# Predictive Analytics for Object-centric Processes: Do Graph Neural Networks Really Help?

Riccardo Galanti[1,2] and Massimiliano de Leoni[2]

[1] IBM, Reggio Emilia, Italy riccardo.galanti@ibm.com
[2] University of Padua, Padua, Italy deleoni@math.unipd.it

**Abstract.** The object-centric process paradigm is increasingly gaining popularity in academia and industry. According to this paradigm, the process delineates through the parallel execution of different execution flows, each referring to a different object involved in the process. Object interaction is present, and takes place through bridging events where these parallel executions synchronize and exchange data. However, the complex intricacy of instances of such processes relating to each other via many-to-many associations makes a direct application of predictive process analytics approaches designed for single-id event logs impossible. This paper reports on the experience of comparing the predictions of two techniques based on gradient boosting or the Long Short-Term Memory (LSTM) network against two based on graph neural networks. The four techniques were empirically evaluated on event logs related to two real object-centric processes, and more than 20 different KPI definitions. The results show that graph-based neural networks generally perform worse than techniques based on Gradient Boosting. Considering that graph-based neural networks have training times that are 8-10 times larger, the conclusion is that their use does not seem to be justified.

**Keywords:** Predictive Analytics · Object-centric Processes · Machine Learning · Graph Neural Networks · Gradient Boosting

## 1 Introduction

Predictive process analytics is the branch of process mining that aims to predict the eventual outcome of process executions. Traditional predictive process analytics techniques rely on the assumption that process instances are composed of single flows of execution. However, recent industrial experience is showing that the assumption of a single execution flow is unfortunately often not met in practice. This led to the introduction of the paradigm of object-centric processes, which has recently been gaining more and more attention because it can naturally model inter-organizational processes more naturally [1]. Any process execution materializes itself as a set of instances that run concurrently, each representing the life cycle of one different object that contributes to the process execution (e.g., the order and the delivery object). These object life cycles run independently and synchronize through bridging events to exchange data required for further processing.

The problem of predictive analytics remains relevant in the context of object-centric processes, as well. Typically, the process outcome is measured using a Key Performance Indicator (KPI), which depends on and is accordingly configured for the specific process being analyzed. The existing techniques for predictive process analytics cannot be

directly applied in the context of object-centric processes because they heavily rely on the concept of a single identifier associated with a single execution flow.

This paper compares the predictive quality of two techniques leveraging on graph-based neural networks with one technique based on LSTM network and with one based on gradient boosting on decision trees. Graph-based neural network can naturally represent the complex many-to-many interaction between objects of object-centric processes. Conversely, LSTM networks and gradient boosting require the manual engineering of features that maintain a meaningful abstraction of the object interactions.

Graph-based neural networks have the significant disadvantage of a very large training time. The research question addressed in this paper is the following: *is the very large training time justified by a significant improvement of the prediction accuracy?* To answer this question, we conducted experiments with two object-centric processes and 21 different KPIs of interest, using the four predictive-analytics techniques mentioned above.

The results show that Gradient Boosting usually has the highest accuracy, compared with both LSTM and the two types of graph-based neural networks used in the experiments. At the same time, the training time is 8-10 times shorter. In sum, it seems that there is no advantage to use graph-based neural networks: a meaningful, manual engineering of features that encode the object interaction allows gradient boosting to reach higher prediction accuracy.

## 2 Preliminaries

### 2.1 Object-Centric Event Logs

Object-centric processes are executed with the support of one or more information systems. It is possible to extract the history of past executions from information systems into a transactional data set organized as object-centric event logs.

Space limitations prevent us from giving a full formalization, which can be found in [4]. Here, we limit ourselves to give the intuition through the example in Table 1, which shows an excerpt of an object-centric event log of an Italian utility-provider company. It consists of a set $E$ of event identifiers (see column *ID*), each associated with an activity name, a timestamp of occurrence, a set of object identifiers of different types associated with the event (columns from *Contract* to *Invoice*), and a set of event attributes with their associated values (columns from *Order_Price* to *Rec_Quantity*).

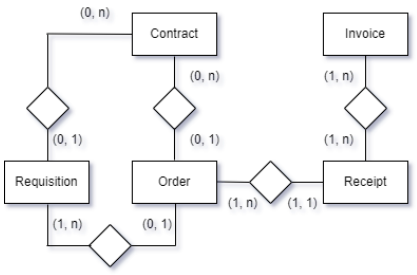
In particular, five object types can be observed, each with a different life cycle: *Contract*, *Requisition*, *Order*, *Receipt*, and *Invoice*. As an example, the event with identifier $e1$ is associated with object $c1$ of type *Contract*, whereas, e.g., event $e20$ is associated with object $i2$ of type *Invoice*, and the three objects $r2$, $r3$, and $r4$ of type *Receipt*.

The object-centric event log can map the relationships between object types. For instance, the contracts $c1$ and $c2$ are associated with the requisitions $rq1$ and $rq2$, respectively (cf. events $e3$ and $e5$). In general, a contract can be associated with multiple requisitions, while each requisition is at most associated with one contract. However, this can not be seen in the event log excerpt because of its small size.

