59	14	8.25 ± 2.72	19	9
Custom	3	2	1.67 ± 0.47	3	2
Genetic	39	11	6.79 ± 2.29	15	9
Custom genetic	6	4	3.00 ± 1.00	5	4
LLM	5	3	2.40 ± 0.49	5	3

more compact tree. This is reflected also in the complexity of the rules extracted for explanations. The *Custom_Genetic* generator combines the benefits of the *Custom* generator and the optimization of the *Genetic* generator.

Table 5 presents the statistics of the decision trees used to extract the rules for the COVID-19 case study presented in Fig. 11. In this case, we observe a similar pattern as in the previous case study. The *Baseline* and *Random* generators produce more complex trees, while the customized generators yield trees with a more compact structure. On one side, the compactness of the surrogate models is beneficial for interpretability, as it leads to simpler rules that may be easier to perceive. On the other side, the use of a small number of features may limit the ability of the surrogate model to capture the complexity of the domain relationships.

Table 5 Comparison of decision tree statistics across the generators for the COVID-19 and mobility dataset

Tree	#Leaves	Max Depth	Leaf Depth	Diameter	#Features Used
Random	136	15	8.54 ± 2.63	28	9
Baseline	60	13	8.20 ± 2.76	18	9
Custom	2	1	1.00 ± 0.00	2	1
Genetic	36	12	6.97 ± 2.80	16	8
Custom genetic	3	2	1.67 ± 0.47	3	2
GPT	5	3	2.40 ± 0.49	5	3

Leaf depth is reported as mean ± standard deviation

We conclude that current instance-based explanation methods are limited by their purely data-driven nature. While they may be effective for binary decisions in domains with low feature complexity (e.g., credit approval), they fall short when applied to temporally structured or semantically rich domains such as the ones studied here.

Rather than viewing this as a shortcoming of our approach, we frame it as an important finding: **embedding domain knowledge into synthetic data generation is a necessary but insufficient condition for producing human-aligned explanations**. The gap lies in the explanation algorithms themselves, which must evolve to reflect domain structure and inter-feature relationships more explicitly.

6.2.2 Toward LLM-enhanced explanation generation

Given the limitations of current rule-based XAI methods, we see promising opportunities for incorporating LLMs into the explanation process. Our experiments demonstrated that LLMs are capable of acquiring and operationalising domain knowledge through iterative interactions with human experts. This capability suggests that LLMs could serve not only as tools for generating synthetic data but also as intelligent agents for producing or refining explanations that are aligned with expert understanding.

A promising direction is the integration of LLMs with outputs from algorithmic XAI methods such as LORE or SHAP. Rather than replacing these methods, LLMs could interpret, validate, and enrich the explanations by combining model outputs with previously learned domain knowledge. For example, an LLM could take a rule or feature attribution generated by an XAI algorithm and augment it with additional context, constraints, or causal relationships derived from earlier conversations with a domain expert. This hybrid reasoning process could transform low-level, data-driven outputs into more coherent and cognitively aligned justifications.

In this way, LLMs offer the potential to bridge the gap between statistical patterns identified by models and the structured, semantically meaningful knowledge used by human decision-makers.

In light of our findings, we return to the central question posed in the introduction: how can domain knowledge be incorporated into both model development and explanation generation without modifying existing ML or XAI algorithms? Our work demonstrates that while knowledge-aware data preparation and neighbourhood generation significantly improve the realism and interpretability of inputs, current XAI methods often fall short in extracting meaningful, human-aligned explanations. This gap suggests that LLMs, given

their ability to absorb and reuse domain knowledge acquired through expert interaction, may serve as the critical missing layer.

7 Conclusion and future work

In this work, we proposed a methodology for integrating human knowledge into machine learning (ML) workflows to enhance both model interpretability and performance. By incorporating expert insight through visual analytics and domain-driven data transformations, we enabled more semantically meaningful representations that align with expert reasoning. Our dual approach targets two critical stages: the construction of interpretable models through thoughtful data preparation and feature engineering, and the generation of explanation-supporting synthetic neighborhoods that adhere to domain constraints.

We validated our approach in two real-world case studies - predicting COVID-19 dynamics and identifying vessel movement patterns, demonstrating its applicability to different types of data. Through these case studies, we showed that domain-aware neighbourhood generation yields synthetic instances that better reflect the structure of the real data and expert expectations. Using quantitative and qualitative evaluation, we confirmed that knowledge-guided instance generation produces neighbourhoods with improved locality, realism, and diversity.

However, our findings also reveal a critical limitation of current algorithmic explanation techniques such as LORE. Despite improvements in neighbourhood quality, the generated explanations often remain logically unconvincing or inconsistent with human reasoning. This exposes a gap between domain-aware data generation and purely data-driven explanation methods. We argue that existing XAI frameworks lack the capacity to represent feature interdependencies or temporal semantics required for meaningful explanations in complex domains.

This limitation points to a new research direction: the integration of large language models (LLMs) not only in data generation but also in explanation generation. Our experiments show that LLMs can understand and operationalise domain constraints; thus, they could be further leveraged to refine or augment explanations produced by standard XAI methods. By enabling semantically rich, context-aware justifications, LLMs may help bridge the gap between model logic and human mental models.

In future work, we aim to extend this methodology in several directions. First, scaling the approach to higher-dimensional feature spaces while preserving interpretability is essential for broader applicability. Second, exploring generative modelling techniques, such as GANs or reinforcement learning, could further enhance the diversity and fidelity of synthetic neighbourhoods. Third, longitudinal user studies could provide deeper insights into how domain experts interact with and benefit from knowledge-driven explanations. Finally, hybrid systems integrating LLMs into the explanation pipeline open possibilities for rule synthesis, feature relevance analysis, and interactive refinement of models.

