Does This Make Sense? Machine Learning-based Detection of Semantic Anomalies in Business Processes

Julian Caspary, Adrian Rebmann, and Han van der Aa

University of Mannheim, Germany
julian.yuya.caspary@students.uni-mannheim.de
{rebmann,han.van.der.aa}@uni-mannheim.de

Abstract. The detection of undesired behavior is a key task in process mining, supported by techniques for conformance checking and anomaly detection. A downside of conformance checking, though, is that it requires a process model as a basis, limiting its applicability, whereas existing anomaly detection techniques look for statistically infrequent behavior, even though infrequency does not necessarily imply undesirability. The recently introduced concept of semantic anomaly detection overcomes these issues by detecting behavior that stands out from a semantic point of view, such as a claim being paid after it has been rejected. In this manner, it detects behavior that is undesirable, while its grounding in natural language analysis allows it to consider behavioral regularities extracted from other processes, alleviating the need to have a process model available. However, the state-of-the-art approach for semantic anomaly detection, a rigid, rule-based approach, is limited in its scope and accuracy. Therefore, we propose a machine learning-based alternative, which uses a classifier trained to recognize whether observed process behavior is normal or anomalous. Our experiments show that this learning-based approach greatly outperforms the state of the art. Users can directly apply our approach to detect semantic anomalies in their own event data by using one of our pre-trained classifiers, even if their data contains so far unseen process behavior.

Keywords: Process mining · Anomaly detection · Natural language processing · Machine learning.

1 Introduction

Process mining analyzes data recorded during the execution of business processes in order to gain insights into an organization's operations [2]. A common task in this regard involves the detection of undesired process behavior, since such occurrences can, e.g., reveal compliance issues, operational inefficiencies, or recording errors. Such undesired behavior can be detected using conformance checking techniques [3], though only when a normative process model is available. Alternatively, techniques for anomaly detection [18] can be used to detect behavior that stands out in a statistical sense, i.e., because it is infrequent. However, infrequent behavior does not necessarily mean that it is undesired, since it could point to rare, but acceptable situations, whereas, conversely, behavior that is common is not necessarily desirable.

The recently introduced concept of semantic anomaly detection [1] overcomes these limitations by aiming to detect process behavior that stands out from a semantic point of view. It achieves this by considering the natural language labels associated with events, which allows it to recognize behavior that does not make logical sense, such as a claim being *paid* after it has been *rejected* or an order that is *updated* before it is *created*. Given that such semantic issues are often applicable across processes, semantic anomaly detection can exploit behavioral regularities extracted from existing resources, such as process model repositories. In this manner, undesired behavior can be detected, without the need to have a process model for the particular process at hand.

The problem that we address in this paper is that the state-of-the-art approach for semantic anomaly detection [1] is limited in terms of its accuracy and scope. In particular, it can only detect anomalies between pairs of activities that relate to the same business object and has limited generalization capabilities due to its rule-based nature.

Therefore, we propose an alternative approach that uses machine learning (ML) and state-of-the-art natural language processing (NLP) techniques. The core of our approach is formed by a classifier that we trained on data from a large process repository, covering a variety of domains. Our approach uses this classifier to detect *out-of-order* and *exclusion* anomalies, thus being able to recognize when two events should have been performed in a different order or should not have both been executed for the same case. We test classifier architectures that use classical ML techniques with word embeddings and deep learning techniques based on transformers. Our evaluation shows that both architectures greatly outperform the state of the art in terms of precision and recall, allowing our approach to detect a broader range of anomalies in a more accurate manner. Due to its demonstrated generalization capabilities, users can apply our approach directly on their event data, i.e., without requiring any training or labeled examples, even if their data contains process behavior that our classifiers have not seen before.

The remainder of this paper is structured as follows. Section 2 motivates the goal of semantic anomaly detection and highlights the limitations of the state of the art. Section 3 defines essential preliminaries. Section 4 presents our proposed ML-based approach, which we evaluate in Section 5. Finally, Section 6 discusses related work, whereas Section 7 concludes the paper.

2 Motivation

This section illustrates the potential of semantic anomaly detection, before describing the current state-of-the-art approach and its limitations.

Illustration. Consider the following two traces of a claims-handling process:

$t_1 = \langle$*receive claim, accept claim, check claim, pay compensation*$\rangle$
$t_2 = \langle$*receive claim, check claim, reject claim, pay compensation*$\rangle$

Without having any additional information about the process, the activity labels in these traces reveal two clearly undesirable process executions: in trace t_1, a claim was *accepted* before it had been *checked*, rather than the other way around, whereas in trace t_2, compensation was *paid*, even though the claim had been *rejected*.

Such examples demonstrate the potential of semantic anomaly detection based on the natural language of activity labels. This task is particularly easy for humans, who

can apply their commonsense and transferable knowledge to the process at hand in order to recognize that both traces show undesirable process behavior. To do this in an automated manner, however, requires an approach to learn such semantic relations between steps in a process, which is notably harder.

State of the art. The approach by Van der Aa et al. [1] tackles this challenge by establishing a *knowledge base* that captures rules about the semantics of appropriate process executions, extracted from a linguistic resource (*VerbOcean*) and a process model repository. Each rule captures a relation that should not be violated between two actions (i.e., verbs) applied to the same business object in a trace. For example, the knowledge base contains a rule that states that business objects should be *checked* before they can be *accepted*, which can be used to detect the anomaly in trace t_1.

Limitations of the state of the art. As the first approach of its kind, the approach by Van der Aa et al. [1] demonstrated the potential of semantic anomaly detection, yet also left considerable room for improvement with respect to its scope and accuracy. In particular, the approach proposed in our paper overcomes the following limitations:

Only intra-object anomalies. The existing approach can only detect anomalies involving two activities related to the same business object. In this manner, it can detect that *accept claim* should not come before *reject claim* in trace t_1, since both relate to a *claim* object. However, the approach cannot recognize that *pay compensation* should not follow *reject claim*, since these relate to different objects. By contrast, our proposed learning-based approach can detect both intra and inter-object anomalies.