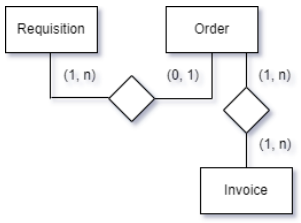
The life-cycle process of an object of type *Contract* refers to the stipulation of a contract with a customer, possibly followed by a *Requisition*, which is an optional object activated with its life cycle when the order needs a purchase requisition. The *Order*

Table 1: Example of an object-centric event log. Each row is an event, and the blank spaces represent attributes' missing values.

| ID | Activity | Timestamp | Contract | Requisition | Order | Receipt | Invoice | User | Order Price | Order Month | Order Group | Rec Quantity |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| e1 | Contract Line Creation | 2017-07-11 9:00 | c1 | | | | | CO01 | | | | |
| e2 | Contract Material Group Changed | 2017-07-14 11:00 | c1 | | | | | CO01 | | | | |
| e3 | Purchase Requisition Line Created | 2017-07-15 12:00 | c1 | rq1 | | | | A456 | | | | |
| e4 | Contract Line Creation | 2017-07-15 14:00 | c2 | | | | | CO01 | | | | |
| e5 | Purchase Requisition Line Created | 2017-07-15 17:00 | c2 | rq2 | | | | A457 | | | | |
| e6 | Purchase Order Line Creation | 2017-07-16 15:00 | c1 | | o1 | | | A458 | 100 | 7 | 100_L50 | |
| e7 | Contract Line Creation | 2017-07-16 16:00 | c3 | | | | | CO01 | | | | |
| e8 | Purchase Order Line Creation | 2017-07-17 15:00 | | rq1 | o2 | | | A458 | 200 | 8 | 100_L51 | |
| e9 | Purchase Order Line Creation | 2017-07-18 15:00 | | rq2 | o3 | | | A458 | 300 | 8 | 100_L52 | |
| e10 | Goods Line Registered | 2017-07-22 15:00 | | | o1 | r1 | | A456 | 100 | 7 | 100_L50 | 10 |
| e11 | Invoice Receipt | 2017-07-22 16:00 | | | | | i1 | A125 | | | | |
| e12 | Requisition Group Changed | 2017-07-22 19:00 | | rq1 | | | | A456 | | | | |
| e13 | Purchase Order Line Creation | 2017-07-23 9:00 | | rq1 | o4 | | | A458 | 600 | 8 | 100_L51 | |
| e14 | Purchase Order Line Creation | 2017-07-23 12:00 | c3 | | o5 | | | A458 | 600 | 8 | 100_L51 | |
| e15 | Goods Line Registered | 2017-07-23 15:00 | | | o2 | r2 | | A456 | 100 | 8 | 100_L50 | 10 |
| e16 | Invoice Registered | 2017-07-29 11:00 | | | | r1,r2 | i1 | A125 | | | | 10 |
| e17 | Invoice Cleared | 2017-07-30 12:00 | | | | | i1 | A125 | | | | |
| e18 | Goods Line Registered | 2017-07-31 15:00 | | | o4 | r3 | | A456 | 600 | 8 | 100_L51 | 10 |
| e19 | Goods Line Registered | 2017-08-09 15:00 | | | o5 | r4 | | A456 | 600 | 8 | 100_L51 | 10 |
| e20 | Invoice Registered | 2017-08-10 11:00 | | | | r2,r3,r4 | i2 | A125 | | | | 10 |
| e21 | Invoice Cleared | 2017-08-15 14:00 | | | | | i2 | A125 | | | | |
| e22 | Goods Line Registered | 2017-08-16 15:00 | | | o3 | r5 | | A456 | 300 | 8 | 100_L52 | 5 |
| e23 | Requisition Supplier Changed | 2017-08-16 17:00 | | rq2 | | | | A456 | | | | |
| e24 | Invoice Registered | 2017-08-18 11:00 | | | | r5 | i3 | A125 | | | | 5 |
| e25 | Invoice Cleared | 2017-08-20 14:00 | | | | | i3 | A125 | | | | |




(a) Energy event log.




(b) IT event log.

Fig. 1: ER-diagram representing the cardinality between the different object types in the two considered object-centric event logs. For each object type, the cardinality with the subsequent or the previous object type is represented as (*min_cardinality*, *max_cardinality*).

life-cycle process consists of several activities representing mainly quantity, price, or date modifications of the order, eventually approved by the head of the department. The *Receipt* life-cycle process is then related to receiving the goods or services requested, followed by the *Invoice* life-cycle process, which includes everything related to payments. Some events are associated with a single object identifier. In contrast, others are associated with multiple object identifiers (i.e., so-called bridge events), enabling the synchronization and data exchange between the object's life-cycle processes. Fig. 1 illustrates how objects are related to each other for synchronization and data exchanges for the aforementioned utility-provider company (see Fig. 1a) and for our second object-

centric event log related to an American technology company (see Fig. 1b). Note that relationships can be of many-to-many or many-to-one nature.

### 2.2 Single-Id Event Logs and Predictive Process Analytics

The traditional predictive process analytics assumes that a trace is naturally composed by a sequence of events, namely a trace $\sigma \in \mathcal{E}^*$ where $\mathcal{E}$ is the universe of events. An event $e \in \mathcal{E}$ records the execution of an activity $\pi_{act}(e)$ that occurred at time $\pi_{time}(e)$. Events also assign values to attributes: $\pi_{vmap}(e)$ returns a function $f$ such that $f(a)$ indicates that $e$ assigns value $f(a)$ to attribute $a$.

