ACTA TECHNICA JAURINENSIS
Vol. XX, No. Y, pp. ZZ-ZZ, 2026
10.14513/actatechjaur.00929

Research Article

Integrating Explainable AI into Model-Driven and Low-Code Enterprise Applications

Dominik Banka[1,*], Tamas Orosz[1], Attila Ritzl[2]

[1]Department of Data Science and Data Engineering, Eötvös Loránd University
Pázmány Péter sétány 1/C, 1117 Budapest, Hungary
[2]SAP Hungary Kft.
Záhony u. 7., 1031 Budapest, Hungary
*e-mail: f64nte@inf.elte.hu

Submitted: 28/11/2025 Revised: 21/01/2026 Accepted: 22/01/2026 Published online: 05/02/2026

Abstract: Explainable artificial intelligence has become increasingly important in enterprise settings, as organisations require transparent and trustworthy decision-support tools. At the same time, low-code and model-driven platforms are widely adopted for building business applications, yet their high level of abstraction often hides the reasoning behind automated recommendations. This study examines how explainability can be systematically incorporated into such environments by introducing a modular approach that separates predictive functions from the generation of human-interpretable explanations. The proposed concept builds on an external reasoning layer that provides both predictive outputs and concise, user-oriented justifications through a unified interface, allowing enterprise systems to present explanations without modifying existing development workflows. To demonstrate the feasibility of the approach, the study applies it to a representative enterprise scenario involving personalised recommendations. The proof-of-concept implementation shows that explanations can be delivered in real time and integrated seamlessly into standard business user interfaces. The results highlight that the proposed solution can enhance transparency, support user trust, and increase the adoption of data-driven features in low-code and model-driven applications. The study contributes a practical architectural pattern that can serve as a foundation for future explainable enterprise systems and provides initial evidence that explanation services can operate effectively alongside contemporary development paradigms.

Keywords: Explainability; Low-code; Model-driven; Integration; Transparency, Enterprise applications

I. INTRODUCTION

Artificial Intelligence (AI) has become integral to enterprise systems, offering predictive insights that enhance automation and decision-making across industries. Yet, the opacity of many machine learning models creates barriers to adoption: business users often struggle to trust recommendations they cannot interpret, while regulatory frameworks increasingly demand transparency and accountability [1]. Explainable Artificial Intelligence (XAI) addresses this challenge by making model reasoning accessible through techniques such as LIME (Local Interpretable Model-Agnostic Explanations) and SHAP (SHapley Additive exPlanations) [2]. In enterprise contexts, explainability is not only a technical feature but also a prerequisite for trust, compliance, and effective human–AI collaboration [3]. The maturity of the XAI field is reflected in several comprehensive reviews that consolidate explanation methods, application domains, and open challenges [4]. Recent systematic reviews indicate that XAI evaluation is still predominantly conducted in isolated, method-centric settings and often relies on anecdotal evidence, with limited insight into how explanations function within integrated, real-world systems [5]. However, despite the growing interest in explainability, little attention has been given to its practical integration into low-code enterprise environments, where the abstraction layers often prevent access to the information required for generating explanations, even though recent research highlights that XAI should span the entire software development lifecycle, from design to deployment [6]. Parallel to these developments, low-code platforms have gained prominence for rapidly building enterprise applications with minimal coding effort. Solutions such as SAP Build or Microsoft

Power Apps empower both developers and business users to assemble applications quickly, often with embedded AI services. These can be complemented by model-driven frameworks such as SAP's Cloud Application Programming Model (CAP), which generates standardized data models and backend services for enterprise applications. However, these integrations generally expose AI as a black-box service, providing limited insight into how predictions are generated and making it difficult for users to rely on the results in practice [7]. This agenda is further accelerated by emerging regulatory expectations (e.g., risk-based governance and transparency duties) and enterprise demands for audit-ready AI [8]. This paper examines how explanation techniques can be embedded into enterprise low-code and model-driven applications in a way that is technically feasible and usable for different organisational roles. The work focuses on a concrete enterprise scenario using SAP's CAP and demonstrates how explanations can be exposed through APIs and consumed in a low-code user interface.

The contributions of this paper are threefold:

- Identify the technical and architectural challenges that arise when integrating XAI methods into low-code and model-driven enterprise platforms.
- Propose a practical architecture that operationalises explainability as a service, enabling predictions and explanations to be retrieved through a unified interface.
- Demonstrating the feasibility of the approach through a prototype built with SAP's CAP and provide initial empirical observations regarding performance and usability.

