

Toward Automated Ticket Escalation in Customer Support Processes: From Emails to Explainable Predictions

Furkan Kocak¹ and Gyunam Park²

¹ University of Padova, Padova, Italy

² Eindhoven University of Technology, Eindhoven, The Netherlands
g.park@tue.nl

Abstract. Business processes in customer support often remain undocumented, with critical decisions such as ticket escalation relying on tacit knowledge rather than explicit process models. Automating such process decisions is challenging because the underlying data is unstructured email text, not structured event logs assumed by existing process mining and predictive monitoring approaches. We present a novel pipeline artifact that addresses this gap: a semi-automated four-stage pipeline that extracts structured features from customer support emails using large language models (LLMs) together with lightweight domain-expert validation, and employs explainable machine learning to predict the optimal timing of ticket creation, a key decision point in the support process. Our pipeline generates an initial feature schema from sample threads, consolidates semantically similar features through embedding-based clustering under domain-expert review, extracts cumulative feature values at each email point, and constructs a structured dataset ready for machine learning. We introduce a SHAP-based active feature learning strategy that identifies which missing features, when acquired, would most improve prediction confidence, enabling principled decisions about when to request additional information versus making immediate predictions. We evaluate our approach in two stages: first, on a reproducible synthetic dataset where comparison against six baselines demonstrates that LLM-extracted features provide complementary semantic information, with combined features achieving the best overall performance (ROC-AUC 0.90) while end-to-end LLM classification performs worst (ROC-AUC 0.46), validating our design of using LLMs as feature extractors rather than decision makers; second, on 1,500 email instances from 800 real-world email threads from Nosco, a commercial CRM platform, where we demonstrate effective feature extraction, predictive performance, and active feature learning for practical deployment.

Keywords: Business Process Automation · Feature Extraction · Explainable AI · Active Learning · Large Language Models

1 Introduction

Business processes in customer support are often undocumented and rely on tacit knowledge rather than explicit process models. Despite the significant body of work on process automation in the Business Process Management (BPM) community, many real-world processes, particularly in small- to medium-sized businesses, generate only unstructured communication data (e.g., emails) rather than the structured event logs assumed by most process mining and monitoring techniques. A critical decision point in

the customer support process is determining at which point in an email conversation a support ticket should be created for formal tracking and resolution. This timing decision requires assessing the cumulative technical complexity, urgency and business impact, multi-step resolution requirements, and historical patterns as they evolve throughout the conversation, knowledge that typically resides in experienced agents’ expertise rather than explicit documentation. Automating such implicit process decisions from unstructured data presents a key challenge for BPM.

Existing approaches to ticket escalation typically assume access to pre-existing structured data or require extensive manual feature engineering with substantial domain expert involvement. Montgomery et al. [24] achieved strong recall on IBM support tickets but required analyst interviews to convert implicit business knowledge into explicit features, an approach that small- to medium-sized businesses cannot afford. More broadly, existing BPM approaches such as prescriptive process monitoring [25] and predictive process monitoring [9,26] assume structured event logs or explicit process models. Our domain has neither. Meanwhile, recent advances in large language models (LLMs) have shown strong capabilities in natural language understanding, but their direct deployment as end-to-end decision makers in business-critical applications faces several challenges: hallucination problems that undermine trust [11], opaque decision-making that resists auditing [17], and privacy concerns when processing sensitive communications through cloud services [27].

These challenges motivate an approach that leverages LLMs’ language understanding without relying on them for final decisions. Specifically, there is a need for methods that can: (1) transform raw, unstructured email data into structured representations suitable for traditional machine learning, (2) build explainable models that support transparency and enable human oversight, (3) handle missing or uncertain information gracefully by identifying when additional data collection is warranted, (4) operate effectively with moderate-sized datasets typical of real-world business contexts, and (5) serve as a reusable artifact that can be adapted to other business process domains where decisions rely on unstructured communication data.

This paper addresses the challenge of automating process decisions from unstructured data by proposing a novel pipeline artifact, a reusable system that combines LLM-based feature extraction with explainable machine learning and active feature learning to automate ticket escalation in customer support processes. The novelty lies not in the individual components but in the integrated pipeline design: specifically, how LLMs, semantic clustering, and SHAP-based active learning are orchestrated to address the unique challenge of process automation from unstructured data. Our approach uses LLMs as feature extractors rather than end-to-end decision makers, enabling transparency while leveraging their natural language understanding capabilities to capture the evolving context within email conversations. Our key contributions are as follows:

- A semi-automated pipeline artifact that identifies features from sample threads, consolidates them through embedding-based clustering with bounded domain-expert validation, and extracts values across all threads, reducing 785 initial features to 35 consolidated representations (95.5% reduction) on real-world data.
- A SHAP-based two-stage confidence assessment that withholds process decisions when specific missing features would change the outcome, improving precision from 0.67 to 0.74 while preserving interpretability and auditability.
- Experimental validation against six baselines (end-to-end LLM, metadata, TF-IDF, sentiment/linguistic, rule-based) on a reproducible synthetic dataset (combined

ROC-AUC: 0.90) and on 1,500 email instances from 800 real threads from Nosco, a commercial CRM platform.

2 Related Work

Our work bridges four research areas: process automation and decision support in BPM, LLM-based structured data extraction, machine learning for ticket escalation, and active feature learning.

Process Automation and Decision Support in BPM BPM research has developed predictive [9] and prescriptive [25] process monitoring for automated decision support, achieving strong results in domains with structured event logs and explicit process models [5]. However, these approaches assume well-defined case identifiers, activities, and timestamps. These assumptions fail in customer support for small- to medium-sized businesses, where processes are implicit and generate only unstructured communication. Our pipeline addresses this gap by transforming unstructured email data into structured representations amenable to process decision support.

LLM-Based Feature Extraction from Unstructured Text LLMs can automate transformation of unstructured text into structured representations. EVAPORATE-DIRECT [2] separates schema identification from value extraction but assumes a fixed schema and does not consolidate redundant features that emerge across diverse documents. GraphRAG [6] and KGGen [23] build knowledge graphs from extracted triples and prune for redundancy; healthcare applications [4] reveal accuracy limitations in 38% of reviewed studies. These approaches lack mechanisms for systematically consolidating semantically similar features and handling missing or uncertain extractions, which our pipeline addresses through embedding-based clustering and SHAP-based active feature learning.

