How Object-Centric is Object-Centric Predictive Process Monitoring?

Introducing Objects into Object-Centric Predictive Process Monitoring

Tim Smit

t.k.smit@students.uu.nl

Utrecht University

First supervisor	Dr. ir. Xixi Lu
Second supervisor	Prof. dr. ir. Hajo A. Reijers
External supervisor	MSc. ir. Mike W.H.A.C. van Bussel

September 19, 2023

Acknowledgements

In the process of this thesis project I've received much support from quite some people. I want to use this space to just express my gratefulness.

First and foremost, I want to thank Jesus Christ, my King and dear Friend. Throughout this thesis project, I've gone through some difficult situations on a personal level. I thank You for guiding me through this and giving me the grace and love I needed to respond to my circumstances while in the middle of the hardest, most intellectually challenging project I've ever been in.

Second, I want to earnestly thank Dr. ir. Xixi Lu. Thank you for helping me become a better, more positively energized, yet critical, researcher. We have laughed a lot, discussed many interesting topics, and most importantly, you've set an environment for me to learn and explore a great deal. I started out not even knowing how to train a deep learning model. During the first phase, I thought it could potentially be useful to learn about Graph Neural Networks as a hobby, hoping I would discover they aren't that hard after all... Well, dream on past Tim, it's hard, really hard. I don't know whether I would have taken on the challenge, if it was not for your guidance and encouragement.

Then, I would like to thank Mike van Bussel and my team at the partnering organization who enabled a large part of this amazing journey. I am grateful for the encouraging words and mentorship you provided in our weekly sessions. I learned a great deal about being a data professional and met many interesting people. I hope and trust our roads don't end here.

Next, I want to express my thanksgiving to Prof. dr. ir. Hajo Reijers for the helpful feedback that crucially improved this research.

In addition, at the home front, I want to thank Lydia my beautiful girlfriend. I feel strengthened by your support. By aligning my head with the things that really matter, by being there to listen to my lamentations, and by giving me air support through prayer. That is just to briefly name a few things. I really feel grateful for you, also in supporting me working on this project.

Besides that, I want to acknowledge the help of friends and peers. Thank you Benjamin for the great times working together on our theses and the useful feedback you gave. Also, thank you housemates for putting up with me being so focussed on this project.

Finally, I would like to express my gratefulness to my parents, brothers, and, Lydia's parents and even grandparents for supporting me through showing interest and care, setting time apart for me to do sports, and for your faithful prayers.

Abstract

The field of process mining is challenged by the complexity of true processes when extracting accurate process behavior and statistics from information systems. Traditional process mining algorithms assume a single case notion, whereas actual processes contain many possible ones, represented by objects. Object-centric process mining has been introduced as a case-agnostic solution, which mitigates the problem of misleading process behavior and statistics via the object-centric event log (OCEL). It allows multiple case notions which are called object types. Objects have many-to-many relationships with events. However, when performing predictive process monitoring on OCELs, issues arise when including object information as features due to these many-to-many relationships. This has not been addressed by existing literature. We propose a heterogeneous object event graph encoding (HOEG), that incorporates events and objects into a graph with different node types. We evaluate our novel encoding against an extant graph-based encoding and several baselines on the task of remaining time prediction. On our HOEG we employ a heterogeneous graph neural network (GNN) architecture that is converted from a homogeneous one. The HOEG-based GNN learns an optimal way to include object information when forming predictions. The experiments are executed on three OCELs, one of which is extracted from an operational process at a large Dutch financial institution. Our results indicate that HOEG outperforms its competition for well-structured OCELs. Furthermore, we argue that HOEG mainly excels when OCELs host informative object attributes and abundant object interactions. Considering this, we propose HOEG as a promising general technique to leverage the multi-dimensional data structure given in OCELs for tasks like predicting process remaining time.

Keywords: Object-Centric Process Mining; Graph Machine Learning; Predictive Process Monitoring; Heterogeneous Graph Neural Networks; Feature Encoding

Contents

1 Introduction

Process mining has long been struggling to mine correct process behavior and statistics from information systems [1–19]. In both industry and academia, a single case notion was being assumed, which often did not reflect the true nature of the processes at hand. In reality, most processes contain multiple interacting objects, a subset of which can be discovered in information system logging [19]. Taking a recruitment process as an example, these objects could be *vacancy*, *application*, *applicant*, *job offer*, *resource*, etc. Generally, these objects are contained as entities in relational databases that are behind the information systems that support a process [19, 20]. Now, when selecting one of these objects as a case notion, one views the event log from a particular perspective, and one "flattens" the event log [5]. Previous work [4, 13] has shown that this leads to divergence, convergence, and deficiency, amounting to misleading statistics and behavior.