Process predictive analytics aims to predict the KPI value of traces $\sigma \in \mathcal{E}^*$. The definition of KPI depends on the process domain, and hereafter it is abstracted as a function:

**Definition 1 (KPI Function).** *Let $\mathcal{W}_K$ be the set of possible KPI values. A KPI is a function $\mathcal{T}_L : \mathcal{E}^* \times \mathbb{N} \not\to \mathcal{W}_K$ such that, given a trace $\sigma \in \mathcal{E}^*$ and an integer index $i \leq |\sigma|$, $\mathcal{T}_L(\sigma, i)$ returns the KPI value of $\sigma$ after the occurrence of the first $i$ events.*[3]

Note that our KPI definition assumes it to be computed a posteriori when the execution is completed and leaves a complete trail as a certain trace $\sigma$. In many cases, the KPI value is updated after each activity execution, namely after the occurrence of a subsequence event. We can then define the prediction problem on single-id, traditional event logs:

**Definition 2 (Prediction Problem on Single-id Event Logs).** *Let $\mathcal{T}_L$ be a KPI function. Let $\sigma = \langle e_1, \ldots, e_k \rangle$ be the trace of a running case, which eventually will complete as $\sigma_T = \langle e_1, \ldots, e_k, e_{k+1} \ldots, e_n \rangle$. The prediction problem can be formulated as predicting the value of $\mathcal{T}(\sigma_T, i)$ for all $k < i \leq n$.*

In the Process Mining literature, this problem has been faced with different Machine Learning models [9]. The training set is composed of pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$ where $\mathcal{X}$ encodes the independent variables (also known as **features**) with their values, and $\mathcal{Y}$ is the dependent variable with its value (i.e., the value to predict). Predictive process analytics requires a KPI definition $\overline{\mathcal{T}}$ as input (cf. Definition 1). Let $\mathcal{W}_K = img(\overline{\mathcal{T}})$ be the domain of possible KPI values (i.e., the image or co-domain of $\overline{\mathcal{T}}$: $\mathcal{Y} = \mathcal{W}_K$). Afterward, each prediction technique requires the definition of the domain $\mathcal{X}$ and a **trace-to-instance encoding function** $\rho : \mathcal{E}^* \to \mathcal{X}$, which maps each trace $\sigma$ or prefix of it to an element $\rho(\sigma) \in \mathcal{X}$.

The prediction model is then trained off-line based on a data set $\mathcal{D}$ that is created from an event log $L$ as follows: Each prefix $\sigma$ of each trace $\sigma_T \in L$ generates one distinct item in $\mathcal{D}$, consisting of a pair $(x, y) \in (\mathcal{X} \times \mathcal{Y})$, where $x = \rho(\sigma)$ and $y = \overline{\mathcal{T}}(\sigma_T, |\sigma|)$. Once the data set item of every trace prefix is created, the model is trained. The resulting prediction model (known as predictor) can be abstracted as an oracle function $\Phi_{\mathcal{D}} : \mathcal{X} \to \mathcal{Y}$.

## 3 Techniques for Object-centric Process Predictive Analytics

The application of object-centric predictive analytics techniques requires to build a set of graph instances. Initially, a single graph is built, where the nodes are the events of the

[3] Given a sequence $X$, $|X|$ indicates the length of $X$.

object-centric event log. The arcs are then added: an arc is added between a node/event $e'$ and a node/event $e''$ if $e'$ and $e''$ have at least one object in common. For instance, there is an arc between the events $e_1$ and $e_2$ in Table 1 because they share the object with identifier $c1$. Arcs are directed: the arc goes out $e'$ and enters $e''$ if the timestamp of $e'$ is smaller than the timestamp of $e''$.[4]

This graph is in fact not strongly connected: not every object is connected to every other object. The graph is thus partitioned into its strongly-connected components: each component becomes a graph instance.

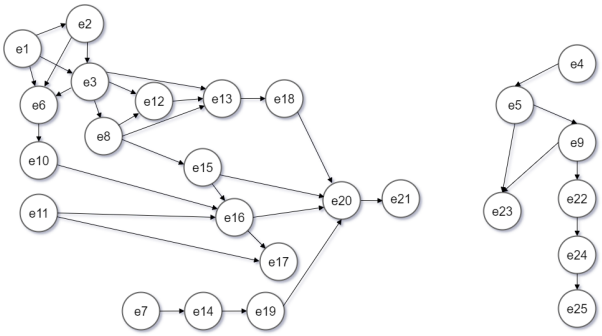



Fig. 2: The event graph extracted from the object-centric event log in Table 1. Each strongly-connected component becomes a graph instance, used to train predictors.

As an example, let us consider the object-centric event log in Table 1. Fig. 2 represents the two graph instances, namely the two strongly-connected components that have been found. For instance, the nodes in the first connected component represented by the event identifiers $e1$, $e2$, $e3$, and $e6$ are connected to each other as the object identifier $c1$ is in common.

In this paper, we consider four techniques for object-centric process predictive analytics, which require a previous construction of the set of graph instances: GCN (Graph Convolutional Network), GGNN (Gated Graph Neural Network), LSTM, and Catboost. While Catboost and LSTM require the flattening of the graph instances, GGNN and GCN can directly take the set of graph instances as input during the training and test phase. The choice fell for Catboost and LSTM because they are seen to generally outperform other methods for predictive analytics [11,12,5]. The remainder introduces how the graphs are encoded when Catboost and LSTM models are used as well as via two types of graph-based neural networks.