Process and Activity Mining from Email Data A dedicated line of BPM research mines process knowledge directly from email corpora. Elleuch et al. [8] discover frequent activities from email logs using pattern-discovery techniques and construct event logs amenable to process mining, and further propose a meta-model formalizing the relationship between emailing systems and business process elements [7]. Khandaker et al. [15] apply transformer models to classify sender intents in emails and map them to process activities in knowledge-intensive processes. Earlier, Mitchell et al. [22] extracted knowledge about users’ activities from raw workstation contents, including email. These approaches focus on reconstructing activities, intents, or process models from email, i.e., recovering the implicit *control flow* of a process. Instead, our work targets *decision point*, i.e., when an ongoing thread should be escalated into a ticket, and produce an ML-ready structured dataset with explainable, confidence-aware predictions.

Machine Learning for Ticket Escalation Montgomery et al. [24] trained XGBoost on 2.5M IBM support tickets (87.36% recall, 2.79% precision, 88.23% workload reduction), but converted implicit business knowledge into explicit features through

analyst interviews, an effort small- to medium-sized businesses cannot afford. Gupta [10] combined exploratory analysis with SHAP for interpretability but still required manual feature engineering. These studies validate ML feasibility for ticket prediction yet assume pre-existing structured data; our work automates feature extraction from raw emails using LLMs while maintaining explainability.

Active Learning and Feature Acquisition Active learning traditionally selects instances to label [28], while *active feature acquisition* (deciding which missing values to query) has received less practical attention [16]. EDDI [20] uses Bayesian experimental design with variational autoencoders but its complexity challenges interpretability, and recent LLM-based approaches [3,13] are black-box. Our SHAP-based approach provides transparent, auditable criteria by assessing both missing-feature importance and prediction stability.

Research Gap No prior work combines LLM-based feature extraction from unstructured process data, semantic consolidation through embedding-based clustering, explainable tree-based prediction, and SHAP-based active feature acquisition. In particular, the email-mining stream above reconstructs activities and process models but does not address operational decision points such as escalation timing, nor does it produce explainable, confidence-aware predictions. Our pipeline fills this gap.

3 Automated Feature Extraction Pipeline

We present a systematic pipeline for transforming raw customer support emails into structured datasets suitable for machine learning. This pipeline serves as a reusable artifact for transforming unstructured process data into structured representations, applicable to any domain where business processes are implicit and documented only through communication records. As illustrated in Fig. 1, the pipeline consists of four stages: (1) feature identification produces an *initial feature schema* from sample threads, (2) semantic clustering and consolidation refines this into a *consolidated feature schema*, (3) feature extraction generates *feature mappings* for all threads, and (4) dataset construction produces the final *structured dataset*. Each stage processes the output of the previous stage, progressively refining raw email data into a structured representation.

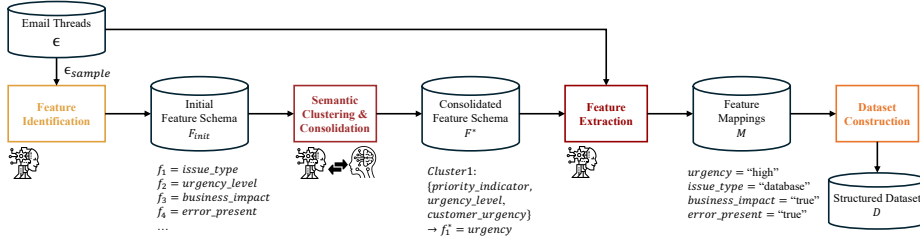


Fig. 1. Automated feature extraction pipeline with four stages. The pipeline transforms raw email threads into a structured dataset.

The input to the pipeline is a collection of email threads $\mathcal{E} = \{E_1, E_2, \dots, E_N\}$, where each thread $E_i = \{e_{i,1}, e_{i,2}, \dots, e_{i,n_i}\}$ is a sequence of emails exchanged between customer and support agent. Each email $e_{i,j}$ contains fields (*sender, subject, body, timestamp*). The output is a structured dataset $D = \{(\mathbf{x}_{i,j}, y_{i,j})\}$ where each instance corresponds to the cumulative knowledge available up to email $e_{i,j}$ in thread E_i . Here, $\mathbf{x}_{i,j} \in \mathbb{R}^q$ is a feature vector representing the cumulative information from emails $\{e_{i,1}, e_{i,2}, \dots, e_{i,j}\}$, and $y_{i,j} \in \{0, 1\}$ indicates whether a ticket should be created at this point in the conversation. We use forward-filling within each thread, where $y_{i,k} = 0$ for all emails $e_{i,k}$ with $k < j^*$ if ticket creation occurs at email e_{i,j^*} .

To illustrate the pipeline, we use a running example throughout this section. Consider an email thread where a customer contacts support about a software issue:

Email 1 (Customer): “Hi, since yesterday’s system update we have been unable to access the inventory module in your system. Our business depends on this module for daily operations, and every hour it is unavailable we lose orders. Please treat this as urgent.”

Email 2 (Support): “Thank you for contacting us. Could you tell us which browser you are using, and whether you see a specific error message when accessing the inventory module?”

Email 3 (Customer): “We have tried both Chrome and Firefox, and the inventory module fails to load in both. Our IT team has looked into it and suspects the module cannot connect to the database.”

In this thread, ticket creation was triggered after Email 3 ($y_{i,3} = 1$) due to the persistent technical nature and business impact, while the earlier emails had $y_{i,1} = y_{i,2} = 0$. We will trace how this example flows through each stage of the pipeline, showing how features are extracted cumulatively at each email point.

3.1 Stage 1: Feature Identification

The first stage takes a sample of email threads $\mathcal{E}_{sample} \subseteq \mathcal{E}$ and uses an LLM to identify features characterizing support interactions relevant to ticket creation decisions, producing an *initial feature schema* $\mathcal{F}_{init}$.

We define a feature schema as a set $\mathcal{F}_{init} = \{f_1, f_2, \dots, f_m\}$, where each feature f_k is a tuple:

$$f_k = (\text{name}_k, \text{type}_k, \text{desc}_k, \text{values}_k)$$

with name_k as the feature identifier, $\text{type}_k \in \{\text{categorical}, \text{numerical}, \text{boolean}\}$ as the data type, desc_k as semantic description, and values_k as an optional set of possible values for categorical features (e.g., $\text{values}_k = \{\text{“high”}, \text{“medium”}, \text{“low”}\}$ for urgency levels), or $\emptyset$ for numerical and boolean features. The LLM receives a prompt that includes both the email thread and the corresponding ticket information (name and description) for threads that resulted in ticket creation, requesting: “Analyze the following email threads along with their associated ticket details and identify features that would be relevant for deciding whether to create a support ticket of this type. For each feature, provide: name, type, description, and for categorical features, also provide a set of possible values.” Because a single thread typically exhibits only a subset of the possible categories, the value set values_k produced by the LLM for that thread is necessarily partial. The value set is therefore not finalized here: it is formed in Stage 2 by taking the union of values observed across all sample threads in $\mathcal{E}_{sample}$ and is then refined by domain experts, who ensure it is comprehensive yet bounded.