Over the years, several approaches to deal with these issues have been proposed [1, 2, 4, 5, 7, 15, 16, 18, 19, 21, 22]. Van der Aalst *et al.* [1], introduce us to the limitations of traditional business process management languages and systems (BPMSs). He demonstrates that any process may contain entities with many-to-many relationships, which causes issues when incorporating these into one process model. Moreover, he argues that interactions between different processes are lost when applying traditional process mining techniques. Hull [15] and Cohn & Hull [16] take a data management perspective, suggesting capturing the evolvement of business artifacts as they pass through business processes. They argue their approach as being more holistic since interaction dynamics can be expressed and data and process developments are merged into one actionable data structure. Fahland *et al.* [18] and Lu *et al.* [19] further popularized this artifact-centric view by introducing artifact-centric conformance checking and process discovery respectively. In 2017, a new object-centric behavioral constraint modeling language was introduced to merge data and behavior perspectives and capture fine-grained object interactions and life-cycles [2]. And in 2020, Ghahfarokhi *et al.* [23] proposed the object-centric event log (OCEL), a new standard. This solution can handle multiple case notions via one multi-dimensional data structure. Finally, process mining (PM) tool vendors in the industry (Celonis and Mehrwerk) are embracing the object-centric process mining paradigm [24, 25]. With these developments, the question arises of how the sub-field of predictive process monitoring (PPM) is affected. Can PPM adopting object-centricity leverage the more comprehensive and correct process behavior and performance statistics for better results? How to handle the new data structure in PPM? What predictive models perform well on OCELs?

To gain an initial understanding of an object-centric process, consider an example of an order at an e-commerce store.

Example 1: Object Multiplicity in Placing an Order

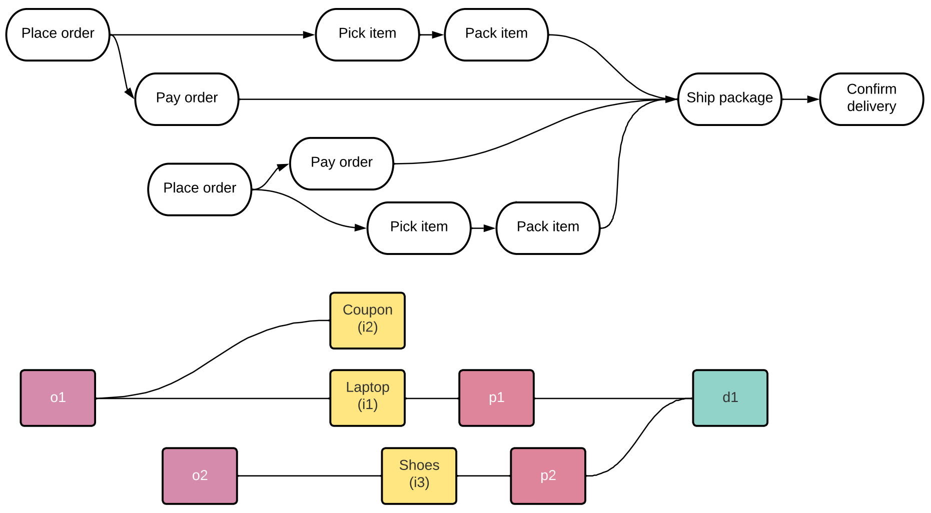

Figure 1. Digital process trace left behind by the order placement scenario. Events are shown on top, and objects (colored on object type) are given at the bottom.

Suppose that a customer places an order, buying a laptop and a digital coupon. The customer needs the laptop the same day and thus pays extra to have same-day delivery. Later that day, another order by the customer is recorded, containing only a pair of shoes among the order line items. After that, when all order-picking-related tasks have been performed, the items are packaged and loaded onto a delivery vehicle. The coupon is not packaged since it is digital, so an email is sent instead of an actual package. Recognizing that the laptop and shoes have the same delivery address, the e-commerce store ships them in the same delivery.

In this example, we can recognize eight objects of four object types (in bold): two sales **order**s, three sales order **items**, two **package**s, and one **delivery** (illustrated in Figure 1). These object types could have attributes such as `urgency` for **order**, or `route length` for **delivery**. Furthermore, Figure 1 visualizes the process execution using the following activities: *Place order*, *Pay order*, *Pick item*, *Pack item*, *Ship package*, and *Confirm delivery*. These activities are related to a `timestamp` and may have the event attribute `resource`.

Conceptualizing this from an object-centric data perspective, we can distinguish two viewpoints: the events and the objects, resulting in an events table and an objects table [23]. Consequently, when performing machine learning on such data, we observe these same two perspectives. When using the events perspective, we obtain a feature vector per event, containing information about the activity, event attributes, and possibly other event-level derivations. Taking the objects perspective, we obtain a feature vector per object, containing information about object attributes and possibly other derived object-based features.