***Catboost and LSTM Models***. *LSTM* is a type of Recurrent Neural Network that uses gates to control the information flow over time. *Catboost* performs Gradient Boosting on Decision Trees [3]. Catboost performs at each iteration $t$ a random permutation of the features and creates a tree based on it. The usage for predictive process analytics has firstly been reported in [5].

Both LSTM and Catboost require flattening: each graph instance is converted into a trace of a single-id event log (cf. Section 2.2). Given a graph instance $g$, the corresponding trace contains every event of $g$ (recall that the $g$'s nodes are events). To retain aggregated information about the interaction, each event, node of the graph instance, is extended with attributes (i.e., features) that summarizes the interaction. This is extensively discussed in [6], and here we limit to give an intuition. For each object type $o$ and event $e$, we include an attribute that stores the number of events of type $o$ to which $e$ is

[4] The special case of an arc between two events with the same timestamp can be dealt separately: in this case, the arc is bi-directional

connected in $g$. Also, for each numerical attribute $a$ of every event connected to $e$, we compute the average value of $a$ in the events connected to $e$.

The result is ultimately a single-id event log, and the prediction problem is as formulated in Definition 2 (cf. Section 2.2). The difference between LSTM and Catboost is related to a different definition of trace-to-instance encoding function $\rho : \mathcal{E}^* \to \mathcal{X}$.

In the Catboost learning domain, $\mathcal{X}$ is a vector that contains one dimension for each process activity, and one dimension for each log attribute. Given a trace $\sigma$ defined over a set $A$ of activities with a set $V$ of event attributes, the encoding function is as follows:[5]

$$\rho_{Cat}(\langle e_1, \ldots, e_n \rangle) = \bigoplus_{a \in A} |\{e \in \sigma.\ \pi_{act}(e) = a\}| \oplus \zeta(e_n)$$

where $\oplus$ denotes the concatenation of two tuples and $\zeta(e)$ is the vector encoding of $e \in \sigma$:

$$\zeta(e) = \bigoplus_{v \in V} [\pi_{vmap}(e)(v)] \tag{1}$$

In the case of LSTM, $\mathcal{X}$ consists of sequences of vectors with $n$ dimensions, where $n$ is the number of event-log attributes: $\mathcal{X} = (\mathbb{R}^n)^*$.[6] Function $\rho$ is then defined as $\rho_{LSTM}(\langle e_1, \ldots, e_m \rangle) = [\zeta(e_1), \ldots, \zeta(e_m)]$ where function $\zeta_\sigma$ is as introduced in Definition 1. Further details of the encoding for Catboost and LSTM are provided in [5].

***Graph-based Neural Network Models***. *GCN* is designed to work on graph data [7]. Each log trace is represented as a graph, and we opted to represent the input graph as proposed in the work of [14]. Here, events of a prefix are represented as graph nodes.

*GGNN* integrates a Gated Recurrent Unit (GRU) cell that explicitly considers the temporal aspect of sequences [8]. We decided to represent the input graph as proposed in the work of [14]. Here, events of a prefix are represented as graph nodes, and edges are used to express relationships between the events of a prefix.

In the case of the GCN, $\mathcal{X}$ is represented by a two-element tuple $(AM, V)$; $AM$ is an adjacency matrix storing which nodes (that in our case represent events) of the graph are connected by an edge and lies in $\mathbb{R}^{|V| \times |V|}$, while $V$ is a node matrix storing features that describe the graph's nodes and lays in $\mathbb{R}^{|V| \times q}$, where $q$ is the number of node features. Please notice that, for each event, i.e. node, of the GCN and GGNN, the node features have been encoded in the same way as LSTM.

In the case of the GGNN, $\mathcal{X}$ represents a three-element tuple $(AM, V, EM)$. $EM$ is an edge matrix that is added in order to store features that describe the edges of the graph and lays in $\mathbb{R}^{|V| \times p}$, where $p$ refers to the number of features describing the edge. In particular, the edge between two events $e'$ and $e''$ encodes the object in common between those of $e'$ and $e''$. Edges are also characterized by a type: (1) *Repeat* (activity of a target event is equal to an activity of a source event), (2) *Backward* (activity of a target event was observed in a previous event of the current prefix), and (3) *Forward* (activity of a target event was not observed in previous events of the current prefix).

[5] To keep the explanation simple, we assume that the enumerations of all attributes $v \in V$ and all activities $a \in A$ are always returned consistently as if there is a total order among the variables and among activities (e.g., the alphabetical order).

[6] In literature, LSTMs are often trained based on matrices. However, a sequence of $m$ vectors in $\mathbb{R}^n$ can be seen, in fact, as a matrix in $\mathbb{R}^{n \times m}$. We use here the data set representation as vectors to simplify the formalization.

## 4 Evaluation Setup

The evaluation is based on the two object-centric processes described in Section 2.[7] The first object-centric process was executed by a well-known Italian utility-provider company, one of the major energy companies in Europe. The company focuses on the production/extraction of electricity and gas and on their distribution in different parts of the world. As mentioned in Section 2, this object-centric process runs through the intertwining of five different processes (i.e., object types): the *Requisition*, the *Order*, the *Invoice*, the *Contract*, and the *Receipt*.