For our running example, the LLM might identify features including:
 $f_1 = (\text{issue_type}, \text{cat.}, \text{"Type of technical problem"}, \{\text{"DB_conn."}, \text{"auth."}, \text{"perf."}\})$
 $f_2 = (\text{urgency_level}, \text{cat.}, \text{"Level of issue's urgency"}, \{\text{"high"}, \text{"medium"}, \text{"low"}\})$
 $f_3 = (\text{business_impact}, \text{bool.}, \text{"Affects business operations"}, \emptyset)$
 $f_4 = (\text{error_present}, \text{bool.}, \text{"Contains error message"}, \emptyset)$

These identified features form the initial schema $\mathcal{F}_{init}$. Since the LLM analyzes multiple diverse threads, $\mathcal{F}_{init}$ may contain hundreds of features, many of which are semantically similar or redundant. The next stage addresses this redundancy through consolidation.

3.2 Stage 2: Semantic Clustering and Consolidation

The second stage consolidates the potentially large and redundant initial feature schema $\mathcal{F}_{init}$ into a smaller, more coherent *consolidated feature schema* $\mathcal{F}^*$. We employ a sentence embedding model to map each feature to a dense vector representation based on its name and description:

$$\psi: f_k \rightarrow \mathbf{e}_k \in \mathbb{R}^d$$

where $\mathbf{e}_k$ is the embedding of the concatenation of $name_k$ and $desc_k$. We then apply HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) [21] to identify clusters:

$$\mathcal{C} = \text{HDBSCAN}(\{\mathbf{e}_1, \dots, \mathbf{e}_m\}, \epsilon, \text{min_samples})$$

producing a set of feature clusters $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ where features within each cluster C_j are semantically similar.

For each cluster C_j , a domain expert reviews the features $\{f_k: f_k \in C_j\}$ and either selects the most representative feature $f_j^* \in C_j$, or creates a new consolidated feature f_j^* that better captures the cluster's semantic content. For categorical features, domain experts also review and refine the set of possible values, ensuring they are comprehensive yet constrained to prevent value explosion during extraction: they merge near-duplicate categories and add values known to occur in practice but absent from the sample. In our evaluation (Sect. 5), support managers from the Nosco platform independently reviewed the clusters and resolved disagreements through discussion, requiring substantially less time than manual feature engineering.

Following clustering, domain experts perform post-processing based on feature correlation analysis and missingness patterns. Specifically, *missingness patterns* refer to the proportion of instances for which a feature cannot be extracted: features with missingness exceeding 80% across all email instances are flagged for review, as persistent absence suggests the feature is either too specific or ambiguously defined for reliable LLM extraction. Features with ambiguous definitions that could lead to inconsistent annotations are removed (e.g., subjective features like `issue_described_clearly`), and highly correlated features (Cramér's $V > 0.85$) are merged into unified representations. The consolidated schema is:

$$\mathcal{F}^* = \{f_1^*, f_2^*, \dots, f_p^*\}$$

where $p \ll m$, significantly reducing the feature space dimensionality.

Returning to our running example, suppose the LLM identified three similar features across different threads in $\mathcal{E}_{sample}$:

$f_a = (\text{customer_urgency}, \text{cat.}, \text{"Level of urgency expressed"}, \{\text{"high"}, \text{"medium"}, \text{"low"}\})$

$f_b = (\text{priority_indicator}, \text{cat.}, \text{"Importance of request"}, \{\text{"high"}, \text{"medium"}, \text{"low"}\})$
 $f_c = (\text{urgency_level}, \text{cat.}, \text{"How urgent the issue is"}, \{\text{"high"}, \text{"medium"}, \text{"low"}\})$

After embedding and clustering, these features form cluster C_1 due to their semantic similarity. The domain expert consolidates them into:

$f_1^* = (\text{urgency}, \text{cat.}, \text{"Customer-expressed urgency level"}, \{\text{"high"}, \text{"medium"}, \text{"low"}\})$

This consolidation process is repeated for all clusters, producing the consolidated schema $\mathcal{F}^*$ that serves as the template for systematic feature extraction in the next stage.

3.3 Stage 3: Feature Extraction

With the consolidated feature schema $\mathcal{F}^*$ established, the third stage extracts actual values from *all* individual emails in the dataset. For each email $e_{i,j}$ in thread E_i and each feature $f_k^* \in \mathcal{F}^*$, we use the LLM as an extraction function to obtain the email-specific value.

Formally, we define $\phi_k: e_{i,j} \times f_k^* \rightarrow \mathcal{V}_k \cup \{\perp\}$ as the extraction function that maps an individual email and feature to a value from the feature’s domain $\mathcal{V}_k$ or to $\perp$ (missing) if the information is not present in that email:

$$v_{i,j,k} = \phi_k(e_{i,j}, f_k^*) = \text{LLM}(\text{prompt}(e_{i,j}, f_k^*))$$

where $\text{prompt}(e_{i,j}, f_k^*)$ constructs a query asking the LLM to extract the value of feature f_k^* from email $e_{i,j}$.

The complete set of *feature mappings* for all emails is:

$$\mathcal{M} = \{(e_{i,j}, f_k^*, v_{i,j,k}) : i = 1, \dots, N, j = 1, \dots, n_i, k = 1, \dots, p\}$$

Applying this extraction to our running example with the consolidated schema $\mathcal{F}^*$ yields email-specific values. For instance, at email $e_{i,1}$ (the customer’s initial message):

$$v_{i,1,1} = \phi_1(e_{i,1}, f_1^*) = \text{"high"} \quad (\text{urgency})$$

$$v_{i,1,2} = \phi_2(e_{i,1}, f_2^*) = \perp \quad (\text{issue_type})$$

$$v_{i,1,3} = \phi_3(e_{i,1}, f_3^*) = \text{true} \quad (\text{business_impact})$$

$$v_{i,1,4} = \phi_4(e_{i,1}, f_4^*) = \perp \quad (\text{error_present})$$

Email $e_{i,1}$ conveys urgency and business impact but does not yet identify the technical issue or report an error message, so ϕ_2 and ϕ_4 return $\perp$. The issue type is determined only at $e_{i,3}$, where the customer traces the problem to the database: $\phi_2(e_{i,3}, f_2^*) = \text{"database_connection"}$. The **error_present** feature is never expressed in the thread and stays $\perp$ at every email. These email-level extracted values are stored in $\mathcal{M}$ and serve as the raw input for dataset construction in the final stage.