By addressing these directions, we aim to further strengthen the integration of human knowledge into AI systems, advancing not only their accuracy and robustness but also their transparency and alignment with human understanding.

Appendix A: Using an LLM (ChatGPT) for synthetic neighbourhood generation in the COVID-19 and mobility case study

We performed this experiment using ChatGPT-4 Turbo (model GPT-4o) and focusing on COVID-19 and mobility applications. We aimed to explore a scenario where a domain expert communicates with an LLM, transferring relevant domain-specific knowledge. The LLM creates and refines the code of a neighbourhood generator, demonstrates examples of its outcomes, and incorporates expert feedback iteratively. This experiment sought to answer the following questions: How realistic is this scenario? What challenges must be addressed to make it effective?

Prompting the LLM

Initial prompt (omitting the data examples)

We developed a machine learning model to predict future levels of pandemic incidence (COVID-19) based on historical disease and mobility data. The dataset used for modelling covered daily counts of new COVID-19 cases and trips within 52 geographic regions, normalized by population size. The data were aggregated into weekly intervals spanning 64 weeks from February 2020 to May 2021, excluding the first six weeks due to significant differences in pandemic patterns during the initial outbreak phase.

The aggregated data were discretized into four levels for disease incidence and four levels for mobility, where lower levels of mobility indicated more severe restrictions, whereas higher levels of disease incidence indicated higher disease spread. For each prediction, we used a temporal context consisting of disease levels from weeks -6 to -2 and mobility levels from weeks -6 to -1 relative to the target event, along with a temporal variable representing the number of days since the pandemic began.

The modelling task focused on capturing the interdependencies between disease incidence and mobility levels. Increases in disease incidence typically led to reduced mobility through restrictions, which subsequently contributed to a delayed decline in disease levels. Conversely, relaxed mobility restrictions often resulted in increased disease incidence after a delay of one to three weeks.

To test the model's robustness and explore its behavior under varying conditions, we want to generate synthetic data instances in the neighbourhood of a given real data instance. These synthetic instances should be created by perturbing the original feature values while adhering to observed transition probabilities for both disease and mobility levels. The generated data should reflect realistic temporal dynamics and interdependencies.

The goal of the synthetic data generation is to investigate how variations in input features influenced the model's predictions for the next level of pandemic incidence. By generating a sufficiently diverse neighbourhood, we aim to ensure that the model produces predictions across all possible levels of pandemic incidence, thereby enhancing the interpretability and reliability of the predictions.

Refinement

After assessing the first result, the LLM was given the following request: "*You need to introduce some randomness in the generation, but keep consistency between the values of the features according to the transition frequencies. The neighbourhood should be sufficiently diverse to obtain at least 15 instances of each class after applying the model*".

Domain Knowledge Involved in the Synthetic Neighbourhood Generator

Below is the LLM-generated description of the domain knowledge incorporated in the neighbourhood generator. It demonstrates the degree of the LLM's "understanding" of the domain.

The synthetic neighbourhood generator incorporated the following domain knowledge to ensure the generation of realistic and meaningful data instances:

- Temporal Dynamics: The relationships between time-dependent variables, such as COVID-19 levels and mobility levels, follow observable transition probabilities. These transitions reflect real-world temporal changes, e.g., how mobility levels shift over weeks or how COVID-19 levels evolve. Transition frequencies for each variable were constructed based on historical data, capturing the likelihood of transitions between specific states.
- Feature Interdependencies: Temporal features such as "Weeks_passed" and "Days_passed" are interrelated, with consistent calculations ensuring the generated instances remain plausible (e.g., "Days_passed" = "Weeks_passed" $\times$ 7).
- Realism in Variations: Variations in feature values (e.g., changes in past-week COVID-19 and mobility levels) must align with the underlying transition frequencies. This ensures that generated synthetic instances resemble realistic scenarios rather than random deviations.
- Diversity and Coverage: Generating synthetic neighbours involves balancing proximity to the original instance with sufficient diversity to explore less common but plausible scenarios, thereby enriching the dataset and improving class coverage.
- Categorical-to-Numerical Mapping: To facilitate downstream processing, categorical values for COVID-19 and mobility levels are systematically mapped to numerical representations while maintaining interpretability.This domain knowledge was iteratively incorporated into the code, ensuring the synthetic neighbours adhered to realistic transitions, captured temporal and spatial patterns, and reflected meaningful diversity.

Assessment of the results

A Python script was developed to take multiple input instances from a specified file and generate a given number of synthetic neighbours for each instance. The results are stored in two output files: one containing categorical feature values and the other with their corresponding numeric representations. Below, we detail the process of testing the code and evaluating its performance.

Using a dataset of real data, 15 input instances were randomly selected, each representing a unique combination of conditions. For each instance, 100 synthetic neighbours were generated. Both original and synthetic instances, represented numerically, were processed through a predictive model that assigned class labels reflecting different COVID-19 levels. To evaluate the effectiveness of the synthetic neighbourhood generator, we created a visualization in the form of a 2D map, where points are positioned based on their similarity in the high-dimensional feature space. To derive the 2D positions, we applied the UMAP algorithm, which preserves both local and global data structures.

The resulting visualization is shown in Fig. 12. Larger circles represent the original instances, while smaller rectangles correspond to the synthetic neighbours. The symbols

are colour-coded according to their predicted classes, allowing for an overall assessment of class distributions. Original instances are further marked with black hollow circles for better identification.