The second object-centric process was executed by a well-known American technology company, one of the major companies worldwide. As mentioned in Section 2, this object-centric process runs through the intertwining of three different processes (i.e., object types): the *Requisition*, the *Order*, and the *Invoice*.

In a preprocessing phase, we removed attributes with missing values in more than 80% of the cases or attributes with the same values in all cases, and one of each pair of duplicate attributes (e.g., we removed the plant name, which is unique, and kept the plant identifier. Third, the large dimension of both companies is also reflected in the cardinality of some categorical attributes. For instance, for the utility-provider company, the codes of the materials shipped worldwide (*order_material_code*) are stored in an attribute that counts up to 4179 different values. We applied the 80-20 rule to reduce the cardinality of the attributes with thousands of different values [10]. Specifically, we kept the most frequent attribute values that covered 80% of the cases and labeled the remaining values as `other`. We considered several KPIs in our evaluation, grouped into three categories:

**Elapsed Time between the first occurrence of the considered object type and the last occurrence of a selected target activity.** The KPIs in this category are measured with respect to the first occurrence of an event that includes an object of the given time. For instance, regarding the utility-provider company, the first target activity of interest is *SES Line Registered*. It indicates that the service requested by the customer is provided. However, as the customer can require several services, it is of interest to know when all the services requested are provided. The second target activity, *SES Line Released*, indicates that a further step is performed, which is the confirmation from the manager that everything is received correctly. Another interesting activity to be monitored is *Invoice Receipt*, which indicates that the invoice is correctly charged to the customer; conversely, *Invoice Cleared* indicates that the invoice is paid. Also for the technology company, two interesting activities to be monitored were *Invoice Receipt* and *Invoice Reconciled* that, similarly to *Invoice Cleared*, indicate that the invoice is paid. The third target activity of interest is *Invoice Submit*, which indicates that the Invoice is registered into the system. The last interesting activity to be monitored is *Invoice Approved*, which indicates that the invoice that is submitted for registration is approved by a manager.

**Pay Delay estimation.** It refers to the number of days exceeding the planned payment date, starting from the contract's creation to the last occurrence of *Invoice Cleared*.

**Occurrence of activity / occurrence of attribute with a particular value.** It refers to whether a certain activity or condition (e.g., a late payment) will occur in the future. This category is boolean, with true indicating the occurrence, and false the

[7] The presence of NDAs prevent the authors from publicly sharing the datasets.

Table 2: Descriptive statistics of selected KPIs.

(a) Numerical KPIs. Process 1 and 2 refer to the energy company and to the IT company, respectively. Values are reported in days.

| Process | ID | KPI | Average | Standard Deviation |
|---|---|---|---|---|
| 1 | 1 | Elapsed Time from Contract to the last SES Line Registered | 278.22 | 230.96 |
| 1 | 2 | Elapsed Time from Contract to the last SES Line Released | 279.42 | 230.94 |
| 1 | 3 | Elapsed Time from Contract to the last Invoice Receipt | 237.11 | 218.61 |
| 1 | 4 | Elapsed Time from Contract to the last Invoice Cleared | 287.14 | 229.69 |
| 1 | 5 | Pay Delay estimation from Contract to the last Invoice Cleared | 11.41 | 53.57 |
| 1 | 6 | Elapsed Time from Order to the last Invoice Receipt | 28.34 | 38.20 |
| 1 | 7 | Elapsed Time from Order to the last Invoice Cleared | 45.89 | 51.89 |
| 1 | 8 | Elapsed Time from Requisition to the last Invoice Receipt | 61.58 | 49.44 |
| 1 | 9 | Elapsed Time from Requisition to the last Invoice Cleared | 115.68 | 60.18 |
| 1 | 10 | Elapsed Time from Requisition to the last SES Line Released | 50.76 | 52.44 |
| 1 | 11 | Elapsed Time from Requisition to the last SES Line Registered | 49.79 | 52.25 |
| 2 | 12 | Elapsed Time from Requisition to the last Invoice Reconciled | 28.95 | 54.94 |
| 2 | 13 | Elapsed Time from Requisition to the last Invoice Receipt | 26.57 | 53.06 |
| 2 | 14 | Elapsed Time from Requisition to the last Invoice Submit | 27.65 | 54.31 |
| 2 | 15 | Elapsed Time from Requisition to the last Invoice Approved | 27.99 | 54.74 |
| 2 | 16 | Elapsed Time from Order to the last Invoice Reconciled | 37.65 | 71.45 |
| 2 | 17 | Elapsed Time from Order to the last Invoice Submit | 36.29 | 70.93 |
| 2 | 18 | Elapsed Time from Order to the last Invoice Approved | 36.74 | 71.40 |

(b) The percentage of graph instances in which the activity occurred, or the attribute is present with that value. KPIs are all related to the process for the energy company.

| KPI | Percentage |
|---|---|
| Occurrence of Activity Purchase Order Blocked (from Contract to the last Invoice Cleared) | 27% |
| Occurrence of Activity Pay Method Changed (from Contract to the last Invoice Cleared) | 26% |
| Occurrence of Attribute Pay Type Assuming Value Late (from Contract to the last Invoice Cleared) | 61% |

absence. First, the company is interested to know in advance whether there would be changes to the payment method (represented by the activity *Invoice Pay Method Changed*). When this activity happens, there are usually delays in payments. The company is also interested in predicting whether there will be problems with the order (represented by the activity *Purchase Order Blocked*) since this situation can bring additional delays caused by the reworks needed to fix the problem. Finally, it is interesting to know whether there will be delays with the payments (represented by the attribute *Pay Type* assuming value Late).