3.4 Stage 4: Dataset Construction

The final stage transforms the email-level feature mappings $\mathcal{M}$ into a structured dataset D ready for machine learning. For each email $e_{i,j}$ in thread E_i , we construct a raw feature vector by applying forward-filling to the email-specific extracted values. The forward-filling mechanism differs by feature type:

For categorical and numerical features, we use the most recently extracted non-missing value from emails $\{e_{i,1}, e_{i,2}, \dots, e_{i,j}\}$:

$$v_{i,j,k}^{\text{cumulative}} = \begin{cases} v_{i,j,k} & \text{if } v_{i,j,k} \neq \perp \\ v_{i,j-1,k}^{\text{cumulative}} & \text{if } v_{i,j,k} = \perp \text{ and } j > 1 \\ \perp & \text{otherwise} \end{cases}$$

For boolean features, once a value becomes **true**, it persists in all subsequent emails within the thread. This ensures that if a boolean feature is observed as **true** at any point in the conversation, later emails are influenced by this knowledge even if the feature is not explicitly mentioned or extracted as **false**.

The cumulative feature vector is then:

$$\mathbf{x}_{i,j}^{raw} = (v_{i,j,1}^{cumulative}, v_{i,j,2}^{cumulative}, \dots, v_{i,j,p}^{cumulative})$$

where $p = |\mathcal{F}^*|$ is the number of consolidated features.

We then apply standard preprocessing transformations T : categorical features undergo one-hot or ordinal encoding, numerical features are standardized to zero mean and unit variance, boolean features are converted to binary values, and missing values are handled appropriately for the target machine learning algorithm. The preprocessing function produces:

$$\mathbf{x}_{i,j} = T(\mathbf{x}_{i,j}^{raw}) \in \mathbb{R}^q$$

where $q \geq p$ due to encoding expansion (e.g., one-hot encoding of categorical features).

The final *structured dataset* is:

$$D = \{(\mathbf{x}_{i,j}, y_{i,j})\}$$

where each instance pairs a preprocessed feature vector $\mathbf{x}_{i,j}$ (cumulative knowledge up to email $e_{i,j}$) with label $y_{i,j} \in \{0,1\}$ indicating ticket creation timing.

Returning to our running example, from email $e_{i,1}$ we extract **urgency** = **high** and **business_impact** = **true**, while **issue_type** and **error_present** are not yet determinable and return $\perp$. The support reply $e_{i,2}$ asks a clarifying question and adds no new feature values (all extractions return $\perp$), so forward-filling carries the $e_{i,1}$ values unchanged into $\mathbf{x}_{i,2}^{raw}$. At $e_{i,3}$ the customer traces the problem to the database, so **issue_type** = **database_connection** is extracted for the first time; **urgency** retains **high** from its most recent non-missing extraction at $e_{i,1}$, and the boolean **business_impact**, once **true**, persists regardless of re-extraction. The feature **error_present** is never expressed in the thread and remains $\perp$ throughout. This yields three cumulative instances $\mathbf{x}_{i,1}, \mathbf{x}_{i,2}, \mathbf{x}_{i,3}$ with labels $y_{i,1} = y_{i,2} = 0$ and $y_{i,3} = 1$, which enter the structured dataset D .

4 Explainable Prediction with Active Feature Learning

Building on the structured dataset D from the extraction pipeline, we develop an explainable prediction system that determines at each email point whether a ticket should be created based on the cumulative conversation history, while also identifying which missing features, when acquired, would most improve prediction confidence.

4.1 Binary Classification Model

Given a feature vector $\mathbf{x}_{i,j} \in \mathbb{R}^q$ from the structured dataset D , we learn a classification function $h: \mathbb{R}^q \rightarrow \{0,1\}$ that predicts whether a ticket should be created at this point in the conversation based on the cumulative information available up to email $e_{i,j}$.

We employ gradient boosting decision trees (GBDT) [14], which build an ensemble sequentially:

$$h(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(\mathbf{x}) \right)$$

where h_t is the t -th decision tree, α_t is its weight, and T is the number of trees. The model is trained on the structured dataset $D = \{(\mathbf{x}_{i,j}, y_{i,j})\}$ using standard gradient boosting optimization.

4.2 SHAP-Based Local Interpretability

To explain each prediction, we compute SHAP (SHapley Additive exPlanations) [19] values that decompose the model output into contributions from individual features. We choose SHAP because it is the only feature-attribution method with axiomatic guarantees, such as local accuracy, consistency, and missingness. This makes the per-feature contributions $\phi_j(\mathbf{x})$ directly comparable and auditable, enabling the active feature learning strategy that ranks missing features by $|\phi_j(\mathbf{x})|$. Crucially, for tree models that handle missing values natively (via default split directions), TreeSHAP attributes the prediction impact of a missing feature to that feature, yielding a non-zero $|\phi_j(\mathbf{x})|$ without any prior imputation. This lets Algorithm 1 rank missing features by importance directly, rather than having to impute surrogate values first.

For each prediction $h(\mathbf{x})$, SHAP assigns a contribution $\phi_j(\mathbf{x})$ to feature j based on cooperative game theory, satisfying:

$$h(\mathbf{x}) = \phi_0 + \sum_{j=1}^q \phi_j(\mathbf{x})$$

where ϕ_0 is the expected model output (baseline) and $\phi_j(\mathbf{x})$ represents the impact of feature j on the prediction. Features with larger $|\phi_j(\mathbf{x})|$ have greater influence on the model’s decision. For tree-based models, TreeSHAP [18] computes these values efficiently in polynomial time.

For our running example, at the decision point $e_{i,3}$ TreeSHAP might produce the following contributions: $\phi_1(\mathbf{x}_{i,3}) = +0.35$ (urgency = high), $\phi_2(\mathbf{x}_{i,3}) = +0.28$ (issue_type = database), $\phi_3(\mathbf{x}_{i,3}) = +0.15$ (business_impact = true), $\phi_4(\mathbf{x}_{i,3}) = +0.22$ (error_present = missing).

Even for missing features, SHAP computes importance values based on how the tree ensemble handles missing data (e.g., through default split directions). These SHAP values flow into the active learning module, which uses them to assess prediction confidence and identify informative features to query.

4.3 Active Feature Learning Strategy

In practice, extracted features often contain missing values (either because the LLM extraction was uncertain or because the information is genuinely absent from the email thread). Requesting additional information from customers incurs time and effort costs. Our active feature learning strategy identifies which features, when queried, would most improve prediction confidence, enabling the system to make informed decisions about when to request more information versus making an immediate prediction.