Limited generalizability. The existing approach is limited in its ability to generalize information on process behavior. Specifically, it can learn individual rules, e.g., that *reject* and *accept* are mutually exclusive actions, and can generalize these to some degree by recognizing synonymous terms, e.g., that a claim can then also not be *rejected* and *approved* (a synonym of accept). However, this is as far as its generalization capabilities go, since the approach does not make connections between the rules it extracts, e.g., in order to recognize that in general *positive outcomes* (accept, approve, confirm, support, etc.) are exclusive to *negative outcomes* (reject, refuse, limit, etc.). By contrast, our proposed learning-based approach incorporates the information from all examples it encounters, to learn such broader behavioral regularities.

No context-specific anomalies. Finally, the existing approach treats all business objects in the same manner. However, many desirable or undesirable behavioral relations are context specific, meaning that they should or should not be allowed for certain business objects. For example, whereas in general it is fine that objects can be *changed* after they have been *created*, e.g., updating a created text document, this does not apply to objects such as a so-called *rush order* in SAP systems, which are special order types that are not allowed to be changed after creation, so that they can be safely processed immediately. Our proposed learning-based approach can make such context-specific distinctions, provided it receives training data that contains examples of when a certain behavioral regularity should and should not hold.

3 Preliminaries

Event model. Our work takes as input an event log L, which is a collection of traces. A trace $t = \langle e_1, e_2, .., e_n \rangle \in L$ is a sequence of events belonging to the same case of a process. Each event in a trace is associated with a textual label, indicating the activity to which it corresponds. Without loss of generality, we denote traces as sequences of their event labels when convenient, as e.g., shown for traces t_1 and t_2 in Section 2.

Process model. A process model defines desired execution dependencies between the activities of a process. For our purposes, it is sufficient to abstract from specific process modeling languages and focus on the behavior defined by a model. Therefore, we define a model M as a set of activity label sequences that lead the defined process from its start to its final state. We also define $M^F \subseteq M$ as the set of *loop-free* label sequences of M, i.e., the sequences that do not repeat any process fragments.

Eventually-follows relation. We use the eventually-follows relation $\prec$ to capture inter-relations between pairs of labels, stemming from recorded traces or allowed process model sequences. Given a trace $t = \langle e_1, e_2, .., e_n \rangle$, we use $e_i \prec_t e_j$ to denote that e_i occurs (directly or indirectly) before e_j in the trace. Similarly, $a_i \prec_M a_j$ holds if a process model M contains an execution sequence in which activity label a_i occurs before a_j, and $a_i \prec_{M^F} a_j$ holds if M contains a loop-free sequence with that relation.

Vector operations. We use $\mathbf{v} = \begin{bmatrix} v_1 \; v_2 \cdots v_n \end{bmatrix}$ to denote a numerical n-dimensional vector, with $v_i \in \mathbb{R}$ for $1 \leq i \leq n$. The average of two vectors, $\mathbf{v}$ and $\mathbf{w}$, is obtained by dividing the sum of the vectors by two, i.e., $(\mathbf{v} + \mathbf{w})/2$. Finally, we denote the concatenation of two vectors as $\begin{bmatrix} \mathbf{v} \; \mathbf{w} \end{bmatrix}$.

4 Approach

This section presents our proposed approach for semantic anomaly detection. As shown in Figure 1, our approach takes as input a trace t and consists of two main components. The first component, the *event-pair extractor*, takes a trace t and extracts a set of eventually-follows pairs of events P_t to be checked for anomalies. Then, the second component, the *anomaly detector*, takes each event pair $e_i \prec e_j \in P_t$ and uses a classifier to determine if events e_i and e_j should be able to follow each other in this particular order, i.e., whether or not this behavior is anomalous. We provide classifiers that we pre-trained on data stemming from a large process model repository, which means that users of our approach do not have to train their own classifier themselves. Based on such a classifier, our approach detects *ordering anomalies*, i.e., cases where e_j should have become before e_i rather than vice versa, e.g., *accept request* followed by *check request*, as well as *exclusion anomalies*, i.e., cases where e_j should not follow e_i because the two are mutually exclusive, e.g., *reject request* followed by *pay compensation*.

4.1 Event-pair Extractor

Our approach detects anomalies for pairs of events that are in an eventually-follows relation in a trace $t = \langle e_1, e_2, .., e_n \rangle$. We use this abstraction level, instead of a directly-follows relation, because semantic inter-relations between process steps often remain

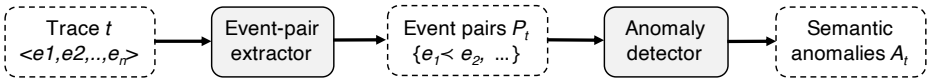


Fig. 1: Overview of our anomaly detection approach.

applicable even when other process steps occur in between them. For example, the notion that *receive request* should precede *accept request* holds true, irrespective of the occurrence of other steps in between, such as *check request* or *read request documents*.

However, when extracting eventually-follows pairs from a trace, it is important to consider the notion of rework, stemming from loops in a process. This is a crucial factor, because rework can have an important impact on the semantics of a process instance, particularly with respect to which events may or may not follow each other. For example, although *reject request* should normally not be followed by *pay compensation* (cf., trace t_2 in Section 2), this does not apply to trace t_3 shown in Figure 2. There, rework was conducted after initially rejecting the request, which made the subsequent payment of compensation acceptable.[1]

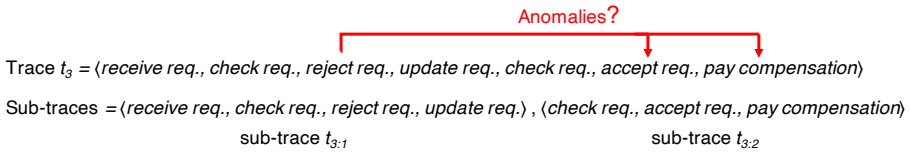


Fig. 2: Illustration of the necessity to identify rework in traces.