Table 2 enumerates the KPIs on which we performed experiments. In particular, Table 2a focuses different numerical KPIs, while Table 2b reports on boolean KPIs related to the occurrence of two activities and to the occurrence of one attribute taking on a certain value: for this table, we report the percentage of the graph instances for which the activity or the pair attribute-value is observed.

Each KPI definition used in our evaluation is of the form *from object type t to the last activity a*, to indicate that the KPI value is undefined for prefixes that ends *(i)* before the first event that incorporates objects of type *t*, or *(ii)* after the last event that refers to *a*. This means that, both in the train and test phases, we do not consider the prefixes for which the KPI values are undefined.

For the LSTM implementation we relied on the Keras framework and, as learning algorithm, we adopted ADAM with Nesterov Momentum (NAdam). For the GCN and the GGNN implementation, we used the Tensorflow framework, leveraging ADAM as learning algorithm. For all the three models, we selected 200 training epochs with a patience of 25. [8] For the Catboost implementation, instead, we leveraged the open source library available at `https://catboost.ai/`.

The splitting in training and test set is performed on the set of graph instances that are obtained from the event log: two third of them are used for training, and one third for testing. During training, a hyperparameter optimization was performed, in which we used the last 20% of the training set as a validation set. For LSTM, in particular, we validated the number of LSTM neurons used for each layer (which varied between 100 and 250), and the number of layers (1, 2 and 4), with a 20% dropout for each layer. We found that the best architecture consisted in most cases of two LSTM layers with 100 neurons each and a 20% dropout for each layer. For Catboost, conversely, we validated the number of trees used (which varied between 1500, 3000 and 4000) and the depth of each single tree (3, 6 and 10). We found that the best architecture consisted in most cases of 1,500 trees, each with a depth of 10.

Regarding the GCN implementation, the architecture was inspired by the work of [13], which included a GCN layer with one channel, followed by a global average pooling layer, a dropout layer with a dropout rate of 50%, two Dense layers with 256 neurons and $tanh$ activation, and a second dropout layer with the same dropout rate. In particular, we validated the number of channels in the GCN layer (which varied between 1 and 2), the number of final dense layers (we considered keeping the two layers or removing them) and the number of neurons for each dense layer (which varied between 100 and 250). However, we found the original architecture to be the most effective one.

Conversely, the architecture of the GGNN, was inspired by the work of [14]; the original architecture included a Gated Graph layer with four GRU cell iterations and $tanh$ activation, followed by a Global Attention layer with 100 output channels and three Dense layers with a dropout rate of 50% each and with 300, 200 and 100 neurons, respectively. In particular, we validated the number of GRU cell iterations in the Gated Graph layer (which varied between 1, 2, and 4), the number of final Dense layers (which varied between 1, 2, 3 or no layers at all) and the number of neurons for each Dense layer (which varied between 100 and 250). We found that the best architecture consisted in most cases of two GRU cell iterations and no final Dense layers at all.

We calculated the Mean Absolute Error (MAE) for the 18 numerical KPIs as values were reasonably well balanced. By contrast, we calculated the F1-Score for the last three KPIs. These KPIs are categorical and relate to activities the energy company wants to prevent. Finally, we report the training time required to train every prediction model for each KPI of interest.

[8] The code can be found at https://github.com/PyRicky/graph_object_centric_prediction_fau/

## 5 Evaluation Results

Table 3 summarizes the results of our comparison. *We first observe that Catboost achieves the highest predictive accuracy among the four prediction models in almost all the considered KPIs, except for one case, where GCN obtains slightly better predictive accuracy. Also, Catboost models are learned significantly faster: summing up the training time of Catboost and GCN in Table 3 for every process and KPI, Catboost requires overall 7h and 49m, while GCN 59h and 19m (870%).*

We further compare the obtained results with the statistics of the selected KPIs in Table 2; in particular, we noticed that the event logs obtained for the numerical KPIs 3 to 7 and for the categorical KPIs 19 to 21 are those that contain more events. In these settings, the predictive accuracy of LSTM is closer to that of Catboost, and considerably outperforms those of GGNN and GCN. However, for KPIs 12 to 18 related to the second case study, while GGNN always performs better than GCN and there is not a clear winner between GGNN and LSTM, Catboost systematically outperforms other methods. Linked to the point above, LSTM can naturally learn from sequences of events, thus learning from the interaction among process objects. By contrast, GGNN and GCN tend to focus on adjacency matrices of nodes (i.e., events) in proximity, being less capable of reason on events that are indirectly connected. However, while the LSTM does not always outperform GGNN, Catboost systematically performs better than the other models because of the aggregated features, designed to capture the object-interaction [6]. Conversely, for the numerical KPIs 8 to 11, which are characterized by fewer events, the GCN outperforms LSTM and shows a predictive accuracy relatively comparable to Catboost. From this, we can conclude that the GCN can occasionally have slightly better performances in the presence of the limited amount of data. On the other hand, if enough data is provided, Catboost systematically outperforms graph-based approaches, which conversely struggle to learn more complicated interaction patterns, and also LSTM, which is known to require large amount of data.