Algorithm 1 presents the complete prediction refinement procedure. The algorithm takes as input an instance $\mathbf{x}$, the trained model h , and confidence thresholds, then decides whether to make an immediate prediction or request additional information. Intuitively, before committing to a prediction the algorithm asks two questions: (i) *are important features missing?* and (ii) *would plausible values for those missing features change the decision?* It requests additional information only when the answer to either

Algorithm 1 Active Feature Learning for Prediction Refinement

```

1: Input: Instance  $\mathbf{x}$ , model  $h$ , thresholds  $\theta_{miss}, \theta_{stab}$ 
2: Output: Prediction decision and optional feature query
3:
4: Compute SHAP values  $\{\phi_j(\mathbf{x})\}_{j=1}^q$  using TreeSHAP
5: Identify missing features  $M(\mathbf{x}) = \{j : x_j = \perp\}$ 
6: Compute  $\text{MissingImpact}(\mathbf{x}) = \sum_{j \in M(\mathbf{x})} |\phi_j(\mathbf{x})|$ 
7:
8: if  $\text{MissingImpact}(\mathbf{x}) > \theta_{miss}$  then
9:   Select  $j^* = \arg\max_{j \in M(\mathbf{x})} |\phi_j(\mathbf{x})|$ 
10:  return (WITHHOLD,  $j^*$ )
11: end if
12:
13: Simulate imputation: For each  $j \in M(\mathbf{x})$ , create  $\mathbf{x}^{(j,v)}$  with  $x_j = v$ 
14:   for  $v$  sampled from training distribution
15: Compute predictions  $\{h(\mathbf{x}^{(j,v)})\}$  and calculate variance  $\sigma^2$ 
16:
17: if  $\sigma^2 > \theta_{stab}$  then
18:   Select  $j^* = \arg\max_{j \in M(\mathbf{x})} |\phi_j(\mathbf{x})|$ 
19:   return (WITHHOLD,  $j^*$ )
20: end if
21:
22: return (PREDICT,  $h(\mathbf{x})$ )

```

question is yes; otherwise it predicts immediately. The algorithm operates in three phases, each assessing a different aspect of prediction confidence.

Phase 1: Missing Feature Impact. Phase 1 (Algorithm 1, lines 4–11) flags instances whose missing features are jointly important: it computes $\text{MissingImpact}(\mathbf{x}) = \sum_{j \in M(\mathbf{x})} |\phi_j(\mathbf{x})|$ over $M(\mathbf{x}) = \{j : x_j = \perp\}$ and, if it exceeds θ_{miss} , withholds the prediction and returns $j^* = \arg\max_{j \in M(\mathbf{x})} |\phi_j(\mathbf{x})|$ as the feature to request. For our running example, only `error_present` is missing, so $\text{MissingImpact}(\mathbf{x}_{i,3}) = |\phi_4| = 0.22$. If we set $\theta_{miss} = 0.3$, the impact is below threshold and the algorithm proceeds to the next phase.

Phase 2: Prediction Stability. Phase 2 (lines 13–18) catches instances whose prediction is sensitive to the unknown values: it imputes each missing feature with values sampled from $P(x_j|D)$, computes the variance $\sigma^2 = \text{Var}(\{h(\mathbf{x}^{(j,v)})\})$ of the resulting predictions, and withholds with the same j^* rule if $\sigma^2 > \theta_{stab}$. For our running example, we might simulate `error_present` $\in \{\text{true}, \text{false}\}$ and observe that both imputations yield $h(\mathbf{x}_{i,3}) = 1$, resulting in low variance. The prediction is stable despite the missing feature.

Phase 3: Confident Prediction. If both checks pass, the algorithm returns $h(\mathbf{x})$. For our running example, both criteria pass and the algorithm returns (PREDICT, $h(\mathbf{x}_{i,3}) = 1$), recommending ticket creation based on the high urgency, database-related issue type, and business impact, even without explicit confirmation of an error message.

Design rationale and thresholds. The two checks address complementary failure modes: missing-impact flags absent influential features, while stability flags decision boundaries passing through the uncertainty region of those features. Both prioritize queries by SHAP importance, minimizing customer interaction per query. Lower thresholds increase caution; higher thresholds increase automation. As a concrete starting point, we recommend calibrating θ_{miss} from the empirical distribution of MissingImpact

on a held-out set (e.g., the 75th percentile, flagging the most uncertain quartile) and θ_{stab} from the observed prediction variance on the same set; both can then be shifted along the automation–quality trade-off.

5 Application and Experimental Evaluation

We validate our pipeline artifact through application to Nosco³, a commercial CRM platform that manages customer support interactions for small- to medium-sized businesses. This section presents the experimental setup, evaluates our approach on a reproducible synthetic dataset, and reports results from applying the full pipeline to real-world customer support data⁴.

5.1 Experimental Setup

Nosco processes email communications between businesses and customers. Support agents manually decide whether incoming emails should be converted into formal support tickets, requiring understanding of technical issues, urgency, business impact, and historical context. This business process is not explicitly documented and relies on agent experience; the email data constitutes a form of unstructured process execution data from which no structured event log can be directly derived.

We collected 800 email threads from Nosco’s database (only threads that resulted in ticket creation). The dataset was structured with one instance per email, representing cumulative knowledge up to that point. Using forward-filling within each thread, emails prior to ticket creation receive label $y_{i,j}=0$, while the email triggering ticket creation receives $y_{i,j}=1$, resulting in 1,500 total instances with natural class imbalance. We used stratified 5-fold cross-validation.

For feature extraction, we employed **Llama 3.1 8B Instruct** (primary) and **Qwen 2.5 7B Instruct** (validation) via Openrouter API. We deliberately selected models in the 7–8B range: they can be deployed locally so that sensitive customer communications need not be sent to third-party services, and they keep the pipeline reproducible. **Qwen 2.5 7B** serves as an independent second model to cross-check extraction robustness. Feature embedding used **mxbai-embed-large-v1** (1024 dimensions), chosen for its strong performance on semantic-similarity benchmarks and its suitability for the short *name+description* strings being embedded. For classification, we benchmarked six models: CATBoost, XGBoost, LightGBM, Random Forest, Explainable Boosting Machine (EBM), and Naive Bayes (baseline), using default hyperparameters for each model. Gradient-boosted decision trees are well suited to this setting: they handle mixed categorical and numerical features with frequent missing values natively, are consistently strong on tabular data of this scale, and admit exact TreeSHAP explanations. Benchmarking six models rather than fixing one keeps the choice empirical; EBM and Naive Bayes additionally serve as interpretable references.