Therefore, to prevent the detection of incorrect anomalies, we only check for anomalous behavior within the same cycle of an instance's execution, by avoiding the comparison of behavior occurring in different loops through the process. To do this in the absence of a normative process model (which would render anomaly detection unnecessary), we detect rework at a trace level. Specifically, we split traces into sub-traces, by creating a new sub-trace each time we observe a label that was already present in the current sub-trace. For example, as shown in the lower part of Figure 2, trace t_3 is split into two sub-traces, where sub-trace $t_{3:2}$ starts when *check request* occurs for the second time. Alternatively, a process model can be discovered for the event log, in order to recognize loops in the process.

Given such sub-traces (a single one for traces without any repetition), we then extract the set of event pairs P_t so that it includes all pairs of events that are in an eventually-follows relation within an identified sub-trace. For sub-trace $t_{3:2}$, this yields *check request* $\prec$ *accept request*, *check request* $\prec$ *pay compensation*, *accept request* $\prec$

[1] Note that rework considerations would also apply to directly-follows relations, e.g., in ⟨.., *check*, *reject*, *check*, *accept* ⟩ we observe that *check [request]* (directly) follows *reject [request]*, rather than vice versa, yet that this is allowed due to rework being conducted.

pay compensation, whereas P_{t_3} comprises $6 + 3 = 9$ event pairs in total. By contrast, without splitting t_3 into its sub-traces, the set would comprise 21 pairs.

4.2 Anomaly Detector

In this component, we apply a classification model to determine for each event pair in the set P_t if it is anomalous or not. We train this classification model on data from a large process model repository, which can then be used when applying our approach on any event log. We test two model architectures for this: An architecture using Support Vector Machines (SVMs), a traditional machine learning technique, in combination with word embeddings, and a transformer-based architecture, a deep learning technique, using a fine-tuned BERT model.

Given that SVMs are simpler and faster to train, whereas transformers often gain better accuracy on complex problems, the comparison of the two architectures allows us to gain insights into the complexity of the problem (i.e., whether or not traditional machine learning suffices), as well as into the benefit of building on a pre-trained language model (i.e., BERT) and a more computationally-intensive method (i.e., deep learning using transformers).

Model architecture 1: SVM-based classification. Architecture 1 first transforms the natural language labels of an event pair $e_i \prec e_j \in P_t$ into a vector representation by using word embeddings. This vector is then fed into a trained SVM, which will return a classification, i.e., whether or not $e_i \prec e_j$ is an anomaly.

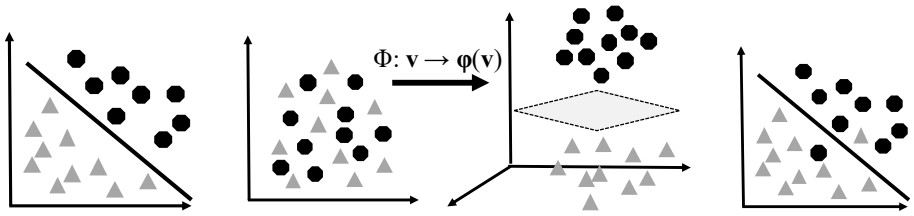
Vector representation using GloVe. Given that SVMs (as most machine learning techniques) require a numerical vector as input, we first turn an event pair $e_i \prec e_j$ into a vector representation $\mathbf{v}_{e_i,e_j}$ using GloVe representations [21].

GloVe (short for *Global Vectors*) is a static word representation technique that can be used to create an embedding for a given word, i.e., a vector representation of the word in a high-dimensional space. Such word embeddings are used to capture the meaning of words in a vector, by placing semantically similar words close to each other in the embedding space. Given an event label, e.g., *accept request*, we first use GloVe to establish an embedding of each word, resulting in two 300-dimensional vectors, e.g., $\mathbf{v}_{accept}$ and $\mathbf{v}_{request}$. Then, to obtain vectors of equal length, independent of the number of words in a label, we take the average of all word vectors of the event label, resulting in a single vector representation, e.g., $\mathbf{e_i} = (\mathbf{v}_{accept} + \mathbf{v}_{request})/2$. Finally, to encode an event pair, we concatenate the vectors of the two event labels, i.e., $\mathbf{v}_{e_i,e_j} = \left[\mathbf{e_i}\ \mathbf{e_j}\right]$, resulting in a vector of size 600, which accounts for the order in which e_i and e_j were observed (i.e., vectors $\mathbf{v}_{e_i,e_j}$ and $\mathbf{v}_{e_j,e_i}$ are different).

By using embeddings as input for text classification, a classifier can recognize event pairs that have a similar meaning, allowing it to generalize from its training data. For example, if the relation *check application* $\prec$ *approve application* is observed during training, a classifier can recognize that the relation *check request* $\prec$ *accept request* is semantically similar (i.e., has a similar vector representation), and thus recognize that this previously unseen behavioral relation is not an anomaly.

Support Vector Machine. We use the obtained vector representation $\mathbf{v}_{e_i,e_j}$ as input for a two-class SVM, which is a common technique for supervised (traditional) machine

learning on textual data [4]. As shown in Figure 3a, an SVM aims to establish a hyperplane that separates data points belonging to different classes in the feature space, in our case event pairs in the high-dimensional vector space obtained through embedding.



(a) Data separated by a linear hyperplane

(b) Use of a kernel function Φ to separate data

(c) Benefit of parameter C.

Fig. 3: Illustration of SVMs (based on [4]).