For more detailed analysis, filters can be applied to the set of the displayed instances to focus on different subsets. Fig. 13 shows the neighbours of the original instances grouped by classes `c1`, `c2`, `c3`, and `c4`, consisting of 2, 3, 5, and 5 original instances, respectively. Similarly, Fig. 14 illustrates the distributions of synthetic neighbours for four selected original instances from different classes. These visualizations enable domain experts to assess both the alignment of synthetic neighbours with their respective original instances and the diversity of their predicted classes.

The visualizations reveal that the synthetic neighbours generally cluster around their respective original instances, as expected. However, scattered neighbours are also present, contributing to the desired diversity. Importantly, the generator produces instances diverse enough to be classified into different categories by the model. While obtaining model predictions is not a part of the synthetic neighbourhood generation code, this post-generation step effectively demonstrates the utility of the synthetic data for testing and model evaluation.

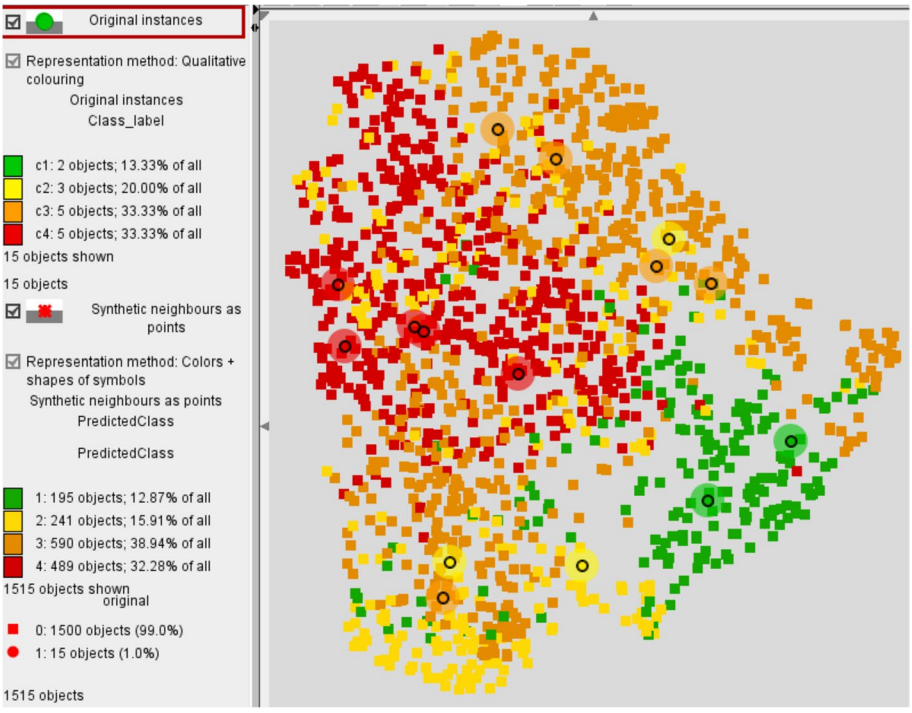

Fig. 12 Original and synthetic instances obtained with the help of ChatGPT for the COVID-19 and mobility case study are represented as points on a map. The points are placed according to instance similarity. The colours correspond to the predicted classes. The square symbols represent the synthetic instances. Larger circles represent the 15 original instances for which the synthetic neighbours were generated. These instances are additionally marked by black hollow circles

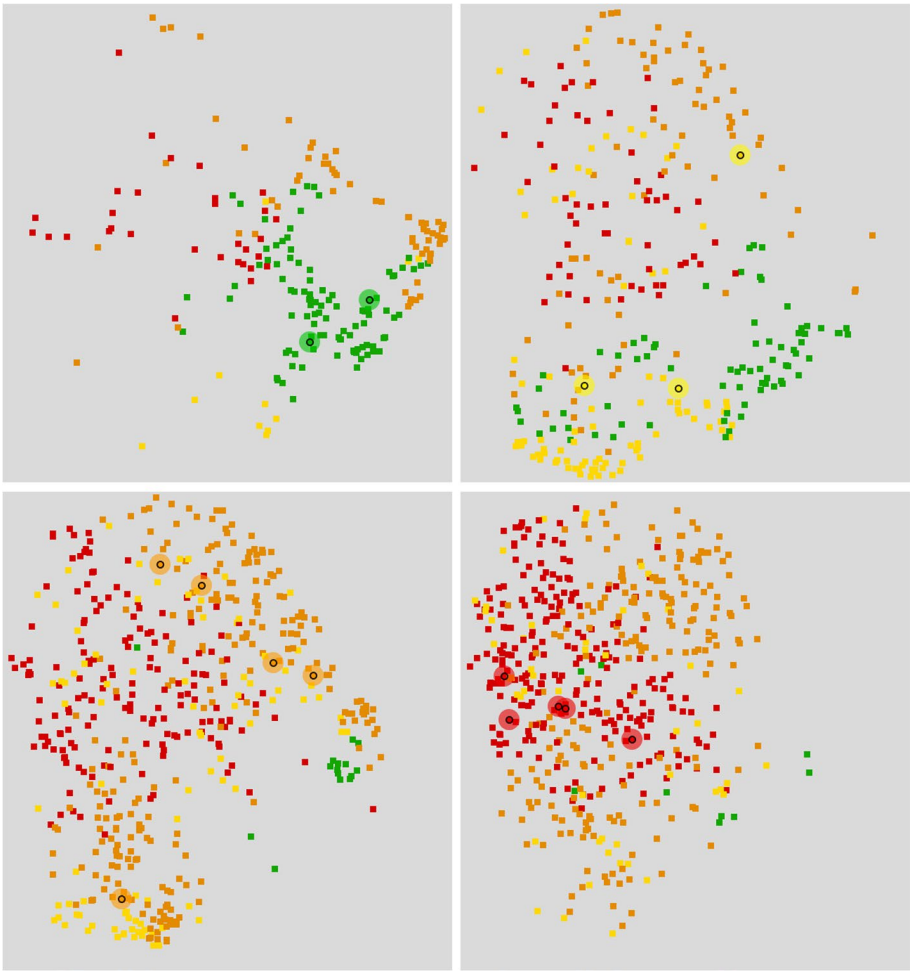

Fig. 13 Distributions of synthetic instances generated for the original instances with the actual classes `c1` (2 instances), `c2` (3 instances), `c3` (5 instances), and `c4` (5 instances)