When the KPI is related to the (non) occurrence of a process' activity (e.g., *Occurrence of Activity Purchase Order Blocked*) that is seldom observed (see KPIs 19 and 20), we observed that Catboost models significantly surpass graph-based neural networks, which are also outperformed by LSTM networks. When the activity is more common, graph-based neural networks show better predictive accuracy, which however usually remains lower than that of LSTM and Catboost models.

We can finally conclude that, when gradient boosting is used, the engineering of features to encode the object interaction enables obtaining prediction accuracy that is higher than that of graph-based neural networks. The preference of gradient-boosting-based techniques over those leveraging on graph-based neural network is further testified by the fact techniques relying on graph-based neural networks require a training time that is eight times longer. The comparison of graph-based neural networks and LSTM's shows that there is no clear winner: sometimes the former perform better, other times the latter does. This does not fully confirm the work by Adams et al. [2], which reported on the superiority of graph-based neural networks. But they only conducted two case studies, likely insufficient to reach more general conclusions.

This is likely partially linked to the manual engineering of object-interaction features, which is an informed tuning for the specific problem of object-centric process predictive analytics. Note how this paper does not flatten the event log as in [6] when

Table 3: Predictive accuracy for KPIs and prediction models. An horizontal line split the Numerical and Boolean KPIs, with the former measured in terms of Mean Absolute Error (MAE) and the latter as F1-Score. Training times are reported in brackets.

| Log | ID | KPI | GCN | GGNN | LSTM | Catboost |
|---|---|---|---|---|---|---|
| 1 | 1 | Elapsed Time from Contract to the last SES Line Registered | 42.85 (1h 5m) | 41.17 (1h 45m) | 45.22 (3h 34m) | **31.9 (11m)** |
| 1 | 2 | Elapsed Time from Contract to the last SES Line Released | 42.31 (1h 13m) | 47.99 (3h 12m) | 50.77 (4h 9m) | **33.09 (21m)** |
| 1 | 3 | Elapsed Time from Contract to the last Invoice Receipt | 42.64 (2h 59m) | 41.19 (8h 7m) | 37.72 (5h 23m) | **30.78 (42m)** |
| 1 | 4 | Elapsed Time from Contract to the last Invoice Cleared | 47.86 (6h 12m) | 44.37 (9h 17m) | 39.05 (12h 6m) | **34.4 (49m)** |
| 1 | 5 | Pay Delay estimation from Contract to the last Invoice Cleared | 17.09 (2h 12m) | 18.47 (10h 4m) | 14.05 (8h 53m) | **12.36 (11m)** |
| 1 | 6 | Elapsed Time from Order to the last Invoice Receipt | 27.51 (2h 29m) | 36.44 (5h 45m) | 26.71 (2h 47m) | **19.15 (21m)** |
| 1 | 7 | Elapsed Time from Order to the last Invoice Cleared | 29.95 (7h 56m) | 37.04 (11h 50m) | 23.22 (4h 19m) | **20.08 (1h 21m)** |
| 1 | 8 | Elapsed Time from Requisition to the last Invoice Receipt | 34.39 (41m) | 45.62 (33m) | 41.36 (56m) | **31.08 (5m)** |
| 1 | 9 | Elapsed Time from Requisition to the last Invoice Cleared | **35.33 (16m)** | 67.04 (27m) | 40.96 (2h 15m) | 36.71 (17m) |
| 1 | 10 | Elapsed Time from Requisition to the last SES Line Released | 32.23 (5m) | 75.21 (20m) | 55.4 (49m) | **26.96 (3m)** |
| 1 | 11 | Elapsed Time from Requisition to the last SES Line Registered | 31.52 (5m) | 81.62 (14m) | 48.97 (48m) | **31.2 (6m)** |
| 2 | 12 | Elapsed Time from Requisition to the last Invoice Reconciled | 45.86 (2h 56m) | 42.12 (4h 47m) | 40.5 (1h 16m) | **27.5 (13m)** |
| 2 | 13 | Elapsed Time from Requisition to the last Invoice Receipt | 48.16 (3h 10m) | 44.31 (4h 17m) | 49.06 (1h 9m) | **29.44 (12m)** |
| 2 | 14 | Elapsed Time from Requisition to the last Invoice Submit | 47.88 (1h 45m) | 43.02 (5h 12m) | 43 (1h 3m) | **27.82 (13m)** |
| 2 | 15 | Elapsed Time from Requisition to the last Invoice Approved | 47.81 (3h 54m) | 43.87 (3h 50m) | 42.53 (1h 2m) | **28.75 (8m)** |
| 2 | 16 | Elapsed Time from Order to the last Invoice Reconciled | 51.3 (2h 50m) | 46.6 (8h) | 50.60 (1h 28m) | **25.48 (9m)** |
| 2 | 17 | Elapsed Time from Order to the last Invoice Submit | 53.13 (4h 51m) | 46.89 (10h 31m) | 45.23 (42m) | **24.02 (14m)** |
| 2 | 18 | Elapsed Time from Order to the last Invoice Approved | 51.61 (5h 14m) | 47.14 (5h 3m) | 47.49 (42m) | **25.26 (14m)** |
| 1 | 19 | Occurrence of Activity Purchase Order Blocked | 0.33 (4h 45m) | 0.37 (9h 12m) | 0.51 (6h 4m) | **0.60 (20m)** |
| 1 | 20 | Occurrence of Activity Invoice Pay Method Changed | 0.38 (3h 35m) | 0.50 (13h) | 0.64 (6h 22m) | **0.74 (20m)** |
| 1 | 21 | Occurrence of Attribute Pay Type Late | 0.73 (1h 6m) | 0.75 (5h 38m) | **0.82 (7h 12m)** | **0.82 (19m)** |

Catboost or LSTM models are employed: here, graph instances are still created and encoded as sequences.