When training an SVM (the procedure is described below), we alter two primary settings: the applied kernel function Φ and the choice of the regularization parameter C. A *kernel function* Φ transforms the dimensionality of the data at hand, aiming to represent the data in a higher dimensionality that allows for better separation. Figure 3b shows an example in which data points are not separable in a two-dimensional space, but that can be clearly separated after applying a kernel function that transforms the data into a three-dimensional space. In our experiments, we test linear, polynomial, and Gaussian functions. Furthermore, we alter the *regularization parameter* C, which sets the degree of misclassification allowed when establishing a hyperplane. Particularly, as shown in Figure 3c, by allowing for a *soft margin* (corresponding to a low C value), some of the training data points may fall outside of the hyperplane, i.e., be misclassified. By allowing for this, the SVM can avoid overfitting to the training data.

Model architecture 2: BERT-based classification. Architecture 2 is a transformer-based architecture that uses a fine-tuned BERT model for anomaly detection.

Transformers and BERT. A *transformer* is a type of neural network architecture that uses self-attention mechanisms to process sequences of data, such as natural language sentences, and learn the relationships between different elements in the sequence [24]. BERT (short for Bidirectional Encoder Representations from Transformers), in turn, is a transformer-based language representation model [6] that has been shown to achieve excellent performance on a broad range of natural language processing tasks.

BERT learns to understand language by processing large amounts of text data (such as the entire English Wikipedia) in an unsupervised manner. This *pre-training* is performed using masked language modeling (Masked LM) and next sentence prediction (NSP). Masked LM trains the model to predict masked tokens based on the context of the surrounding tokens, which allows BERT to learn to represent words in the context of the entire sentence, rather than just based on their local context. In NSP, BERT is trained to predict whether two sentences are consecutive in the input text or not, which helps the model to learn about the relationships between sentences and the broader con-

text of the text. By pre-training on these tasks, BERT learns to represent words and sentences in a way that captures the semantic relationships between them, allowing it to understand natural language text and perform well on a wide range of downstream tasks, such as text classification, question answering, and named entity recognition.

BERT fine-tuning. To use BERT for semantic anomaly detection, we fine-tune a pre-trained BERT model on the task at hand. Fine-tuning has the benefit that the classification model can use the language understanding it obtained during pre-training, while requiring much fewer training samples and computation time than would be required when training such a model from scratch.

To perform fine-tuning, we extend BERT's architecture with an additional output layer for two-class classification (whether an input pair is anomalous or not) and then train it in a supervised manner on a collection of positive and negative training samples. Since BERT takes a sequence as input, we provide it event pairs in the following manner: $[[CLS]$, *receive, request,*$[SEP]$, *check, document, completeness,* $[SEP], [PAD], ..., [PAD]]$, where $[CLS]$ is a special token to indicate a classification task, $[SEP]$ is used to indicate the end of an event label, and $[PAD]$ is used to fill the input vector to its maximum length of 128 (since transformers process an entire input sequence of fixed length at once).

Model training. We train our SVM-based and BERT-based classification models on label pairs extracted from an available process model collection $\mathcal{M}$ (details on the data itself provided in Section 5.1). Given a process model $M \in \mathcal{M}$, we extract training samples in the form of allowed and anomalous label pairs based on the model's loop-free eventually-follows relation $\prec_{MF}$. Specifically, we first establish a set of positive label pairs P_M^+, which consists of all pairs of activity labels that can appear in an eventually-follows relation in model M, without any loops in the process. For the example model M_1 in Figure 4, this yields a set $P_{M_1}^+$ with eight eventually-follows relations, such as *receive request* $\prec^+$ *check request* and *check request* $\prec^+$ *pay compensation*.[2]

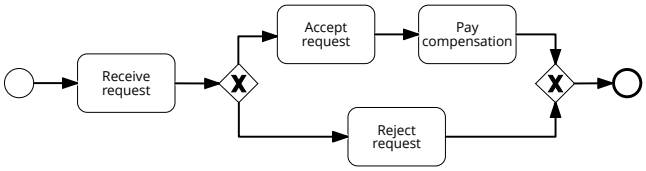


Fig. 4: Exemplary process model used as a basis for training samples.

Then, we establish a set of anomalous samples P_M^- consisting of label pairs not allowed in model M, i.e., that are not included in P_M^+. To provide a balanced training set, we populate P_M^- by randomly selecting pairs that are not in P_M^+, until we have an equal number of positive and negative samples. This would yield a set P_M^- that also consists of eight relations for the example from Figure 4, e.g., including *reject request* $\prec^-$ *pay compensation* and *accept request* $\prec^-$ *check request*.

[2] For clarity, we use $\prec^+$ and $\prec^-$ to denote positive and negative training samples, respectively.

Anomaly detection. Finally, we use a trained classification model to classify each event pair $e_i \prec e_j \in P_t$ as either anomalous or not, resulting in a set of anomalous relations $A_t \subseteq P_t$. Note that, before feeding a label pair into a classifier, we first sanitize the labels using a previously proposed tokenization technique [22], which deals with, e.g., underscores and camel case labels.

4.3 Approach Output

Our approach yields a set of detected anomalies A_t per trace. When presenting the results to a user, we recognize that a single issue in a process can lead to multiple anomalous label pairs. For example, a trace $t_4 = \langle$*accept claim, receive claim, check claim*$\rangle$ will yield two anomalous relations, i.e., $A_{t_4} = \{$*accept claim* $\prec$ *receive claim*, *accept claim* $\prec$ *check claim* $\}$, which both relate to the premature occurrence of *accept claim*. Furthermore, we verbalize the detected issues using a standard template in order to make them easier to interpret. For the exemplary trace t_4, this then yields the following output:

"Anomaly in t_4: *accept claim* occurred before *receive claim* and *check claim*."

Finally, we aggregate the anomalies detected for all traces in an event log L, resulting in a multi-set of identified issues and their respective frequencies.