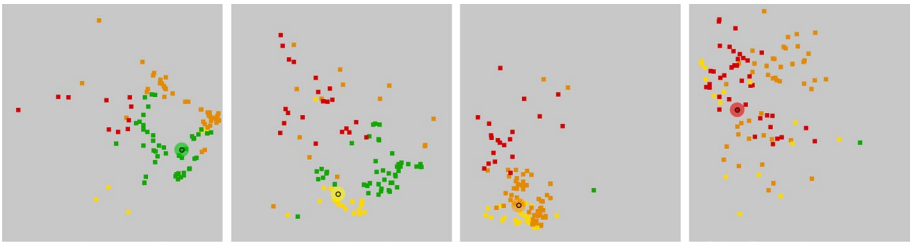

Fig. 14 Distributions of synthetic instances generated for 4 selected original instances

Conclusion

This experiment demonstrated the feasibility of using an LLM to generate synthetic neighbourhoods that balance randomness with domain-consistent variations. The approach can potentially be adapted to other domains with temporal or sequential data. However, several challenges were identified:

- Complexity in Translating Domain Knowledge: LLMs require clear, well-thought-out inputs to effectively encode complex domain rules. Informal descriptions may need reformulation by experts for accurate implementation.
- Validation and Debugging: The LLM-generated code can occasionally deviate from the intended logic or contain errors, requiring careful validation and iterative refinements. These tasks require coding expertise, which domain experts may not possess. Hence, there is a need for tools that bridge this skill gap.
- Visualization as a Feedback Tool: Visualization was instrumental in assessing the distribution and diversity of synthetic instances. However, additional visualizations are needed to evaluate the realism of the generated feature value combinations.Overall, while using LLMs for synthetic data generation shows significant promise, achieving seamless collaboration with domain experts will require advances in visualization, validation, and communication tools.

Appendix B: Using an LLM (Google Gemini) for synthetic neighbourhood generation in the vessel movement case study

For the vessel movement case study, we made an experiment with Google Gemini version 2.5 Flash.

Prompting the LLM

The following prompt was submitted to the LLM: I need to create a Python program for generation of synthetic data instances in the neighbourhood of a given instance. The purpose is to generate local explanations of predictions produced by a model that recognises patterns of vessel movements based on 9 features. Input data for the model consist of values of numeric attributes characterizing segments of vessel trajectories. The model utilizes nine features describing vessel speed characteristics, trajectory shape, and proximity to ports. Logarithmic transformation was applied to features with highly skewed value distributions in the training data.

- Speed statistics: `SpeedMinimum`, `SpeedQ1`, `SpeedMedian`, `SpeedQ3`. These features represent the speed distribution, from minimum observed speed to quartile-based spread.
- `Log10Curvature`: The logarithm of the curvature of the time series of the vessel's distance from the starting point. The curvature is computed as the ratio between the sum of absolute consecutive changes in the time series and the amplitude of values. A value close to 1 indicates nearly straight movement, while higher values indicate the presence

of turns.

- `DistStartTrendAngle`: The angle of the linear trend fitted to the time series of the vessel's distance from its starting point. A higher angle indicates a stronger trend of moving away, while a lower angle suggests slower movement or a return toward the starting position.
- `Log10DistStartTrendDevAmplitude`: The logarithm of the amplitude of deviations from the trend line of the distance to the starting point. It quantifies path tortuosity, reflecting how much the vessel's movement deviates from a straight-line trend. Higher values indicate more erratic or zigzagging movement.
- Port proximity: `MaxDistPort`, `Log10MinDistPort`. These features represent the maximum and (log-transformed) minimum distance from the nearest port, which are useful for recognizing anchoring and port manoeuvring.`Log10Curvature` and `Log10DistStartTrendDevAmplitude` jointly describe the shape complexity of a trajectory segment. `Log10Curvature` captures global movement characteristics, while path tortuosity reflects local zigzagging. Together, they help distinguish between steady outward movement, looping, and in-place manoeuvring.

The features are used to classify movement behaviours in trajectory segments (episodes) to one of the six classes: 1) straight movement; 2) curved movement; 3) trawling; 4) port-connected (entering or exiting a port); 5) near port (manoeuvring in port vicinity); 6) anchored. Each data instance describes one episode by the 9 features listed above.

The Python program needs, for a given instance with a true or predicted class label, perturb the feature values and thereby create a specified number of synthetic instances that are likely to be classified as belonging to each of the 6 classes, including the class of the given instance. It is desirable to keep balance between the numbers of instances of each class, although it is not known in advance how the model will classify them. It is desirable to achieve the class variety by making small changes of the feature values.

The synthetic instances must be realistic, i.e., satisfying certain constraints:

- Numeric relationships between the speed values: `SpeedMinimum` $\leq$ `SpeedQ1` $\leq$ `SpeedMedian` $\leq$ `SpeedQ3`
- Realistic `SpeedQ3`: should be not much higher that the maximum occurring in the training data, which is 22.
- High speed cannot be achieved during curvy movements.
- The range of `DistStartTrendAngle` in the training data was $[-0.24, 0.36]$; the values in synthetic data should not be far beyond.
- The same applies to `Log10Curvature` [0, 2.25] and `Log10DistStartTrendDevAmplitude` $[-2.8, 1.8]$.
- The smallest value of `Log10MinDistPort` was -3.05.Here are some examples of instances of different classes: ...