The remainder of this paper presents the proposed architecture, the prototype implementation, and the results of the initial evaluation.

II. METHODOLOGY AND SYSTEM ARCHITECTURE

This section outlines the methodological basis of the proposed approach and the architectural design that enables explainability to be integrated into low-code enterprise applications. First, the explanation techniques underpinning the solution are introduced, followed by the rationale for separating prediction and explanation logic into an external service. The latter part of the section presents the system architecture and the integration points used in the prototype.

1. Methodological Foundations

The approach presented in this study is based on model-agnostic, post-hoc explanation techniques that can operate independently of the underlying predictive model [9], and recent surveys continue to highlight the importance of emerging XAI techniques that prioritise interpretability, robustness and practical deployability in enterprise settings [10]. This is a practical requirement in enterprise low-code environments, where applications typically function as thin clients and have no direct access to model parameters or training data. Toolkits such as AI Explainability 360 demonstrate how diverse explanation methods can be packaged as reusable services for enterprise applications [11]. In such settings, explanation methods must be capable of running outside the application layer and must return outputs in a structured form that can be consumed by heterogeneous frontends. For these reasons, techniques that produce instance-level feature attributions and can be exposed through standard service interfaces are particularly well suited.

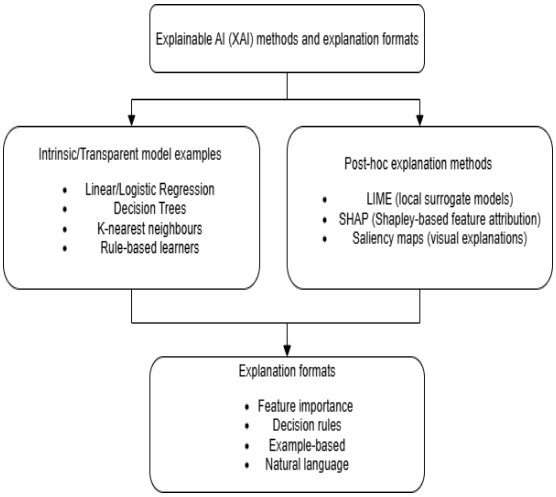

Figure 1. *Intrinsic and post-hoc explanation methods and their role in generating model-level and instance-level explanations.*

Explanation techniques are often grouped into intrinsic and post-hoc categories [12]. Intrinsic models, such as decision trees or linear models, are interpretable by design, as Illustrated in **Fig. 1** Post-hoc methods, by contrast, provide insights into a model's behaviour without modifying the model itself. They can produce explanation formats such as feature-level contributions, rule-based descriptions, example-based reasoning or short textual statements. These outputs are particularly relevant in enterprise settings, where system recommendations must be traceable and aligned with established business logic [13]. Among post-hoc methods, SHAP and LIME were selected for this work due to their practical suitability for externalised explanation services.

- SHAP provides additive feature attributions grounded in cooperative game theory. The resulting values can be represented as structured JSON objects, making them

straightforward to expose through REST or OData services.

- LIME approximates local model behaviour through lightweight surrogate models, offering a computationally efficient alternative when real-time responsiveness is required.

Both techniques generate instance-level explanations aligned with typical low-code UI interaction patterns (cards, tables, tooltips) and support dynamic rendering within application workflows. These methodological considerations in formed the selection of specific explanation techniq ues, which are discussed below. While both SHAP and LIME are architecturally supported by the proposed approach, the current prototype implementation relies exclusively on SHAP-based explanations. SHAP was selected as the primary explanation method due to its stable, additive feature attributions and its suitability for generating concise, human-readable justification strings in an enterprise context. LIME is therefore discussed as a compatible post-hoc technique, but it is not actively used in the experimental evaluation presented in this paper.