5 Experimental Evaluation

This section reports on evaluation experiments conducted to assess the accuracy of our proposed approach, including its two model architectures, and compare it to the state-of-the-art rule-based approach. We describe the data collection Section 5.1 and the experimental setup in Section 5.2. In Section 5.3, we present the evaluation results demonstrating that our ML-based approach accurately detects semantic anomalies and greatly outperforms its rule-based competitor in this regard. Finally, Section 5.4 shows an application scenario in which we apply our approach on a real-world event log. The employed implementation, data collection, evaluation pipeline, and raw results are all available in our repository.[3]

5.1 Data Collection

To evaluate our approach, we require a data collection consisting of traces with known anomalies, or, more specifically, event pairs for which a gold standard classification as anomalous or not is available. Since there are no real-world event logs available that include such a gold standard, we instead obtain gold standard data from a large collection of real-world process models from the BPM Academic Initiative (BPMAI) [25].

Specifically, we selected those process models from BPMAI that are in the BPMN notation, have English labels, and that can be turned into a sound workflow net, yielding a total set $\mathcal{M}$ of 2,813 process models. This set comprises process models from a broad range of types and domains, including typical processes related to the handling of orders

[3] https://gitlab.uni-mannheim.de/processanalytics/ml-semantic-anomaly-dection

and requests, as well as more specialized processes, e.g., from software engineering and healthcare domains.

Train-test split. To evaluate our approach in an unbiased manner, we established a random split of the process model collection by dividing $\mathcal{M}$ into a training set, $\mathcal{M}_{train}$, comprising 70% (i.e., 1,969) of the models, and a test set, $\mathcal{M}_{test}$, containing the remaining 30% (844). The training set was used for model training, including hyper-parameter optimization (see Section 5.2), whereas the test set is exclusively reserved for assessing the performance of our approach. The train-test split is available on our repository.

Characteristics. For each model $M \in \mathcal{M}$, we establish equally-sized sets of (unique) normal and anomalous eventually-follows pairs, i.e., P_M^+ and P_M^-, using the method described in the *Model training* paragraph of Section 4.2, this means that 50% of the label pairs in the training and in the test are anomalies, whereas the rest corresponds to regular process behavior.

Table 1 shows the characteristics of the train and test set separately. It shows that nearly half of the label pairs in the test set (10,073) are not included in the training set. With so many unseen label pairs in the test set, a successful anomaly detection approach needs to be able to generalize well from the pairs that it observes during training, making the data collection highly suitable for our purpose.

Table 1 also reports on the number of label pairs that relate to the same business object (BO), such as *create order* $\prec$ *accept order*, since the rule-based state of the art [1] is restricted to such pairs.

Table 1: Characteristics of the data collection. The *Unseen* column refers to labels or label pairs that only occur in the test set, not the training set.

	Training set	**Test set**	
	Total	Total	Unseen
Process models	1,969	844	–
Unique labels	9,089	4,715	2,684
Total label pairs	53,598	23,770	10,073
Unique label pairs	43,483	21,934	9,906
Total label pairs (same BO)	3,724	1,714	723
Unique label pairs (same BO)	2,711	1,488	694

5.2 Experimental Setup

Implementation and environment. We implemented our approach in Python (see our repository) and conducted experiments on a machine with 768GB of RAM, an Intel Xeon 2.6 GHz CPU, and an Nvidia RTX 2080 Ti GPU (used to fine-tune BERT).

Hyper-parameter optimization. We conducted a hyper-parameter search to identify the most promising configuration for each of our model architectures.

For the SVM-based architecture, we tested different kernel functions, i.e., a linear, a polynomial, and a Gaussian radial basis function (RBF), various values for the degree D of the polynomial kernel function, i.e., $D \in \{2, 4, 6, 8, 10\}$, and different settings for the regularization parameter C, i.e, $C \in \{2^{-5}, 2^{-3}, 2^{-1}, 2, 2^3\}$. Moreover, we tested the effect of reducing the dimensionality of the embedding vectors using Principal Component Analysis (PCA) prior to training the SVM, because this can help reduce the time required for training. The results obtained after using PCA showed that this causes too much information loss, though, and thus leads to considerably worse performance. Therefore, we use the original vector size of 600, obtained by concatenating two 300-dimensional GloVe embeddings, one per label of a pair.

For the BERT-based architecture, we tested two base models for English: *bert-base-cased*, which is pre-trained on text that is case sensitive, and *bert-base-uncased*, which is pre-trained on all lower case text. Furthermore, we tested different learning rates (5e-5, 4e-5, 3e-5, 2e-5) and warm-up steps the model performs when fine-tuning (0, 500, and 1000 steps). We use 3 epochs for fine-tuning in order to avoid over-fitting [6].[4]

To select a configuration, we randomly split the models of the training set $\mathcal{M}_{train}$ into a 90% part that is used for the actual training and 10% that are used for validation. We then conducted a train-validation run per configuration, and selected the configuration that yielded the best results:

- *SVM*: an RBF kernel with a C-value of 2, and a vector size of 600.
- *BERT*: *bert-base-uncased* with a learning rate of 5e-5 and 500 warm-up steps.

We trained an SVM-based and a BERT-based classification model on the entire training set using these optimal configurations, which we use for our experiments on the test set and provide as pre-trained models to users of our approach in our repository.

Baseline. We compare our approach against the rule-based approach by Van der Aa et al. [1], of which the details are described in Section 2. It is important to note that this baseline can only detect anomalies for label pairs that share the same business object (as reported in Table 1); the baseline thus automatically classifies pairs with distinct BOs as non-anomalous. We compare our work against the configuration with the best results reported in the original paper, referred to as *SEM4* in their experiments. Most importantly, this configuration uses a semantic similarity threshold to improve the generalizability of the rules learned by the approach.

Measures. We measure the performance of our approach in terms of precision, recall, and F_1-scores, obtained by comparing the predicted classes of label pairs (i.e., anomaly or normal behavior) to the gold standard. Given a class $c \in \{Anomaly, Normal\}$, we denote the number of pairs correctly assigned to c as *tp*, the number of pairs that are incorrectly assigned to c as *fp*, and the number of pairs that belong to c in the gold standard, yet, are not assigned to c as *fn*. Precision (Prec.) is then defined as $tp/(tp+fp)$, recall (Rec.) as $tp/(tp+fn)$, and the F_1-score as the harmonic mean of the two.