Input instances for the Python program look like these but do not include the field "Class" with string values.

LLM's response

I will create a Python program that generates synthetic data instances based on a given vessel movement instance. This program will perturb the feature values of the input instance, aiming to create new instances that are likely to represent different vessel behaviors, while adhering to realistic constraints.

The program will account for:

- Nine key features: `SpeedMinimum`, `SpeedQ1`, `SpeedMedian`, `SpeedQ3`, `Log10Curvature`, `DistStartTrendAngle`, `Log10DistStartTrendDevAmplitude`, `MaxDistPort`, and `Log10MinDistPort`.
- Realistic constraints: Including relationships between speed quartiles, maximum speed limits, the interplay between speed and curvature, and specific ranges for angular and curvature-related features, as well as port proximity.
- Targeted perturbations: Although we don't have the actual classification model, the program will apply perturbations strategically to generate instances that align with the characteristics of each of the six movement classes (straight, curved, trawling, port-connected, near port, anchored). This approach increases the likelihood of creating diverse synthetic data that could be classified into different categories by a real model.The output will be a list of dictionaries, where each dictionary represents a synthetic data instance with its 9 feature values.

Here is the Python program: ...

The code was sufficiently good right from the beginning; only very minor changes in data input and output were further requested.

Result

The LLM described the resulting program as follows: The provided Python program is designed for the generation of synthetic data instances, primarily to facilitate local explanations of predictions made by a vessel movement pattern recognition model. This model operates on nine numerical features characterizing segments of vessel trajectories. The synthetic data generation process ensures that the newly created instances are both diverse and realistic, adhering to specific physical and contextual constraints.

The nine features utilized by the model and, consequently, by this generator are: `SpeedMinimum`, `SpeedQ1`, `SpeedMedian`, `SpeedQ3` (describing speed distribution); `Log10Curvature` (logarithm of trajectory curvature); `DistStartTrendAngle` (angle of distance trend from start); `Log10DistStartTrendDevAmplitude` (logarithm of deviation amplitude from trend); `MaxDistPort`, and `Log10MinDistPort` (maximum and log-transformed minimum distance from the nearest port). Features like `Log10Curvature` and `Log10DistStartTrendDevAmplitude` are critical for capturing trajectory shape complexity, distinguishing between global movement and local tortuosity.

The core of the synthetic data generation lies within the `SyntheticDataGenerator` class. For a given input instance, the program generates a specified number of synthetic neighbors. This process involves perturbing the feature values of the original instance using a controllable `perturbation_scale`. A key aspect is the strategic application of these

perturbations to create instances that tend towards one of six predefined vessel movement classes: straight movement, curved movement, trawling, port-connected, near port, and anchored. This class-tendency-driven perturbation aims to achieve a balanced variety of synthetic instances across all classes, even without direct access to the classification model.

Crucially, the program enforces a comprehensive set of realistic constraints to ensure the generated data remains plausible:

- Speed Quartile Ordering: $\mathrm{SpeedMinimum} \leq \mathrm{SpeedQ1} \leq \mathrm{SpeedMedian} \leq \mathrm{SpeedQ3}$ is strictly maintained.
- Maximum Speed: `SpeedQ3` is capped at 22.0, reflecting observed training data limits.
- Speed-Curvature Interaction: High speeds are made incompatible with high curvature values; if both occur, speed is reduced or curvature adjusted.
- Feature Ranges: `DistStartTrendAngle`, `Log10Curvature`, and `Log10DistStartTrendDevAmplitude` are clamped within their empirically derived training data ranges (e.g., `Log10Curvature` between 0.0 and 2.25).
- Port Proximity Minimum: `Log10MinDistPort` is constrained to be no less than -3.05.
- Non-negativity: All speed values and `MaxDistPort` are ensured to be non-negative.These constraints are applied after initial perturbations to correct any unrealistic values, thus enhancing the fidelity of the synthetic dataset.

The workflow integrates seamlessly with CSV file operations. The `generate_and_save_synthetic_data_from_csv` function facilitates batch processing: it reads original, labeled instances from an input CSV file, systematically generates N synthetic neighbors for each original instance (where N is configurable), and then compiles all generated instances into a single dataset. This consolidated synthetic dataset, devoid of class labels, is then written to a new output CSV file, ready for subsequent analysis or use in model interpretability frameworks. This entire process provides a robust method for expanding datasets with realistic variations, particularly valuable for exploring model behavior in the vicinity of specific data points.

Evaluation of the program performance

To evaluate the performance of the LLM-generated synthetic data generator, we selected 120 example instances from the training data, i.e., 20 instances per class across the six behaviour categories. For each original instance, the generator produced 100 synthetic neighbours, resulting in a total of 12,000 synthetic instances. These were passed through the trained pattern classification model to obtain predicted class labels. The resulting distribution of predicted classes was as follows: class 1 – 712 instances, class 2 – 1434, class 3 – 1776, class 4 – 2016, class 5 – 3756, and class 6 – 2306.

The labelled synthetic instances were analysed using visual analytics tools. Unlike the COVID-19 case study, where feature values were categorical, this case involved numerical features, necessitating different visualization techniques. Specifically, we used parallel coordinates plots (PCPs), where each axis represents a feature and each line corresponds to a data instance, connecting its values across all features.