2. Rationale for an External Explanation Layer

Low-code enterprise applications are typically designed around strong abstraction layers [14]. While this supports rapid development and shields users from technical complexity, it also limits access to the internal components of predictive models. The application layer usually interacts with AI through pre-packaged blocks or external API calls, receiving a prediction but not the underlying reasoning. As a result, explanation logic cannot be embedded directly into the user interface or the low-code workflow engine. A further constraint is that enterprise backends such as SAP's CAP enforce strict separation of concerns. Business logic, data modelling and service exposure are well defined, but model execution is not part of the runtime environment. This creates a practical need to decouple prediction and explanation from the application backend and to run them in an isolated component. For these reasons, the proposed approach introduces an external explanation layer that encapsulates both the predictive model and the post-hoc explanation methods. This layer is responsible for generating predictions and instance-level explanations and exposes both through uniform REST endpoints. The responses are returned in a structured JSON format, making them compatible with CAP service entities and consumable by low-code frontends. Separating explanation generation into an independent service provides several benefits. First, it allows explanation methods such as SHAP and LIME which may require access to training-time artefacts or local model perturbations to run without constraints imposed by the enterprise runtime. Second, it supports scalability: resource-intensive computations can be isolated and executed asynchronously if needed [15]. Third, it enables multi-frontend interoperability, as the same explanation service can support diverse applications through common APIs. Industry proposals such as AI
FactSheets illustrate how structured documentation of model behaviour can support such governance-oriented integration patterns [16]. This architectural rationale forms the basis of the system design described in the next subsection.

3. System Architecture

The system architecture follows the design principles outlined above and separates prediction and explanation logic into a dedicated service layer. The overall structure consists of three components:

1. The explanation service hosting the predictive model and post-hoc explanation methods,
2. the enterprise backend implemented in SAP's CAP, and
3. the low-code frontend that consumes predictions and explanations through standard service interfaces.

The interaction between the three layers is organised as a simple but extensible request–response flow. As shown in **Fig. 2**, the low-code frontend initiates the process by triggering a prediction or explanation request through an OData V4 service (a standardized REST-based data protocol widely used in enterprise applications) exposed by the CAP backend. The frontend does not communicate directly with the AI module; instead, the backend acts as an integration layer and forwards the request to the external AI service. On the backend, the request is processed by a CAP handler (a server-side business logic component in SAP's Cloud Application Programming Model) implemented in Java. The handler extracts the relevant input fields from the OData entity, invokes the AI service's REST endpoint (/predict or /explain), and receives the result as a JSON object. The response is mapped to CAP entities (*PredictionResult* and *ExplanationResult*), which ensures type safety and makes the results accessible to any frontend capable of consuming OData services. The AI service layer is responsible for executing the predictive model and generating instance-level explanations. The service loads a serialized model artefact (*model.pkl*) and a background dataset needed for SHAP computations.

For explanation requests, the service computes SHAP values or LIME feature weights, depending on the endpoint invoked. Both explanation formats are returned in a structured JSON schema, allowing seamless transformation into CAP entities. Once the backend receives the model outputs, it exposes them to the frontend using standard OData navigation properties. The low-code UI (Fiori Elements or SAP Build Apps) can retrieve the explanation results either by automatic metadata binding or by explicit API calls. Since the returned explanation objects follow a simple tabular structure (feature–value pairs), they can be rendered as analytical cards, tooltips, tables or side panels without custom client-side logic.

This architecture ensures that prediction and explanation logic remain completely decoupled from the enterprise application runtime. It also allows the AI component to evolve independently—for example, different models, updated SHAP background sets or alternative explanation algorithms can be deployed without modifying the backend or the frontend. The use of standard REST and OData interfaces further supports interoperability with additional applications that may require access to prediction or explanation data.

III. PROTOTYPE IMPLEMENTATION

1. Dataset and Model Training

The prototype is based on a supervised learning model that predicts a personalised recommendation score for sport events in a Sport event registration scenario. The training data is stored in a CSV file (training.csv) and contains historical interactions between users and sport events. Each row represents an event instance with categorical and numerical attributes, as well as a target label indicating how attractive the event was to the user (label_score on a 0–100 scale). The dataset used in the prototype consists of 122 historical user–event interaction records collected for a single representative user. This intentionally limited setup reflects an early-stage enterprise scenario, where historical interaction data is often sparse or available only for a small number of users. As a result, the dataset is not intended to support claims about generalizable recommendation performance, but rather to serve as a functional basis for demonstrating prediction and explanation integration.

Given this scope, the primary objective of model training is to obtain a stable and interpretable model suitable for explanation generation, rather than to maximize predictive accuracy. This design choice allows the behaviour of the explainability service to be evaluated in isolation, without confounding effects introduced by heterogeneous user populations or large-scale data distributions.