Early attempts in object-centric predictive process monitoring show promising results [26–29]. Most notably Adams *et al.* [27] propose a framework for event-level object-centric feature extraction and Berti *et al.* [30] propound object-level feature extraction. The former (i.a.) innovatively suggests an event-level feature encoding in which an object-centric directly-follows graph (OC-DFG) is discovered per process execution and used as an encoding structure. They then append features to the nodes in these graphs. The latter delves into exploiting object interactions via object graphs (see Def. 4 in [30]).

Though an object-centric event feature graph encoding already introduces more informative features compared to traditional event log-based features, we recognize limitations when it comes to learning from object characteristics. Using the previous example, we elaborate on two limitations that come with aggregating object information into the events perspective: unavailability of objects and information loss due to object attribute aggregation.

First, `route length` cannot be appended as an event feature, as only the last two events relate to a **delivery** (`d1`). A solution would be filling `route length` for the events that do not refer to a *delivery* object. However, this can confuse the predictive model as there is no natural default `route length` value to use for filling.

Second, suppose we want to predict the remaining time of this example process at the *Ship package* event. In the example, the first order has a high `urgency`, say 1. The second order has less urgence, as the customer did not pay an additional fee for same-day delivery. Assume `o2` has an `urgency` value of 3. Furthermore, suppose another customer, at a different time, places an order `o3`, buying the same laptop, but with an urgency value of 2. If we encode `urgency` as an event feature, we must take an aggregate, like the average of all `urgency` values that *Ship package* refers to. For both process executions, this would amount to `urgency` = 2. Now, we would learn similar patterns for these traces and give an equal prediction for both process executions, even though one included a high `urgency` order and the other a regular `urgency` one.

On the objects perspective, the object graph approach does not consider events explicitly. Applying this to the example, we observe that we miss the granular information provided by each event.

These observations inspire the investigation of features and encodings that facilitate the learning of patterns from both event and object perspectives within OCELs. This has not yet been explored in object-centric PPM literature. With this research, we aim to address this gap. The central research question then is:

How to leverage the multi-dimensional data structure given in object-centric event logs in order to predict process remaining time?

Supporting this are the following sub-questions:

SRQ1 How do existing approaches perform machine learning tasks on object-centric event logs?

SRQ2 How does extant predictive process mining literature take advantage of the enhanced data structure provided by the object-centric event log?

SRQ3 What feature encodings enable machine learning models to take advantage of the enhanced data structure provided by the object-centric event log for predictive tasks?

SRQ4 How do machine learning models operating on these feature encodings perform when predicting process remaining time?

Consequently, it is our research objective *to design a general technique for encoding object-centric events, objects, attributes, and features, facilitating predictive tasks without the need for flattening events, filling unavailable object features, or aggregating on objects*. Given that predicting remaining times in object-centric process executions stands as a generic task in the field, this serves as our target for evaluating the various feature encoding types investigated. In light of this, we contribute to the field a framework for thinking about features derived from object-centric event logs. That is, using our results and analysis one can make a nuanced trade-off in choosing either an event-based or an event-object-based encoding for graph machine learning on OCELs.

Accomplishing our research objective and resolving our central question involves providing a theoretical background (Section 2) and surveying related works (Section 3). Section 4 then proposes our novel encoding to tackle the limitations of current efforts presented in the previous chapter. Subsequently, the Experimental Setup delineates the framework for the experiments that are executed. These are used to evaluate the novel encoding against the state of the art on three object-centric datasets. Most notably, one dataset was extracted from a complex real-life process that is under high pressure at a large Dutch financial institution, where compliance is eminent. Section 6 presents the results of the experiments. The Discussion, then, gives critical interpretations of the results, recommendations for configuring our novel encoding, and a synthesis conveying the implications of the results. We conclude in Section 8 by summarizing our work, attending to our research questions, and providing suggestions for future work.

2 Theoretical Background

This section provides the theoretical preliminaries. Definitions are given for both the core process mining concepts and the machine learning (ML) ideas used.

2.1 Object-Centric Event Logs

An object-centric event log (OCEL) consists of events and objects related to a business process execution. At a minimum, an OCEL contains event vectors with a related activity, timestamp, and one (or more) object reference(s). The coming paragraphs give visually supported formal definitions for an OCEL and reify the concepts that are introduced through an approachable realistic example.

Definition 2.1 (Universes)**.** These following are the universes used in the formal definition of OCEL [23].

- U_e is the universe of event identifiers.
- U_{act} is the universe of activities
- U_{att} is the universe of attribute names.
- U_{val} is the universe of attribute values.
- U_{typ} is the universe of attribute types.
- U_o is the universe of object identifiers.
- U_{ot} is the universe of objects types.
- U_{timest} is the universe of timestamps.