[4] We provide detailed results of the hyper-parameter optimization in our repository.

5.3 Results

This section presents the results obtained through our evaluation experiments. We first focus on an in-depth analysis of the classification performance of our approach and the baseline, followed by a report on the training and inference times.

Overall results. Table 2 gives an overview of the main results of our experiments. It shows precision, recall, and F_1-scores per model architecture and the baseline, for different subsets of the event pairs included in the test set.

Overall, the SVM-based model achieves a reasonable F_1-score of 0.69 when considering the entire test set, which shows its general capability to distinguish semantic anomalies from normal behavior. The similar F_1-scores for the individual classes, i.e. 0.68 for the *Anomaly* and 0.70 for *Normal* class, indicate that the model's has learned to recognize anomalous behavior and normal behavior equally well.

Table 2: Results of the evaluation experiments obtained on the test set. Bold numbers indicate the best score for that particular row.

			SVM			**BERT**			**BL [1]**		
Scope	**Class**	**Support**	**Prec.**	**Rec.**	**F_1**	**Prec.**	**Rec.**	**F_1**	**Prec.**	**Rec.**	**F_1**
All pairs	Anomaly	11,885	0.70	0.66	0.68	**0.76**	**0.74**	**0.75**	0.69	0.01	0.01
	Normal	11,885	0.68	0.72	0.70	**0.75**	0.77	**0.76**	0.51	**0.99**	0.67
	Overall	23,770	0.69	0.69	0.69	**0.76**	**0.76**	**0.76**	0.60	0.50	0.54
All pairs w. same BO	Anomaly	857	0.73	0.71	0.72	**0.81**	**0.76**	**0.78**	0.69	0.07	0.12
	Normal	857	0.72	0.73	0.73	**0.77**	0.82	**0.80**	0.51	**0.97**	0.67
	Overall	1,714	0.72	0.72	0.72	**0.79**	**0.79**	**0.79**	0.60	0.52	0.56
Unseen pairs	Anomaly	5,009	0.64	0.64	**0.64**	0.62	**0.66**	**0.64**	**0.67**	0.01	0.01
	Normal	5,064	**0.64**	0.64	0.64	**0.64**	0.60	0.62	0.50	**0.99**	**0.67**
	Overall	10,073	**0.64**	**0.64**	**0.64**	0.63	0.63	0.63	0.58	0.50	0.54
Unseen pairs w. same BO	Anomaly	343	0.72	0.73	0.73	**0.79**	**0.76**	**0.77**	0.68	0.07	0.12
	Normal	380	0.75	0.75	0.75	**0.79**	0.82	**0.80**	0.54	**0.97**	0.69
	Overall	723	0.74	0.74	0.74	**0.79**	**0.79**	**0.79**	0.60	0.54	0.57

Our BERT-based model outperforms its SVM-based alternative in all aspects on the entire test set. It achieves a good overall F_1-score of 0.76, which shows that it accurately classifies unseen behavior into semantic anomalies and normal behavior. The better results compared to the SVM-based model suggest that the general language understanding of the transformer in combination with its process-specific fine-tuning improves the performance on our anomaly detection task. At the same time BERT's performance is also stable across classes, achieving comparable scores (0.74–0.77) for all metrics, for both the *Anomaly* and the *Normal* class.

Both our models greatly outperform the baseline (with the exception of recall on the *Normal* class), which achieves an overall F_1-score of 0.54, versus 0.69 and 0.76 of our models. Part of this difference occurs because the baseline, by definition, cannot detect

anomalies for event pairs with different business objects, which comprises about 93% of the total pairs. As a result, the baseline assigns the *Normal* class in the vast majority of cases, resulting in a low precision (0.50) but high recall (0.99) for that class, while achieving a recall of only 0.02 for the *Anomaly* class, with a precision of 0.64.

Same BO pairs. We also computed the results for the subset of label pairs that share the same business object (i.e., intra-object anomaly detection), this, among others, provides a fairer comparison to the baseline. We observe that—in line with expectations—the performance of the baseline improves for this subset, achieving an overall F_1-score of 0.56 compared to 0.54 on the full collection, caused by an increase in recall to 0.07 for the *Anomaly* class (versus 0.01 for the total collection. However, the baseline is still outperformed by both our models, which achieve overall F_1-scores of 0.72 (SVM) and 0.79 (BERT). The performance of the SVM-based model slightly improved from 0.69 overall F_1-score on the entire dataset to 0.72 on the subset of data, whereas the BERT-based model demonstrated larger gains, achieving an overall F_1-score of 0.79 compared to 0.76 for the full collection. We can thus note that even if we only consider data that the baseline is designed to handle, our approach still consistently achieves better results.

Unseen label pairs. To be able to assess how well our approach can deal with unseen behavior, we computed the results for the subset of label pairs that only occur in $\mathcal{M}_{test}$ and thus have not been observed by our models during training.

The results obtained for this subset show that both model architectures of our approach can generalize reasonably well to such unseen data, although it is clear that this anomaly detection task is more challenging. We observe that the performance of the BERT-based model drops to an F1-score of 0.63, from 0.76 for the entire test set, whereas the SVM-based model is more stable, achieving an F1-score of 0.64, compared to 0.69 for the entire set.

The main generalization capabilities of our approach become apparent when considering the detection of intra-object anomalies, i.e., by considering unseen label pairs with the same business object. For this subset, both model architectures perform equally well on unseen pairs as on the set including seen pairs, achieving F1-scores of 0.79 (BERT-based) and 0.74 (SVM-based). It should be noted that this subset is relatively small, though, consisting of 723 label pairs.