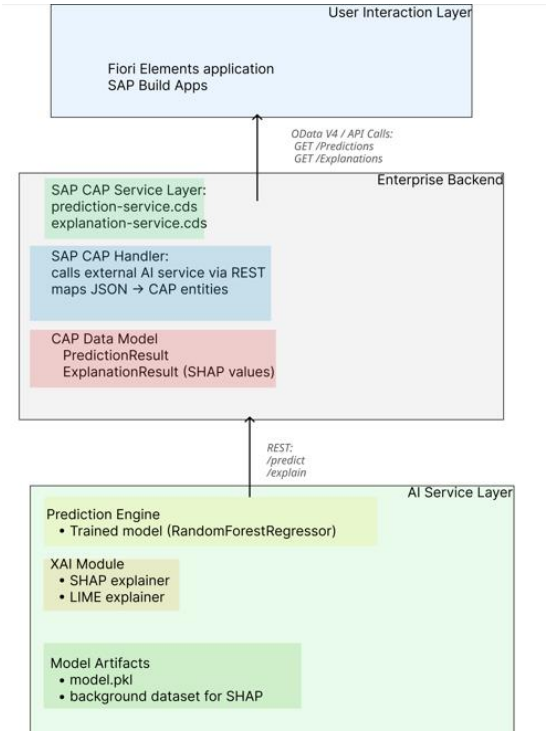

Figure 2. *System architecture of the proposed explainability-as-a-service approach integrating an external XAI service with a CAP-based backend*

The following input features are used:

- **sportType** (categorical): sport category of the event (e.g. yoga, spinning);
- **area** (categorical): location of the event;
- **timeOfDay** (categorical): binned representation of the start time (e.g. morning, noon, afternoon, evening);
- **user_id** (categorical): identifier of the user;
- **capacity** (numeric): maximum number of participants;
- **startHour** (numeric): hour of the event start time.

The data is pre-processed in the training script by cleaning string fields, coercing numeric values, and clipping the target label to the [0, 100] range. Categorical features are encoded using a *OneHotEncoder*, combined with numeric features in a *ColumnTransformer*. The predictive model is a *RandomForestRegressor* configured with 200 trees and out-of-bag estimation [17]. The encoder and the model are wrapped in a single Pipeline object, which is trained on a hold-out split of the dataset and then persisted using *joblib* as *model.pkl*. Alongside the model, a *columns.json* file is generated to store the list of categorical and numerical columns. These artefacts are loaded by the XAI service at runtime and form the basis of both prediction and explanation.

2. XAI Service Implementation

The explanation logic is implemented as an external Flask-based microservice. The service loads the trained scikit-learn pipeline (model.pkl) and the associated metadata (columns.json) on first use. A TreeExplainer from the SHAP library is constructed on top of the random forest component of the pipeline.

The service exposes three HTTP endpoints:

- GET /health – returns a simple status indicator and whether the model artefact is available;
- POST /predict – accepts a JSON payload containing a user_id and an event object, transforms the input into the model's feature representation, and returns a normalised recommendation score between 0 and 100;
- POST /explain – accepts the same input structure and returns a list of feature-level explanations based on SHAP values.

Incoming event objects are converted into model features by the featurize function, which normalises string fields (*sportType*, area, *timeOfDay*) and derives a *startHour* either from the given field or from the event's start time. The */predict* endpoint constructs a pandas DataFrame, calls the pipeline's predict method, and rounds the output to an integer score.

For */explain*, the service passes the preprocessed input through the pipeline's pre step, applies the SHAP explainer to the transformed vector, and maps the resulting SHAP values back to human-readable feature names (one-hot encoded categories plus numeric fields). Features are ranked by absolute contribution, and the most influential ones are selected either by a requested topK parameter or a minimum percentage threshold. Each explanation item contains:

- the feature name,
- the signed SHAP impact,
- the absolute impact,
- the percentage contribution, and,
- a short human-readable reason string generated by the *humanize_reason* helper.

To support enterprise usability, raw SHAP values are not presented directly to end users. Instead, the explanation service transforms feature attributions into short, human-readable reason strings. The most influential features are first identified based on their relative contribution to the prediction, and only a small subset is retained to avoid information overload. These features are then mapped to predefined, business-oriented categories such as sport preference, location convenience, or time-of-day suitability. The resulting labels are combined into concise textual summaries that can be displayed directly in low-code user interfaces. This deterministic transformation ensures that explanations remain concise, consistent, and aligned with enterprise terminology, which is essential for user trust and adoption. The final JSON response consists of a modelVersion identifier, the explanation method ("shap"), and the list of explanation items. This structured format is designed to be consumed easily by both CAP services and low-code user interfaces.