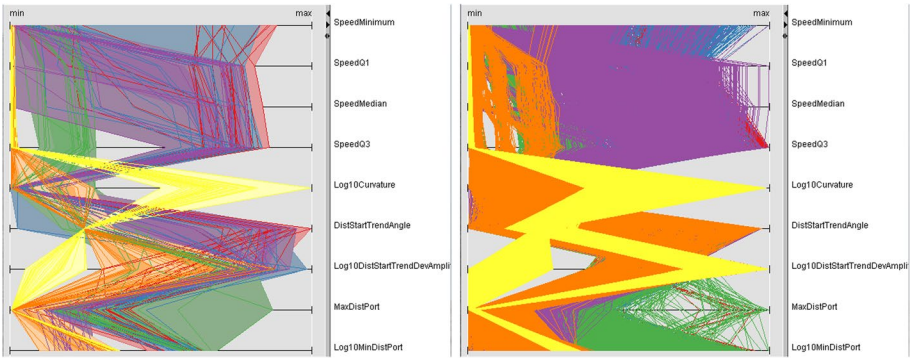

Fig. 15 Representation of instances on a parallel coordinates plot with parallel coordinate axes representing the features. Left: Real instances. Right: Synthetic neighbours of the real instances

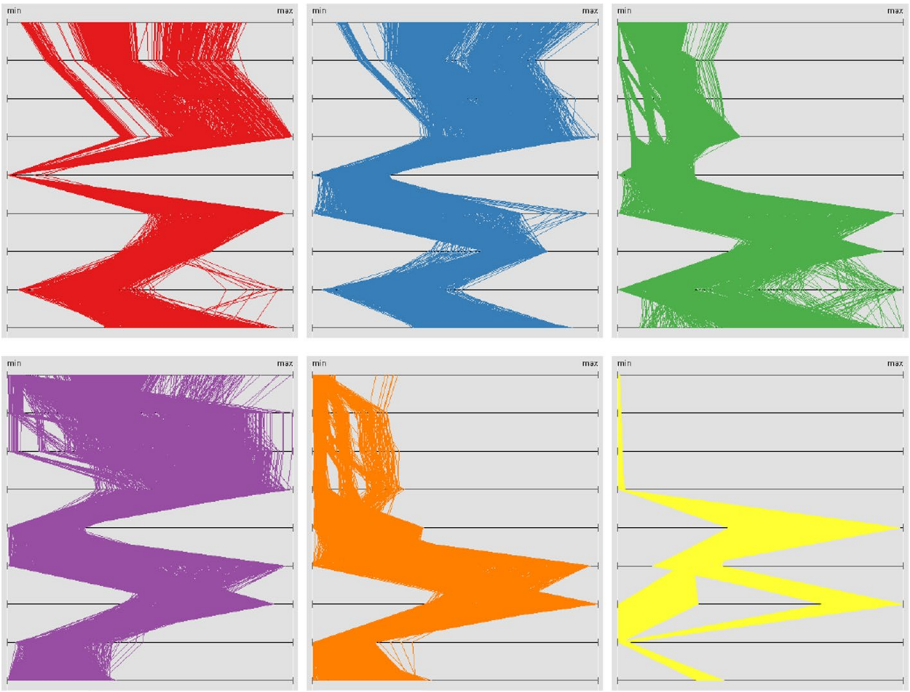

Fig. 16 Screenshots of the PCP display show the distributions of the feature values separately for each class: 1) red - straight movement; 2) blue - curved movement; 3) green - trawling; 4) purple - port-connected; 5) orange - near port; 6) yellow - anchored

Figure 15 shows the comparison between the original and synthetic data. The left panel presents the 120 real instances, while the right panel displays the 12,000 synthetic neighbours generated by the LLM-based program. While the plots reveal the overall feature space coverage, the high density of overplotted lines makes it difficult to distinguish patterns among individual classes.

To better explore class-specific characteristics, we applied interactive filtering to isolate and visualize subsets of synthetic instances by their predicted classes. Figure 16 shows the distributions of feature values for each predicted class separately. This view allowed us to compare and assess whether the synthetic instances reflect the expected behaviour patterns of each class.

The visualizations confirm two important qualities of the generated synthetic neighbourhoods. First, the synthetic instances remain close to their original sources in the feature space, demonstrating good locality. Second, they exhibit sufficient diversity: feature values vary across meaningful ranges while respecting domain constraints. This suggests that the generator successfully balances locality and variation.

Overall, the experiment demonstrates that the LLM-generated code can produce high-quality, domain-consistent synthetic data. The resulting neighbourhoods support model interpretation and evaluation by providing diverse yet plausible variations of real instances.

Author contributions E.C. and S.R. prepared Figures 1, 5–8, A1, B2, and B3, and contributed to writing Section 3.2. B.K., G.A., and N.A. prepared Figures 2–4, C4, D5–7, and E8–9, and contributed to writing Section 3.1. All authors contributed to the development of the remaining sections of the manuscript, as well as to the writing and critical review of the entire work.

Funding Open access funding provided by Consiglio Nazionale Delle Ricerche (CNR) within the CRUI-CARE Agreement. This work was supported by Federal Ministry of Education and Research of Germany and the state of North-Rhine Westphalia as part of the Lamarr Institute for Machine Learning and Artificial Intelligence (Lamarr22B), by EU in projects SoBigData++ (grant agreement no. 871042) and CREXDATA (grant agreement no. 101092749), NextGenerationEU programme under the funding schemes PNRR-PE-AI scheme (M4C2, investment 1.3, line on AI) FAIR (Future Artificial Intelligence Research), and TANGO project, grant agreement no. 101120763.

Data availability Data link is provided within the manuscript. Both datasets used are open access.

Declarations

Conflict of interest The authors declare no competing interests.