Overall, these results reveal that anomaly detection can be well-generalized to intra-object relations, e.g., by learning that objects should be *received* before they are *checked*, whereas it is more challenging to learn rules that also apply to unseen activities that relate to different business objects, e.g., just because an *order* must be created before a *delivery*, does not mean that this also applies to two unseen objects.

Post-hoc analysis. In order to gain deeper insights into the results, we go beyond a quantitative analysis and take a closer look at the correct and incorrect classifications of our approach and the baseline. Specifically, we focus on our BERT-based model, which has demonstrated better performance. We find that our approach is able to correctly identify a wide range of semantically problematic behaviors in the test set. For instance, it finds that *reject credit* should not happen before *assess risk*, that *wait for payment* should only happen after *create invoice*, and that *receive payment* should not be followed by *confirm order*. Note especially that none of these anomalies can be detected by the baseline, since the activities per pair refer to distinct business objects.

Looking at the baseline's results in detail, we observe that, even though it specifically targets the detection of semantic anomalies that involve the same business object, our approach finds additional, relevant intra-object anomalies that the baseline could not detect. For instance, our model correctly detects that *approve application* should not follow *cancel application* and that *evaluate request* should not happen before *prioritize request*, which the baseline does not find. Such cases illustrate the capability of our approach to better consider the meaning of entire activities, not just the actions applied to the same object, as done by the baseline.

However, there is also behavior that our approach fails to classify correctly. For instance, our approach detects *receive payment* followed by *pick shipment* as an anomaly, whereas it is well-imaginable that for some order handling process a shipment is indeed only sent after payment for that shipment was collected. Conversely, our approach did not find that, e.g., *send loan request* followed by *fill out loan request* may be problematic, for instance, if these are executed by the same resource. Our approach currently does not consider such context-dependent anomalies, which would require the incorporation of resource information, a direction for future research.

Computation time. Table 3 depicts training and inference times of our models and the baseline. The training time refers to the time it takes to train a model using the pairs in $\mathcal{M}_{train}$. The duration refers to the actual training, thus excluding the time it takes to load process models and establish training pairs (which is the same for all approaches).

Table 3: Average training and inference times of our models and the baseline.

	SVM	**BERT**	**BL [1]**
Training time	23.4s	1,859.8s	2.8s
Inference time per label pair	0.01s	0.01s	0.03s

We find that our SVM-based model requires a training time of 23 seconds, whereas the BERT-based model requires 1,859 seconds (~31 minutes) for fine-tuning in 3 epochs. The knowledge base population of the baseline only takes about 3 seconds, since it just performs a single pass over the label pairs, storing their counts. As such, there is a trade-off between lower training times and optimized performance. Nonetheless, the performance gain is so strong that we give a clear recommendation for using the BERT-based model. This is especially the case because users do not need to train our approach themselves, but can directly use the fine-tuned model provided in our repository.

With respect to inference time, both our models and the baseline are fast, classifying label pair in less than 0.03 seconds on average.

5.4 Real-world Application

Finally, we also applied our approach on real-world data: the *permit log* from the BPI 2020 challenge [9], which captures data on work trips conducted by university employees. The process flow concerns the request for and approval of a travel permit, the trip itself, a subsequent travel declaration, as well as associated reimbursements.

Although there is no gold standard available that indicates true anomalies in this process, our approach (using the BERT-based model) detects various interesting situations, as shown in Table 4. The examples correspond to situations in which trips happened before a permit was properly handled or approved (*a1* and *a2*), declarations submitted before a trip rather than after (*a3*), as well as payments that were approved before the respective permit was (*a4*). Still, we also recognize that certain detected anomalies look concerning, but are fine in light of the specifics of the process. This applies to anomaly *a5*, which corresponds to payments occurring before a declaration was actually approved. Although this seems problematic, it is possible in this process for payments related to pre-paid expenses.[5]

ID	Detected anomaly	Frequency
a1	*end trip* occurred before *permit final approved by supervisor*	2,800
a2	*start trip* occurred before *permit submitted by employee*	2,205
a3	*declaration submitted by employee* occurred before *start trip*	4,707
a4	*request for payment approved by administration* occurred before *permit approved by supervisor*	611
a5	*declaration final approved by supervisor* occurred after *request payment* and *payment handled*	4,292

Table 4: A selection of anomalies detected for the real-world *permit log*.

6 Related Work

Various approaches for anomaly detection in process mining have been proposed. Most of these are frequency-based, arguing that uncommon behavior is not of interest or undesirable, as opposed to our semantic approach. Such frequency-based anomaly detection is an inherent part of certain process discovery algorithms [14], which use it to preserve only the most common process behavior. Close to our approach are ML-based techniques, such as works that use autoencoders [13,19], as well as LSTMs (long short-term memory) [12], working in unsupervised or semi-supervised manners. Whereas most approaches only consider control-flow information, others also incorporate additional perspectives, such as BINet [18] for deep learning-based anomaly detection detection, and pattern-based techniques employed in the context of filtering in the process discovery [16] and the repair of event log imperfections [8]

Our work also relates to other NLP applications that focus on distinguishing normal from abnormal relations, primarily in the form of *commonsense reasoning*. Beyond using ML-based techniques, such reasoning can be done based on, e.g., lexical resources, such as WordNet [17] or VerbOcean [5], which capture relations between words, or commonsense knowledge graphs [11, 20], which capture common relations between entities. Such reasoning is, for example, employed to improve the quality of actions and state changes extracted from natural language texts [15, 23].

[5] Note that such false positives would be avoided when using an object-centric event log, since there would be no relation between the events related to pre-payments and declarations.

7 Conclusion

In this paper, we proposed an ML-based approach for the detection of semantic anomalies in business processes, allowing users to detect undesired behavior without depending on the availability of a normative process model. By building on state-of-the-art NLP techniques to train an anomaly classifier, our approach has learned to distinguish normal from undesired process behavior based on the textual labels associated with recorded events. Our experiments demonstrate that our learning-based approach greatly outperforms an earlier, rule-based approach for semantic anomaly detection in terms of both scope and accuracy.