IV. Integration into theCAP Backend

The enterprise integration is implemented in the *EventServiceHandler* of the Sport Events CAP application. The central entity is SportEvent, which represents individual sport events and includes two virtual fields for recommendations:

- **recommended** : Integer – numeric score between 0 and 100;
- **whySummary** : String(500) – short textual justification.

During read operations on *SportEvent*, a *@Before* handler adjusts the CQN query so that all required fields (participants, status, virtual fields) are available for post-processing. The handler uses a configurable base URL (XAI_BASE, read from the XAI_URL environment variable) and an HttpClient to call the /predict endpoint. The returned score is stored in the recommended field of the SportEvent instance. In a second pass, the same events are sent to the */explain* endpoint with a topK parameter (e.g. 3). The service returns a list of explanation objects, from which the handler extracts the human-readable reason strings. These are concatenated into a single summary text separated by bullet characters and assigned to the whySummary field. If the service is unavailable or no explanations are returned, the summary remains empty.

This integration pattern keeps the CAP backend agnostic of the underlying model and explanation method. The CAP service only needs to construct JSON requests and interpret JSON responses, while all model-specific logic is contained in the external Python service.

V. Frontend Behaviour

On the UI side, the Fiori Elements-based list report for SportEvent includes the recommended and *whySummary* fields in the line item annotation. As a result, end-users see, for each event, a numeric recommendation score as well as a short, natural-

language justification (e.g. “Matches your preferred sport · Convenient location · Preferred time of day”).

These fields behave like any other OData properties from the perspective of the UI framework. The underlying explainability logic remains invisible to the application developer using SAP Fiori Elements or, in a future extension, SAP Build Apps. This demonstrates that explainability can be integrated as a reusable service without modifying the low-code frontend itself.

VI. RESULTS

This section summarises the main results of the experimental prototype, focusing on the performance of the model, the behaviour of the explainability service, and the integration outcomes observed in the CAP-based enterprise application. All quantitative results, latency measurements, and example explanations reported in this section are based exclusively on SHAP-based explanations. Although LIME is supported by the proposed architecture, its runtime behaviour and explanation quality were not evaluated in the current prototype and are left for future work.

1. Model Performance

The predictive model was trained on the Sport Events dataset using an 80/20 train–validation split. The RandomForestRegressor, combined with the preprocessing pipeline described earlier, produced consistently high accuracy across both subsets:

- Training set:
  - MAE = **1.12**
  - R^2 = **0.998**
- Validation set:
  - MAE = **1.93**
  - R^2 = **0.990**

Although these values are close to the upper performance bound, they do not indicate overfitting. Instead, they reflect the characteristics of the dataset used in this prototype. The number of samples is limited, the feature space is relatively low-dimensional, and the target behaviour is highly regular, resulting in low intrinsic variance. Consequently, the model is able to approximate the underlying patterns almost perfectly on both the training and validation sets.

These observations imply that the goal of the prototype is not to maximise predictive accuracy, but to provide a stable basis for generating meaningful explanations. The model serves as a functional component within the explainability-as-a-service architecture, ensuring that SHAP-based attributions reflect genuine patterns present in the data rather than noise. Further evaluation with larger and more diverse datasets would be required to assess generalisation performance in real-world enterprise environments.

2. Response Time and Service Behaviour

The runtime characteristics of the external XAI service were assessed using direct HTTP calls, where response times were captured with the **time_total** metric of curl. All measurements were performed on a development laptop equipped with an Intel Ultra 5 CPU and 32 GB RAM, without any caching or precomputation layers enabled.

Table 1. *Mean response latency of the external XAI service endpoints measured during prototype evaluation*

Endpoint	*Mean latency*	*Scenario*
/predict	0.058	Single instance
/explain	0.0056	Top-3 features

As visible in **Table 1.** , for the **/predict** endpoint, a typical request completed in approximately **0.058** seconds, and repeated runs showed only minor variation around this value. This latency is well below the threshold considered acceptable for interactive enterprise applications, where sub-100 ms responses are generally sufficient for seamless user experience.

The **/explain** endpoint, which performs SHAP-based feature attribution, completed significantly faster than expected. In repeated measurements, the total execution time was around **0.0056** seconds, indicating that the explanation logic remains lightweight for the current model size and input structure. Even though SHAP computations can become expensive in larger deployments, the prototype showed no signs of bottlenecks under typical runtime conditions.