For the baseline comparison, we additionally use a publicly available synthetic dataset of 350 email threads covering the same domain of industrial equipment and CRM support. The dataset includes 238 threads that resulted in ticket creation and 112 threads (32%)

³ <https://leanbit.eu/en/>

⁴ The source code for the experiments is available at <https://doi.org/10.5281/zenodo.20392524>.

where issues were resolved without escalation, providing both positive and true-negative instances. Thread lengths vary from 2 to 10 emails (mean ≈ 5 , std > 1.5), with ticket creation occurring at variable points in the conversation (not always the last email), yielding approximately 1,500 total instances with a positive rate of $\sim 20\%$. This design avoids the trivial correlation between email position and label that would bias metadata-only baselines, enabling reproducible comparison against alternative approaches.

5.2 Evaluation on Synthetic Dataset

To assess the value of our LLM-based feature extraction approach, we compare against six baselines on the publicly available synthetic dataset (350 threads, $\sim 1,500$ instances, $\sim 20\%$ positive rate). We apply the same pipeline to this dataset, producing LLM-extracted features, and compare against: (1) *TF-IDF*: term frequency-inverse document frequency [1], bag-of-words features that weight each term by how characteristic it is of a message (500 features, 1–2 grams) with classifiers, (2) *Sentiment/Linguistic*: VADER (a lexicon- and rule-based sentiment analyzer) [12] sentiment scores, urgency/frustration keyword counts, and linguistic indicators (question/exclamation counts, all-caps ratio), (3) *Rule-Based*: handcrafted heuristic rules based on thread length, urgency keywords, and consecutive customer messages, (4) *Metadata-only*: conversational features (email count, sender patterns, word counts), and (5–6) *End-to-End LLM*: directly prompting Llama 3.1 8B in both zero-shot and few-shot (with chain-of-thought and escalation criteria) variants. All ML-based models use 5-fold stratified cross-validation with default hyperparameters.

Table 1. Baseline comparison on publicly available synthetic dataset (350 threads, $\sim 1,500$ instances, $\sim 20\%$ positive rate; 5-fold CV, mean $\pm$ 95% CI). Best result per column in bold.

Features	Model	AUC	Precision	Recall	F1
<i>LLM-Extracted Features (Ours)</i>					
LLM + Metadata	CATBoost	0.83 ± 0.05	0.72 ± 0.04	0.84 ± 0.05	0.77 ± 0.05
LLM + TF-IDF	CATBoost	0.88 ± 0.07	0.80 ± 0.04	0.87 ± 0.04	0.83 ± 0.02
All Combined	CATBoost	0.90 ± 0.05	0.79 ± 0.04	0.91 ± 0.02	0.85 ± 0.02
<i>Text-Based Baselines</i>					
TF-IDF	CATBoost	0.61 ± 0.09	0.56 ± 0.09	0.59 ± 0.10	0.57 ± 0.09
Sentiment/Ling.	XGBoost	0.69 ± 0.08	0.65 ± 0.05	0.65 ± 0.09	0.65 ± 0.06
<i>Structural Baselines</i>					
Metadata only	LightGBM	0.70 ± 0.02	0.62 ± 0.02	0.72 ± 0.05	0.66 ± 0.02
Rule-Based	Heuristic	–	0.80 ± 0.13	0.24 ± 0.05	0.37 ± 0.06
<i>Direct LLM Classification</i>					
End-to-End LLM	zero-shot	0.46 ± 0.00	0.46 ± 0.00	0.45 ± 0.00	0.46 ± 0.00
End-to-End LLM	few-shot	0.48 ± 0.00	0.49 ± 0.00	0.79 ± 0.00	0.60 ± 0.00

Tab. 1 presents results across all baselines. The **Model** column reports, for each feature set, the best-performing classifier among the six benchmarked in Tab. 2; rows therefore differ (e.g., CATBoost for LLM-based features, LightGBM for metadata). The *Rule-Based* and *End-to-End LLM* rows are not learned classifiers and are labeled by their method (heuristic, zero-shot, few-shot). Three key findings emerge. *First, LLM-extracted features provide strong complementary value.* Combining all feature

sources achieves the best performance (AUC: 0.90 ± 0.05 , F1: 0.85 ± 0.02), with LLM features improving metadata-only performance by 0.13 AUC and TF-IDF-only by 0.27 AUC, demonstrating that semantic, structural, and lexical signals capture distinct predictive information. *Second, standalone baselines show limited discriminative power.* TF-IDF (AUC: 0.61), metadata (AUC: 0.70), and sentiment features (AUC: 0.69) each carry moderate but insufficient predictive value. The rule-based heuristic achieves high precision (0.80) but very low recall (0.24). Critically, LLM-extracted features uniquely enable SHAP-based explanations that identify *which specific factors* drive predictions, enabling the active feature learning strategy described in Sect. 4. *Third, end-to-end LLM classification performs worst.* Both zero-shot (AUC: 0.46) and few-shot (AUC: 0.48) variants fail to make nuanced timing decisions, validating our design choice of using LLMs as feature extractors rather than end-to-end decision makers.

5.3 Application to Real-World Customer Support

We now apply the complete pipeline to real-world customer support interactions from Nosco.

Feature Extraction Results We evaluated the feature extraction pipeline by applying it to 100 randomly sampled email threads and measuring the reduction in features through semantic clustering.

The LLM generated 785 initial features across diverse categories: damage and repair indicators, equipment specifications (including product models), return and shipping logistics, follow-up requirements, documentation requests, and contextual metadata (e.g., `urgency`, `warranty_status`). Only `company` and `product` were pre-existing CRM metadata; all other features were extracted from email content.

We embedded each feature using `mxbai-embed-large-v1` and applied HDBSCAN with $\epsilon=0.5$ and `min_samples=2`, which identified 24 semantically coherent clusters from 257 features. The remaining 528 features (67.26% of all features) were classified as noise by HDBSCAN. Domain experts reviewed both clusters and noise, identifying 11 additional informative features from noise that had insufficient density for automatic clustering.

Following clustering, domain experts performed post-processing based on feature correlation analysis and missingness patterns. Features with ambiguous definitions were removed (e.g., `issue_described_clearly`, `follow_up_required`). Highly correlated features (Cramér’s $V > 0.85$) were merged (e.g., `repair_action_by_servien`, `dce_provider`, `repair_action_taken` with correlation up to 0.93 were merged into `customer_service_action`). The two domain experts (support managers from the Nosco platform with 3+ years of customer support experience) completed this review in approximately 4 hours of total effort, substantially less than manual feature engineering from scratch.

The consolidation process reduced the feature set from 785 to 35 features (95.5% reduction in feature space). The 35 consolidated features comprised: 12 categorical features, 22 boolean features, and 1 numerical feature. We used `Llama 3.1 8B Instruct` to extract values for all 35 features across 1,500 email instances (cumulative knowledge at each email point), producing the final dataset. Missing values were present in all of the features, primarily in features requiring specific technical details not always mentioned in emails (e.g., `damage_consequences`, `cause_of_damage`, `error_code`).