Still, our work is subject to limitations. Our approach itself is limited by its focus on event pairs. Although this perspective is chosen because it allows us to achieve accurate results and fine-granular anomaly insights (i.e., much more specific than detecting whether or not a trace is anomalous), the event-pair perspective does not allow us to detect missing behavior, e.g., that *check request* was skipped (unlike the baseline approach [1]). Also, the performance of our approach depends on the similarity of behavioral regularities observed during training and those in the event log on which it is applied. Positive points in this regard are that we trained our approach on data from a broad range of domains and that we have demonstrated its capabilities to generalize to unseen data, especially when it comes to intra-object anomalies. Furthermore, in the absence of real-world logs with known anomalies, our experiments are conducted using generated samples. However, these samples are established based on real-world process models, whereas we also show the potential of our work in a real-world application.

In future work, we aim to lift the concept of ML-based semantic anomaly detection to the recent wave of generative large language models, such as ChatGPT and GPT4, once such technology becomes freely accessible, so that experiments can be conducted in a reproducible manner. Here, we would also like to stress that our conceptual idea is independent of a specific language model, so that the same approach can later be updated according to new developments on the NLP side. Furthermore, we also aim to incorporate additional perspectives into the detection of anomalies. Particularly, we aim to encode resource roles and categorical attribute values, allowing our approach to, e.g., consider who performed a certain step and what the outcome of a decision was. Finally, in terms of application scenarios, we plan to integrate our work into analysis pipelines in which semantic correctness plays an important role, such as in the privatization of event data, where the insertion of obvious noise should be avoided [10], and in next activity prediction, where predicted next steps should make semantic sense [7].

Reproducibility: Our employed implementation, data, and obtained results are available through the project repository linked in Section 5.

References

1. van der Aa, H., Rebmann, A., Leopold, H.: Natural language-based detection of semantic execution anomalies in event logs. Information Systems **102**, 101824 (2021)
2. van der Aalst, W.M.P.: Process mining: Data science in action, vol. 2. Springer (2016)
3. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance checking. Springer (2018)

4. Chauhan, V.K., Dahiya, K., Sharma, A.: Problem formulations and solvers in linear SVM: a review. Artificial Intelligence Review **52**(2), 803–855 (2019)
5. Chklovski, T., Pantel, P.: Verbocean: Mining the web for fine-grained semantic verb relations. In: EMNLP. pp. 33–40 (2004)
6. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: NAACL. pp. 4171–4186. ACL, Minneapolis, Minnesota (2019)
7. Di Francescomarino, C., Ghidini, C.: Predictive process monitoring. Process Mining Handbook. LNBIP **448**, 320–346 (2022)
8. Dixit, P.M., Suriadi, S., Andrews, R., Wynn, M.T., ter Hofstede, A.H., Buijs, J.C., van der Aalst, W.M.: Detection and interactive repair of event ordering imperfection in process logs. In: CAISE. pp. 274–290. Springer (2018)
9. van Dongen, B.: BPI challenge 2020 (2020). https://doi.org/10.4121/UUID:52FB97D4-4588-43C9-9D04-3604D4613B51
10. Fahrenkrog-Petersen, S.A., Kabierski, M., van der Aa, H., Weidlich, M.: Semantics-aware mechanisms for control-flow anonymization in process mining. Information Systems p. 102169 (2023)
11. Havasi, C., Speer, R., Alonso, J.: ConceptNet 3: a flexible, multilingual semantic network for common sense knowledge. In: RANLP. pp. 27–29. John Benjamins Philadelphia, PA (2007)
12. Krajsic, P., Franczyk, B.: Semi-supervised anomaly detection in business process event data using self-attention based classification. Procedia Computer Science **192**, 39–48 (2021)
13. Krajsic, P., Franczyk, B.: Variational autoencoder for anomaly detection in event data in online process mining. In: ICEIS (1). pp. 567–574 (2021)
14. Leemans, S.J., Fahland, D., Van Der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: BPM Workshops. pp. 66–78. Springer (2014)
15. Losing, V., Fischer, L., Deigmoeller, J.: Extraction of common-sense relations from procedural task instructions using BERT. In: 11th Global Wordnet Conference. pp. 81–90 (2021)
16. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.: Data-driven process discovery-revealing conditional infrequent behavior from event logs. In: CAISE. pp. 545–560. Springer (2017)
17. Miller, G.A.: WordNet: a lexical database for english. Communications of the ACM **38**(11), 39–41 (1995)
18. Nolle, T., Luettgen, S., Seeliger, A., Mühlhäuser, M.: Binet: Multi-perspective business process anomaly classification. Information Systems p. 101458 (2019)
19. Nolle, T., Luettgen, S., Seeliger, A., Mühlhäuser, M.: Analyzing business process anomalies using autoencoders. Machine Learning **107**(11), 1875–1893 (apr 2018)
20. Omeliyanenko, J., Zehe, A., Hettinger, L., Hotho, A.: LM4KG: Improving common sense knowledge graphs with language models. In: ISWC. pp. 456–473. Springer (2020)
21. Pennington, J., Socher, R., Manning, C.: GloVe: Global vectors for word representation. In: EMNLP. pp. 1532–1543. ACL, Doha, Qatar (2014)
22. Rebmann, A., van der Aa, H.: Enabling semantics-aware process mining through the automatic annotation of event logs. Information Systems **110**, 102111 (2022)
23. Tandon, N., Dalvi, B., Grus, J., Yih, W.t., Bosselut, A., Clark, P.: Reasoning about actions and state changes by injecting commonsense knowledge. In: EMNLP. pp. 57–66 (2018)
24. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. NeurIPS **30** (2017)
25. Weske, M., Decker, G., Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Model Collection of the Business Process Management Academic Initiative (2020)