Both endpoints consistently returned well-formed JSON responses, and no service interruptions or request failures were observed during the evaluation period. These results confirm that the externalised XAI layer is fast enough to support synchronous invocation from CAP-based enterprise applications. acceptable.

3. Integration Results in the CAP Application

The CAP backend successfully integrated both the prediction and explanation services using synchronous HTTP requests.

After enriching the *SportEvent* entity with the virtual fields recommended and *whySummary*, the Sport Events Fiori User Interface automatically displayed the additional columns without modifications to the frontend code.

Fig. 3 shows the output of the prototype as displayed in a Fiori Elements List Report.

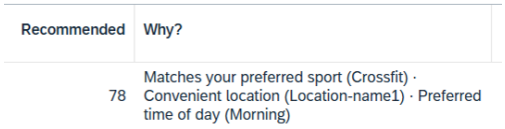

Figure 3. *Output of the prototype in the Fiori Elements UI*

To assess explanation quality at a minimal level, we qualitatively inspected the generated explanations for plausibility and domain consistency. **Table 2**. presents a representative example of a SHAP-based explanation for a single recommendation instance. In this example, sport type preference is identified as the most influential factor, followed by location and time-of-day suitability. This ordering aligns with intuitive expectations in a sport event registration context, where users typically prioritise the type of activity first, and then consider practical constraints such as location and schedule. While this evaluation does not constitute a formal user study or quantitative faithfulness assessment, it provides initial evidence that the explanation service highlights meaningful input factors rather than spurious correlations. This qualitative validation is sufficient for the scope of the present work, which focuses on architectural feasibility and enterprise integration rather than on comparative evaluation of explanation methods.

Table 2. *Example ranking of feature categories for a single sport event recommendation.*

Feature category	*Contribution*	*Explanation*
Sport Type	High	Preferred Sport
Location	Medium	Location fit
Time of day	Medium	Preferred time

The application automatically binds both the recommendation score and the generated explanation text to UI fields, without requiring any custom front-end logic. The explanation text is composed from the top SHAP-based feature contributions and provides short, human-readable reasons such as the preferred sport type, the time-of-day preference, or the convenience of the location. This demonstrates that the proposed approach can surface XAI outputs directly in low-code enterprise user interfaces.

A typical example shown to end users:

- **Recommended:** 87
- **Why?:** "Matches your preferred sport · Convenient location · Preferred time of day"

The explanations updated dynamically for each record during list retrieval, confirming that:

- the CAP - XAI request construction works reliably,
- the XAI service responds with consistent JSON,
- the Fiori UI can consume the enriched entity structure directly.

4. Observed Limitations

Testing identified three current limitations:

- Small training dataset:

The model generalises sufficiently for prototyping but could be improved with more diverse user–event interaction data.

- SHAP performance on large batches:

While individual explanation calls are fast, computing explanations for dozens of events in parallel may require asynchronous processing or caching.

- No frontend-level ranking or UI components:

In this prototype, the frontend only displays the scores and explanations. Interactive drill-down (bar charts, visual SHAP plots) is technically feasible but not implemented.

5. Summary of Results

Overall, the prototype demonstrates that:

- model training and SHAP-based explanations work reliably in an externalised Python service;
- CAP can integrate prediction + explanation services without modifying its internal logic;
- low-code UIs (Fiori Elements) can display XAI outputs with minimal configuration;
- the explanation service remains responsive enough for interactive usage.

The results validate the feasibility of the proposed "explainability-as-a-service" architecture in a realistic enterprise setting.

VII. Discussion

The results of the prototype demonstrate that explainability can be operationalised as a separate service layer in enterprise environments, but they

also highlight important considerations for robustness, scalability, and organisational fit. This section discusses the broader implications of the findings, the architectural trade-offs, and the alignment of the approach with current enterprise development practices. Separating prediction and explanation logic into an external service proved to be an effective strategy for low-code and model-driven environments. The architecture allowed SHAP- and LIME-based reasoning to operate independently of the enterprise backend while maintaining standardised integration points through REST interfaces.

This reinforces the assumption underlying the methodology: explainability must not depend on access to model internals inside the enterprise runtime, particularly in settings where:

- the backend provides only data modelling and business logic (e.g., CAP),
- the frontend is low-code and cannot perform computationally heavy reasoning,
- model execution must remain isolated for maintainability or security reasons.