To assess LLM extraction robustness, we validated a subset of extractions using **Qwen 2.5 7B Instruct** as a secondary model. The two models showed consistent extraction patterns, with discrepancies primarily occurring in ambiguous cases where the email text did not clearly indicate a feature value, exactly the cases that result in missing values in the final dataset.

Predictive Performance We benchmarked six classification models on the Nosco dataset (800 threads, 1,500 instances) for predicting ticket creation timing using LLM-extracted features. We report precision, recall, F1, and ROC-AUC with 95% confidence intervals (5-fold CV).

Tab. 2 presents results from 5-fold cross-validation. CATBoost achieved the best overall performance with ROC-AUC of 0.73 ± 0.04 , F1 score of 0.67 ± 0.04 , and balanced precision (0.67 ± 0.03) and recall (0.67 ± 0.04). Its native handling of categorical features through ordered target encoding contributed to superior performance. CATBoost obtained narrow confidence intervals across all metrics, demonstrating stability across data subsets. XGBoost (AUC: 0.65 ± 0.05) and LightGBM (AUC: 0.66 ± 0.05) showed performance comparable to Naive Bayes baseline (AUC: 0.65 ± 0.05), while EBM (AUC: 0.66 ± 0.04) showed competitive performance. Random Forest provided lower performance (AUC: 0.63 ± 0.05). These results confirm CATBoost’s superior performance for predicting ticket creation timing.

Active Feature Learning We evaluated whether the SHAP-based active feature learning strategy (Algorithm 1) improves prediction quality by selectively withholding predictions when confidence is low. We analyzed 150 test instances from the Nosco dataset, applying the algorithm to each to determine whether to predict immediately or withhold and request additional features.

Tab. 3 presents the results. The baseline configuration uses CATBoost making predictions on all 150 test instances without filtering, achieving performance consistent with the 5-fold cross-validation results (AUC: 0.73, Precision: 0.67, Recall: 0.67). With confidence filtering via Algorithm 1, predictions were made on only 76 instances (50.7% coverage) that satisfied both criteria. Precision increased from 0.67 to 0.74 (10.4% relative), reducing false positives. ROC-AUC improved from 0.73 to 0.75. Recall decreased from 0.67 to 0.61 (9.0% relative), reflecting reduced coverage from withholding uncertain predictions.

To illustrate Algorithm 1, we present a case where CATBoost predicted ticket creation (probability = 0.521) with weak confidence. The algorithm computes missing feature

Table 2. Performance comparison of classification models on Nosco dataset (5-fold CV, mean $\pm$ 95% CI)

Model	AUC	F1	Precision	Recall
CATBoost	0.73 ± 0.04	0.67 ± 0.04	0.67 ± 0.03	0.67 ± 0.04
XGBoost	0.65 ± 0.05	0.59 ± 0.03	0.59 ± 0.03	0.59 ± 0.03
LightGBM	0.66 ± 0.05	0.61 ± 0.05	0.61 ± 0.05	0.61 ± 0.05
Random Forest	0.63 ± 0.05	0.57 ± 0.04	0.57 ± 0.04	0.57 ± 0.04
EBM	0.66 ± 0.04	0.62 ± 0.05	0.62 ± 0.04	0.62 ± 0.04
Naive Bayes	0.65 ± 0.05	0.61 ± 0.04	0.61 ± 0.04	0.61 ± 0.04

Table 3. Impact of SHAP-based active feature learning on 150-instance test set (Nosco)

Configuration	AUC	Precision	Recall	F1	Coverage
Baseline (all predictions)	0.73	0.67	0.67	0.67	100%
With Algorithm 1 filtering	0.75	0.74	0.61	0.67	50.7%

Table 4. Selected simulation results showing how different values for missing features **errors** and **user_action** affect prediction (baseline probability: 0.521). Label flip occurs when probability drops below 0.5.

errors	User _ Action	Probability	Label Flip
Alarm _ Indicated	no	0.47	Yes
Alarm _ Indicated	yes	0.48	Yes
Software _ Mentioned	no	0.38	Yes
Software _ Mentioned	yes	0.39	Yes
Welding _ Performance	yes	0.50	No

impact (Phase 1). With θ_{miss} calibrated from the SHAP value distribution, this impact is significant, triggering Phase 2.

The algorithm simulates plausible imputations. Tab. 4 shows selected results. Predicted probabilities range from 0.38 (Software_Issue) to 0.50 (Welding_Performance), exceeding θ_{stab} . Both criteria being met, the algorithm returns (WITHHOLD, $j^* = \text{errors}$), where errors has the highest $|\phi_j| = 0.14$ among missing features. This demonstrates that missing critical technical details prevented the model from making an accurate decision. Notably, this type of targeted information request is only possible with interpretable LLM-extracted features; metadata baselines cannot identify which specific information to request from customers.

6 Discussion

Interpreting the results. Combining LLM-extracted, structural, and lexical features gives the best performance (ROC-AUC 0.90 on the synthetic dataset), and each source adds over the others, so the three capture different signals: semantic context, conversational structure, and surface lexical cues. End-to-end LLM classification performs worst (ROC-AUC 0.46–0.48), because deciding *when* to escalate requires weighing cumulative, partly quantitative evidence across a thread, which a single generative pass does not handle reliably. This supports our positioning of the LLM as a feature extractor rather than a decision maker: the predictive model itself uses established gradient-boosting methods, so the baseline comparison isolates the value added by LLM-extracted features rather than advancing classification. The drop from synthetic (0.90) to real-world (0.73) performance reflects the noise, class imbalance, and missingness of real email data; we treat 0.73 as a realistic lower bound for deployment.

Treating withheld predictions as abstentions When Algorithm 1 withholds, it issues a *request for information*, not a “do not escalate” decision. Scoring a withheld instance as a negative prediction would conflate uncertainty with a substantive decision and penalize the system for recognizing that it lacks information. We therefore evaluate withheld instances as abstentions and report coverage alongside precision and recall, making the precision–coverage trade-off explicit (Tab. 3).

Scalability of the human-in-the-loop step Expert involvement is limited to schema construction, takes roughly four hours in our study (well below the manual feature engineering required by prior work), and is incurred once rather than per thread. It can be reduced further by letting the LLM pre-rank clusters by coherence so that experts confirm only low-confidence ones.