Graph-based neural networks conversely are general purpose, and are not informed on the specific characteristics of the graphs that encodes the interactions. Internally, they need to learn an abstraction of the graph structure, and no specific known characteristics of these graph instances can be leveraged on for a better abstraction. This yields lower accuracy, as well as the effort to learn an abstraction is paid via higher training time.

## 6 Conclusion

The object-centric process paradigm is increasingly gaining popularity in academia and industry. According to this paradigm, the process is seen as the interplay of numerous processes that constitute the life cycles of different objects of various types, where these life cycles periodically synchronize. The presence of many-to-many interactions between objects (i.e., processes) prevents the direct application of existing techniques, designed for traditional processes with one single id and execution flow.

This paper reports on the experience of comparing the predictions of two techniques based on gradient boosting on decision tree or LSTM against two based on graph neural networks. The four techniques were empirically evaluated on event logs related to two real object-centric processes, and more than 20 different KPI definitions. The results illustrate that the technique based on gradient boosting generally shows the highest accuracy, likely thanks to a proper engineering of the features to encode the object interaction. At the same time, it is more than 800% faster than the techniques relying on graph-based neural networks. For the two adopted datasets and the 21 KPIs employed in our experiments, gradient boosting on decision trees is more suitable than graph-based neural networks for object-centric process predictive analytics.

Note how the Gradient-Boosting technique discussed in this paper does not use the event-log flattening solution proposed in [6]: in this paper, we still represent the objects' interaction via graph instances, and the sequence conversion is only done as last step, retaining a set of features that meaningfully encode the graph-like structure.

Future work aims to conduct additional experiments on different publicly-available datasets and to also perform cross-validation to further validate the findings reported here. We also plan on testing different manual engineering of the object-interaction features; it cannot be excluded, indeed, that the features that so far we manually engineered are not the right abstraction of the object interaction in every case study. Moreover, we cannot exclude that a different trace encoding function for LSTM and graph-based neural networks could improve the performances of these models.

*Acknowledgement.* This research is partly funded by Department of Mathematics of University of Padua, through the BIRD project "Data-driven Business Process Improvement" (code `BIRD215924/21`).

## References

1. van der Aalst, W.M.P.: Object-centric process mining: Dealing with divergence and convergence in event data. In: Proceedings of 17th International Conference on Software Engineering and Formal Methods (SEFM 2019). LNCS, vol. 11724, pp. 3–25. Springer (2019)
2. Adams, J.N., Park, G., Levich, S., Schuster, D., van der Aalst, W.M.P.: A framework for extracting and encoding features from object-centric event data. In: Service-Oriented Computing. pp. 36–53. Springer Nature Switzerland (2022)
3. Dorogush, A.V., Ershov, V., Gulin, A.: Catboost: Gradient boosting with categorical features support. In: Proceedings of the Workshop on ML Systems at NIPS 2017 (2017)
4. Farhang Ghahfarokhi, A., Park, G., Berti, A., van der Aalst, W.M.P.: OCEL Standard (January 2020), Avail. at `http://ocel-standard.org/1.0/specification.pdf`
5. Galanti, R., de Leoni, M., Monaro, M., Navarin, N., Marazzi, A., Di Stasi, B., Maldera, S.: An explainable decision support system for predictive process analytics. Engineering Applications of Artificial Intelligence **120**, 105904 (2023)
6. Galanti, R., de Leoni, M., Navarin, N., Marazzi, A.: Object-centric process predictive analytics. Expert Systems with Applications **213** (2023)
7. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2017)
8. Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R.: Gated Graph Sequence Neural Networks. In: Proceedings of the 4th Intl. Conference on Learning Representations (ICLR) (2016)
9. Márquez-Chamorro, A.E., Resinas, M., Ruiz-Cortés, A.: Predictive monitoring of business processes: A survey. IEEE Transaction on Services Computing **11**(6), 962–977 (2018)
10. Newman, M.E.J.: Power laws, Pareto distributions and Zipf's law, vol. 46, pp. 323–351. Taylor & Francis (2005)
11. Park, G., Song, M.: Prediction-based resource allocation using LSTM and minimum cost and maximum flow algorithm. In: Proceedings of the International Conference on Process Mining, ICPM 2019, Aachen, Germany, 2019. pp. 121–128. IEEE (2019)
12. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: Proceedings of the 29th International Conference on Advanced Information Systems Engineering (CAiSE 2017). vol. 10253, pp. 477–492. Springer (2017)
13. Venugopal, I., Töllich, J., Fairbank, M., Scherp, A.: A comparison of deep-learning methods for analysing and predicting business processes. In: 2021 International Joint Conference on Neural Networks (IJCNN). pp. 1–8. IEEE (2021)
14. Weinzierl, S.: Exploring gated graph sequence neural networks for predicting next process activities. In: Business Process Management Workshops. pp. 30–42. Springer (2022)