The successful integration within CAP also shows that enterprise runtime environments are sufficiently flexible to incorporate external AI logic without breaking their internal separation-of-concerns principles. However, the architecture also relies heavily on synchronous HTTP communication. While this was acceptable for the Sport Events scenario, larger-scale enterprise systems may require asynchronous processing, caching, or message-based pipelines (e.g., event-driven architectures) to prevent throughput bottlenecks. The prototype confirmed that low-code frontends can directly consume explanation outputs if they are presented in a structured and lightweight form. The simple text-based summaries generated by the backend were easily incorporated into the Fiori Elements UI. This demonstrates that explainability can be embedded into low-code user interfaces without requiring custom components. These findings align with recent research emphasising that explanations must be tuned to user roles rather than to data scientists, and that information overload can reduce, rather than improve the user trust. Large research programs such as DARPA XAI have similarly stressed the importance of user-tailored explanations [18]. In the Sport Events context, concise explanations were sufficient for decision-making, but regulated or safety-critical domains may require more detailed, auditable output formats.

Although the prototype validates the feasibility of the architecture, multiple limitations must be acknowledged.

- The model was trained on a limited dataset, constraining both predictive validity and explanation fidelity. More varied user–event interaction data would allow stronger conclusions about explanation quality.
- Even though single-instance explanations were fast, scaling to larger batch processing would require optimisation. Techniques such as kernel SHAP, background dataset reduction, or explanation caching may be necessary for high-load scenarios.
- The current work does not examine how different user groups interpret or value explanations. For enterprise adoption, empirical evidence on user comprehension, trust improvement, or perceived usefulness would be highly relevant.

These limitations represent natural next steps for future work and do not undermine the core architectural findings.

VIII. CONCLUSION

This paper presented an explainability-as-a-service architecture designed to integrate post-hoc XAI techniques into enterprise low-code and model-driven environments. The motivation for the work stemmed from two parallel developments: the increased reliance on AI-driven decision support in enterprise applications and the rapid adoption of low-code tools that abstract away technical details but consequently limit access to model internals. These factors make it difficult to deliver transparent, auditable and user-oriented explanations within typical enterprise system landscapes.

The study demonstrated that separating prediction and explanation logic into an external service layer provides a practical and flexible way to embed explainability into heterogeneous enterprise frontends. Using a prototype implementation built with SAP's CAP and a Python-based service exposing SHAP and LIME explanations, we showed that instance-level justifications can be generated on demand and consumed by low-code user interfaces without modifying the underlying application runtime. The results confirm the feasibility of this architectural pattern and highlight its suitability for environments with strict separation-of-concerns principles. At the same time, the findings reveal areas requiring further investigation. Explanation quality and predictive performance depend heavily on training data volume, and the computational cost of SHAP-based reasoning poses scalability challenges for real-time scenarios. Additionally, although the prototype provides functional explanations in a live enterprise context, user-centred evaluation is still needed to assess how

different stakeholder groups interpret and value the explanations produced.

Overall, the proposed approach contributes to the emerging design space of transparent enterprise AI by demonstrating how model-agnostic explanation techniques can be operationalised within low-code and model-driven platforms. Future work will focus on expanding the evaluation with larger datasets, assessing user comprehension, and extending the architecture with monitoring, caching and governance capabilities to support regulated and large-scale deployments.

ACKNOWLEDGEMENT

Supported by the EKÖP-KDP-25 University Research Scholarship Program Cooperative Doctoral Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund.

REFERENCES

[1] S. Thiebes, S. Lins, and A. Sunyaev, "Trustworthy artificial intelligence," *Electron Markets*, vol. 31, no. 2, pp. 447–464, June 2021. https://doi.org/10.1007/s12525-020-00441-4

[2] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why Should I Trust You?': Explaining the Predictions of Any Classifier," presented at the in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD), San Francisco, CA, USA: ACM, 2016, pp. 1135–1144. https://doi.org/10.1145/2939672.2939778

[3] S. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," presented at the Proc. 30th Conf. Neural Information Processing Systems (NeurIPS), Long Beach, CA, USA: Curran Associates, Inc., 2017, pp. 4765–4774. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html

[4] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Muller, "Explaining Deep Neural Networks and Beyond: A Review of Methods and Applications," *Proc. IEEE*, vol. 109, no. 3, pp. 247–278, Mar. 2021. https://doi.org/10.1109/JPROC.2021.3060483

[5] M. Nauta *et al.*, "From Anecdotal Evidence to Quantitative Evaluation Methods: A Systematic Review on Evaluating Explainable AI," *ACM Comput. Surv.*, vol. 55, no. 13s, pp. 1–42, Dec. 2023. https://doi.org/10.1145/3583558

[6] L. Arora, S. S. Girija, S. Kapoor, A. Raj, D. Pradhan, and A. Shetgaonkar, "Explainable Artificial Intelligence Techniques for Software Development Lifecycle: A Phase-specific Survey," arXiv.org. Accessed: Nov. 27, 2025.