7 Conclusion and Future Work

This paper presented a novel pipeline artifact for automating process decisions from unstructured data, demonstrated through ticket escalation in customer support processes. The pipeline combines LLM-based feature extraction with semantic clustering and SHAP-based active feature learning, addressing a gap in BPM tooling where existing approaches require structured event logs. Validated on 1,500 email instances from 800 real-world threads from Nosco, our approach consolidates 785 LLM-extracted features into 35 representations (95.5% reduction), achieves ROC-AUC of 0.90 when combining all feature sources on a synthetic dataset and 0.73 with CATBoost on real-world data, and improves precision from 0.67 to 0.74 through confidence-based filtering.

Our work has several limitations. First, LLM-based feature extraction is stochastic across models and prompts; we mitigated this by validating with Qwen 2.5 7B and involving experts in consolidation. Second, the pipeline was validated on one domain (industrial equipment and CRM support); transferability requires further study. Third, threads are treated independently, ignoring cross-thread process state (e.g., queue load, SLA deadlines). Fourth, the active feature learning strategy is evaluated indirectly via downstream precision and coverage; a direct comparison against an oracle is left for future work. Fifth, the stability check imputes missing values from the marginal training distribution rather than conditioning on thread context, which may mis-state instability for atypical threads. Sixth, the real-world Nosco dataset contains only threads that resulted in ticket creation (within-thread negatives arise through forward-filling), which may bias the real-world evaluation. Finally, the real-world data cannot be publicly released due to privacy constraints; we mitigate this by releasing the code and a public synthetic dataset.

References

1. Aizawa, A.N.: An information-theoretic perspective of tf-idf measures. *Inf. Process. Manag.* **39**(1), 45–65 (2003)
2. Arora, S., et al.: Language models enable simple systems for generating structured views of heterogeneous data lakes. *Proceedings of the VLDB Endowment* **17**(2), 92–105 (2023)
3. Bayer, M., Lutz, J., Reuter, C.: ActiveLLM: Large Language Model-based Active Learning for Textual Few-Shot Scenarios. *arXiv e-prints arXiv:2405.10808* (2024)
4. Chen, D., Alnassar, S.A., Avison, K.E., Huang, R.S., Raman, S.: Large language model applications for health information extraction in oncology: Scoping review. *JMIR Cancer* **11**, e65984 (2025)
5. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: *Fundamentals of Business Process Management*. Springer, 2nd edn. (2018)
6. Edge, D., et al.: From local to global: A graph RAG approach to query-focused summarization. *CoRR* **abs/2404.16130** (2024)
7. Elleuch, M., Assy, N., Laga, N., Gaaloul, W., Ismaili, O.A., Benatallah, B.: A meta model for mining processes from email data. In: *SCC*. pp. 152–161. IEEE (2020)

8. Elleuch, M., Ismaili, O.A., Laga, N., Gaaloul, W., Benatallah, B.: Discovering activities from emails based on pattern discovery approach. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM Forum. pp. 88–104. LNBIP, Springer (2020)
9. Francescomarino, C.D., Ghidini, C.: Predictive process monitoring. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook, LNBIP, vol. 448, pp. 320–346. Springer (2022)
10. Gupta, S.: A hybrid machine learning framework of gradient boosting decision tree and sequence model for predicting escalation in customer support. In: IEEE International Conference on Big Data. pp. 5511–5518. IEEE (2020)
11. Huang, L., et al.: A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Manag. Inf. Syst.* **43**(2) (2025)
12. Hutto, C.J., Gilbert, E.: VADER: A parsimonious rule-based model for sentiment analysis of social media text. In: Adar, E., Resnick, P., Choudhury, M.D., Hogan, B., Oh, A. (eds.) ICWSM. The AAAI Press (2014)
13. Jeong, D.P., Lipton, Z.C., Ravikumar, P.K.: Llm-select: Feature selection with large language models. *Transactions on Machine Learning Research* **2025** (2025)
14. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.: Lightgbm: A highly efficient gradient boosting decision tree. In: Guyon, I., et al. (eds.) NeurIPS. pp. 3146–3154 (2017)
15. Khandaker, F., Senderovich, A., Zhao, J., Cohen, E., Yu, E., Carbajales, S., Chan, A.: Transformer models for mining intents and predicting activities from emails in knowledge-intensive processes. *Eng. Appl. Artif. Intell.* **128**, 107450 (2024)
16. Li, Y., Oliva, J.: Active feature acquisition with generative surrogate models. In: ICML. Proceedings of Machine Learning Research, vol. 139, pp. 6450–6459 (2021)
17. Li, Y., Tan, Z., Liu, Y.: Privacy-Preserving Prompt Tuning for Large Language Model Services. arXiv e-prints arXiv:2305.06212 (2023)
18. Lundberg, S.M., Erion, G.G., Chen, H., DeGrave, A.J., Prutkin, J.M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., Lee, S.: From local explanations to global understanding with explainable AI for trees. *Nat. Mach. Intell.* **2**(1), 56–67 (2020)
19. Lundberg, S.M., Lee, S.: A unified approach to interpreting model predictions. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) NeurIPS. pp. 4765–4774 (2017)
20. Ma, C., Tschitschek, S., Palla, K., Miguel Hernández-Lobato, J., Nowozin, S., Zhang, C.: EDDI: efficient dynamic discovery of high-value information with partial VAE. In: ICML. vol. 97, pp. 4234–4243. PMLR (2019)
21. McInnes, L., Healy, J., Astels, S.: hdbscan: Hierarchical density based clustering. *J. Open Source Softw.* **2**(11), 205 (2017)
22. Mitchell, T.M., Wang, S.H., Huang, Y., Cheyer, A.: Extracting knowledge about users’ activities from raw workstation contents. In: AAAI. pp. 181–186. AAAI Press (2006)
23. Mo, B., et al.: Kggen: Extracting knowledge graphs from plain text with language models. *CoRR abs/2502.09956* (2025)
24. Montgomery, L., Damian, D., Bulmer, T., Quader, S.: Customer support ticket escalation prediction using feature engineering. *Requir. Eng.* **23**(3), 333–355 (2018)
25. Padella, A., de Leoni, M., Dogan, O., Galanti, R.: Explainable process prescriptive analytics. In: Burattin, A., Polyvyanyy, A., Weber, B. (eds.) ICPM. pp. 16–23. IEEE (2022)
26. Park, G., van der Aalst, W.M.: Operational process monitoring: An object-centric approach. *Computers in Industry* **164**, 104170 (2025)
27. Sun, T., Shao, Y., Qian, H., Huang, X., Qiu, X.: Black-box tuning for language-model-as-a-service. In: ICML. Proceedings of Machine Learning Research, vol. 162, pp. 20841–20855. PMLR (2022)
28. Xu, Y., Sun, F., Zhang, X.: Literature survey of active learning in multimedia annotation and retrieval. In: Lu, K., Mei, T., Wu, X. (eds.) ICIMCS. pp. 237–242. ACM (2013)