References

Abdul, A. M., Vermeulen, J., Wang, D., Lim, B. Y., & Kankanhalli, M. S. (2018). Trends and trajectories for explainable, accountable and intelligible systems: An HCI research agenda. *ACM*. https://doi.org/10.1145/3173574.3174156

Adadi, A., & Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE Access*. https://doi.org/10.1109/ACCESS.2018.2870052

Andrienko, N., Andrienko, G., Artikis, A., Mantenoglou, P., & Rinzivillo, S. (2024). Human-in-the-loop: Visual analytics for building models recognising behavioural patterns in time series. *IEEE Computer Graphics and Applications*. https://doi.org/10.1109/MCG.2024.3379851

Andrienko, N., Andrienko, G., & Rinzivillo, S. (2016). Leveraging spatial abstraction in traffic analysis and forecasting with visual analytics. *Information Systems, 57*, 172–194. https://doi.org/10.1016/j.is.2015.08.007

Andrienko, N., Andrienko, G., & Shirato, G. (2023). *Episodes and topics in multivariate temporal data,42*(6), Article e14926. https://doi.org/10.1111/cgf.14926

Bhattacharya, A., Verbert, K., et al. (2024). Exmos: Explanatory model steering through multifaceted explanations and data configurations. In: Proceedings of the CHI Conference on Human Factors in Computing Systems. ACM. https://doi.org/10.1145/3544548.3581135

Buchanan, B.G., Davis, R., & Feigenbaum, E.A. (2006). Expert systems: A perspective from computer science. The Cambridge handbook of expertise and expert performance pp. 87–103.

Cheng, F., Ming, Y., & Qu, H. (2021). DECE: decision explorer with counterfactual explanations for machine learning models. *IEEE*. https://doi.org/10.1109/TVCG.2020.3030342

Dong, G., & Liu, H. (2018). Feature engineering for machine learning and data analytics. *CRC Press*. https://doi.org/10.1201/9781315181080

Endert, A., Ribarsky, W., Turkay, C., Wong, B. W., Nabney, I., Blanco, I. D., & Rossi, F. (2017). The state of the art in integrating machine learning into visual analytics. *Computer Graphics Forum*. https://doi.org/10.1111/cgf.13092

Giabbanelli, P. J., & Jackson, P. J. (2015). Using visual analytics to support the integration of expert knowledge in the design of medical models and simulations. *Procedia Computer Science*. https://doi.org/10.1016/j.procs.2015.05.195

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., & Smola, A. (2012). A kernel two-sample test. *Journal of Machine Learning Research, 13*(25), 723–773.

Guidotti, R., Monreale, A., Giannotti, F., Pedreschi, D., Ruggieri, S., & Turini, F. (2019). Factual and counterfactual explanations for black box decision making. *IEEE Intelligent System* https://doi.org/10.1109/MIS.2019.2957223

Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., & Pedreschi, D. (2019). A survey of methods for explaining black box models. *ACM Computer Survery* https://doi.org/10.1145/3236009

Hohman, F., Kahng, M., Pienta, R. S., & Chau, D. H. (2019). Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE Transactions on Visualization and Computer Graphics* https://doi.org/10.1109/TVCG.2018.2843369

Karpatne, A., Atluri, G., Faghmous, J. H., Steinbach, M. S., Banerjee, A., Ganguly, A. R., Shekhar, S., Samatova, N. F., & Kumar, V. (2017). Theory-guided data science: A new paradigm for scientific discovery from data. *IEEE Transactions on Knowledge and Data Engineering* https://doi.org/10.1109/TKDE.2017.2720168

Kidd, A.L. (ed.). (1987). Knowledge Acquisition for Expert Systems: A Practical Handbook. Springer, New York, NY. https://doi.org/10.1007/978-1-4613-1823-1

Kulesza, T., Stumpf, S., Burnett, M. M., Yang, S., Kwan, I., & Wong, W. (2013). Too much, too little, or just right? ways explanations impact end users' mental models. *IEEE Computer Society*. https://doi.org/10.1109/VLHCC.2013.6645235

Ponce-de Leon, M., del Valle, J., Fernandez, J. M., Bernardo, M., Cirillo, D., Sanchez-Valle, J., Smith, M., Capella-Gutierrez, S., Gullón, T., & Valencia, A. (2021). COVID-19 Flow-Maps an open geographic information system on COVID-19 and human mobility for Spain. *Scientific Data*. https://doi.org/10.1038/s41597-021-01093-5

Ponce-de Leon, M., del Valle, J., Fernández, J.M., Bernardo, M., Cirillo, D., Sanchez-Valle, J., Smith, M., Capella-Gutierrez, S., Gullón, T., & Valencia, A. (2021). COVID19 Flow-Maps daily cases reports.https://doi.org/10.5281/zenodo.5217386

Ponce-de Leon, M., del Valle, J., Fernández, J.M., Bernardo, M., Cirillo, D., Sanchez-Valle, J., Smith, M., Capella-Gutierrez, S., Gullón, T., & Valencia, A. (2021). COVID19 Flow-Maps daily-mobility for Spain.https://doi.org/10.5281/zenodo.5539411

Ponce-de Leon, M., del Valle, J., Fernández, J.M., Bernardo, M., Cirillo, D., Sanchez-Valle, J., Smith, M., Capella-Gutierrez, S., Gullón, T., & Valencia, A. (2021). COVID19 Flow-Maps population data. https://doi.org/10.5281/zenodo.5226351

Liao, Q.V., & Varshney, K.R. (2021). Human-centered explainable AI (XAI): from algorithms to user experiences. CoRR arxiv:2110.10790

Lu, Y., Garcia, R., Hansen, B., Gleicher, M., & Maciejewski, R. (2017). The state-of-the-art in predictive visual analytics. *Computer Graphics Forum*. https://doi.org/10.1111/cgf.13210

Lundberg, S. M., & Lee, S. I. (2017). *A unified approach to interpreting model predictions.* Red Hook, NY, USA: NIPS'17, Curran Associates Inc.

Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artificail Intelligence* https://doi.org/10.1016/J.ARTINT.2018.07.007

Ming, Y., Qu, H., & Bertini, E. (2019). Rulematrix: Visualizing and understanding classifiers with rules. *IEEE Transactions on Visualization and Computer Graphics,* https://doi.org/10.1109/TVCG.2018.2864812

Muralidhar, N., Islam, M.R., Marwah, M., Karpatne, A., & Ramakrishnan, N. (2018). Incorporating prior domain knowledge into deep neural networks. In: *2018 IEEE International Conference on Big Data (Big Data)*. pp. 36–45. https://doi.org/10.1109/BigData.2018.8621955

Peng, L., Lin, Z., Andrienko, N., Andrienko, G., & Chen, S. (2025). Contextualized visual analytics for multivariate events. *Visual Informatics, 9*(2), Article 100234. https://doi.org/10.1016/j.visinf.2025.100234

Raissi, M., Perdikaris, P., & Karniadakis, G. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*. https://doi.org/10.1016/j.jcp.2018.10.045

Rajabi, E., & Etminani, K. (2024). Knowledge-graph-based explainable AI: A systematic review. *Journal of Information Science* https://doi.org/10.1177/01655515221112844

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *ACM*. https://doi.org/10.1145/2939672.2939778

Ribeiro, M.T., Singh, S., & Guestrin, C. (2018). Anchors: High-precision model-agnostic explanations. In: McIlraith, S.A., Weinberger, K.Q. (eds.) *Proc. of the Thirty-Second AAAI Conference on Artificial Intelligence*. AAAI Press. https://doi.org/10.1609/AAAI.V32I1.11491

Riveiro, M., & Thill, S. (2021). "that's (not) the output I expected!" on the role of end user expectations in creating explanations of AI systems. *Artificail Intelligence*. https://doi.org/10.1016/J.ARTINT.2021.103507

von Rueden, L., Mayer, S., Beckh, K., Georgiev, B., Giesselbach, S., Heese, R., Kirsch, B., Pfrommer, J., Pick, A., Ramamurthy, R., Walczak, M., Garcke, J., Bauckhage, C., & Schuecker, J. (2023). Informed machine learning – a taxonomy and survey of integrating prior knowledge into learning systems. *IEEE Transactions on Knowledge and Data Engineering*. https://doi.org/10.1109/TKDE.2021.3079836

Sacha, D., Kraus, M., Keim, D. A., & Chen, M. (2019). Vis4ml: An ontology for visual analytics assisted machine learning. *IEEE Transactions on Visualization and Computer Graphics*. https://doi.org/10.1109/TVCG.2018.2864838

Schramowski, P., Stammer, W., Teso, S., Brugger, A., Herbert, F., Shao, X., Luigs, L., & Kersting, K. (2020). Making deep neural networks right for the right scientific reasons by interacting with their explanations. *Nature Machine Intelligence, 2*(8), 476–486. https://doi.org/10.1038/s42256-020-0212-3

Simkute, A., Surana, A., Luger, E., Evans, M., & Jones, R. (2022). XAI for learning: Narrowing down the digital divide between "new" and "old" experts. *ACM*. https://doi.org/10.1145/3547522.3547678

Spinner, T., Schlegel, U., Schäfer, H., & El-Assady, M. (2020). explAIner: A visual analytics framework for interactive and explainable machine learning. *IEEE Transactions on Visualization and Computer Graphics*. https://doi.org/10.1109/TVCG.2019.2934629

Székely, G. J., & Rizzo, M. L. (2013). Energy statistics: A class of statistics based on distances. *Journal of Statistical Planning and Inference, 143*(8), 1249–1272. https://doi.org/10.1016/j.jspi.2013.03.018https://www.sciencedirect.com/science/article/pii/S0378375813000633.

Teso, S., & Kersting, K. (2019). Explanatory interactive machine learning. In: Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society. pp. 239–245. ACM. https://doi.org/10.1145/3306618.3314293

Wang, J., Liu, S., & Zhang, W. (2024). Visual analytics for machine learning: A data perspective survey. *IEEE Transactions on Visualization and Computer Graphics*. https://doi.org/10.1109/TVCG.2024.3357065

Zhao, J., Glueck, M., Isenberg, P., Chevalier, F., & Khan, A. (2018). Supporting handoff in asynchronous collaborative sensemaking using knowledge-transfer graphs. *IEEE Transactions on Visualization and Computer Graphics*. https://doi.org/10.1109/TVCG.2017.2745279

Authors and Affiliations

Eleonora Cappuccio[1] · Bahavathy Kathirgamanathan[2] · Salvatore Rinzivillo[1] · Gennady Andrienko[2,3] · Natalia Andrienko[2,3]

✉ Eleonora Cappuccio
eleonora.cappuccio@cnr.it

Bahavathy Kathirgamanathan
bahavathy.kathirgamanathan@iais.fraunhofer.de

Salvatore Rinzivillo
salvatore.rinzivillo@cnr.it

Gennady Andrienko
gennady.andrienko@iais.fraunhofer.de

Natalia Andrienko
natalia.andrienko@iais.fraunhofer.de

[1] Institute of Information Science and Technologies "Alessandro Faedo", CNR, Via Moruzzi 1, 56124 Pisa, Italy

[2] KD - Knowledge Discovery, Fraunhofer Institute IAIS, Schloss Birlinghoven, 53757 Sankt Augustin, NRW, Germany

[3] School of Science and Technology, City St George's, University of London, Northampton Sq., London EC1V 0HB, UK