[7] F. Haag, K. Hopf, P. M. Vasconcelos, and T. Staake, "Augmented cross-selling through explainable AI -- a case from energy retailing," Aug. 24, 2022, *arXiv*: arXiv:2208.11404. https://doi.org/10.48550/arXiv.2208.11404

[8] P. Pokala, "Artificial Intelligence In Enterprise Resource Planning: A Systematic Review Of Innovations, Applications, And Future Directions," *SSRN Electron. J.*, 2025. https://doi.org/10.5281/zenodo.14170247

[9] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, *et al.*, "Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," *Information Fusion*, vol. 58, pp. 82–115, 2020. https://doi.org/10.1016/j.inffus.2019.12.012

[10] D. E. Mathew, D. U. Ebem, A. C. Ikegwu, P. E. Ukeoma, and N. F. Dibiaezue, "Recent Emerging Techniques in Explainable Artificial Intelligence to Enhance the Interpretable and Understanding of AI Models for Human," *Neural Process Lett*, vol. 57, no. 1, p. 16, Feb. 2025. https://doi.org/10.1007/s11063-025-11732-2

[11] V. Arya, R. K. E. Bellamy, P.-Y. Chen, A. Dhurandhar, M. Hind, S. C. Hoffman, *et al.*, "AI explainability 360: An extensible toolkit for understanding data and machine learning models," *Journal of Machine Learning Research.*, vol. 21, no. 130, pp. 1–6, 2020. https://www.jmlr.org/papers/v21/19-1035.html

[12] R. Guidotti, A. Monreale, S. Ruggieri, F.

AUTHOR CONTRIBUTIONS

Dominik Banka: Conceptualization, Experiments, Theoretical analysis, Writing

Tamás Orosz: Supervision, Review and editing.

Attila Ritzl: Supervision, Review and editing.

DISCLOSURE STATEMENT

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

ORCID

Tamas Orosz https://orcid.org/0000-0003-0595-6522

Attila Ritzl https://orcid.org/0009-0007-2742-4881

Turini, F. Giannotti, and D. Pedreschi, "A survey of methods for explaining black box models," *ACM Comput. Surv.*, vol. 51, no. 5, p. 93:1-93:42, 2018. https://doi.org/10.1145/3236009

[13] C. Meske, B. Abedin, M. Klier, and F. Rabhi, "Explainable and responsible artificial intelligence: A research agenda for understanding the socio-technical implications of AI," *Electron. Markets*, vol. 32, no. 4, pp. 2103–2106, 2022. https://doi.org/10.1007/s12525-022-00607-2

[14] M. Kandaurova, D. Skog, and P. M. Bosch-Sijtsema, "The Promise and Perils of Low-Code AI Platforms," *MIS Q. Exec.*, vol. 23, no. 3, pp. 275–289, 2024. https://doi.org/10.17705/2msqe.00098

[15] V. Belle and I. Papantonis, "Principles and Practice of Explainable Machine Learning," *Front. Big Data*, vol. 4, July 2021. https://doi.org/10.3389/fdata.2021.688969

[16] M. Arnold, R. K. E. Bellamy, M. Hind, S. Houde, S. Mehta, A. Mojsilovic, *et al.*, "FactSheets: Increasing Trust in AI Services through Supplier's Declarations of Conformity," 2018, *arXiv*: arXiv:1808.07261. https://doi.org/10.48550/arXiv.1808.07261

[17] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (XAI)," *IEEE Access*, vol. 6, pp. 52138-52160, 2018. https://doi.org/10.1109/ACCESS.2018.2870052

[18] D. Gunning and David W. Aha, "DARPA's explainable artificial intelligence (XAI) program," *AI Magazine*, vol. 40, no. 2, pp. 44–58, 2019. https://doi.org/10.1609/aimag.v40i2.2850