Definition 2.2 (Object-Centric Event Log)**.** An object-centric event log [23] is a tuple $OCEL = (E, AN, AV, AT, OT, O, \pi_{typ}, \pi_{act}, \pi_{time}, \pi_{vmap}, \pi_{omap}, \pi_{otyp}, \pi_{ovmap}, \leq)$ such that:

- $E \subseteq U_e$ is the set of event identifiers.
- $AN \subseteq U_{att}$ is the set of attributes names.
- $AV \subseteq U_{val}$ is the set of attribute values, such that $AN \cap AV = \emptyset$.
- $AT \subseteq U_{typ}$ is the set of attribute types.
- $OT \subseteq U_{ot}$ is the set of object types.
- $O \subseteq U_o$ is the set of object identifiers.
- $\pi_{typ} : AN \cup AV \to AT$ is the function associating an attribute name or value to its corresponding type.
- $\pi_{act} : E \to U_{act}$ is the function associating an event (identifier) to its activity.
- $\pi_{time} : E \to U_{timest}$ is the function associating an event (identifier) to a timestamp.

- $\pi_{vmap} : E \rightarrow (AN \nrightarrow AV)$ such that $\pi_{typ}(n) = \pi_{typ}(\pi_{vmap}(e)(n)) \quad \forall e \in E \forall n \in \text{dom}(\pi_{vmap}(e))$ is the function associating an event (identifier) to its attribute value assignments (See Relation 1 in Figure 2).
- $\pi_{omap} : E \rightarrow \mathcal{P}(O)$ is the function mapping an event (identifier) to a set of related object identifiers (cf. Relation 2 in Figure 2).
- $\pi_{otyp} \in O \rightarrow OT$ assigns one and only one object type to each object identifier.
- $\pi_{ovmap} : O \rightarrow (AN \nrightarrow AV)$ such that

$$\pi_{typ}(n) = \pi_{typ}(\pi_{ovmap}(o)(n)) \quad \forall n \in \text{dom}(\pi_{ovmap}(o))\, \forall o \in O$$

is the function that relates an object to its attribute value assignments (cf. Relation 3 in Figure 2). As defined, it takes two parameters, object o and an attribute name n, for $o \in O$ and $n \in AN$.
- $\leq$ is a total order (i.e., it respects the antisymmetry, transitivity, and connexity properties). A possible way to define a total order is to consider the timestamps associated with the events as a pre-order (i.e., assuming some arbitrary, but fixed order for events having the same timestamp). Figure 2 models this as an attribute of the `Log` class.

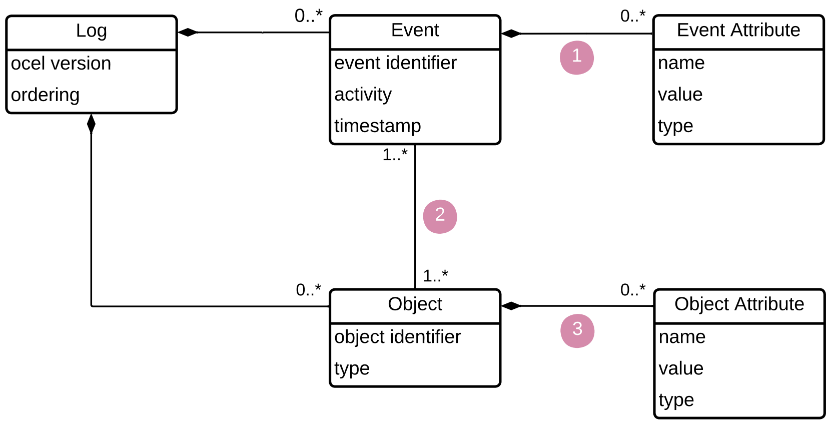

Figure 2. A UML Class diagram depicting the metamodel for OCEL (v1.0), describing the relationships between the entities visually. Adapted from [23].

Definition 2.3 (Traditional Event Log)**.** In light of an object-centric event log, a traditional event log is a tuple $EL = (E, AN, AV, AT, \Sigma, \pi_{typ}, \pi_{act}, \pi_{time}\pi_{vmap}, \pi_{omap} \leq)$ where the definitions in Definition 2 hold. Moreover, $\Sigma = \{\sigma \in U_{ot}\}$, where trace σ is a sequence of event identifiers such that $|\pi_{omap}(e)(\sigma)| = 1 \; \forall e \in E$. We see that σ corresponds to a single object type in an OCEL.

We can observe that an object-centric event log is a generalization of a traditional event log, where each event has one and only one case identifier linking events to traces. Suppose there exist multiple objects that relate to events in an OCEL. This means that one event may refer to multiple objects of possibly different types. When considering only one object type, we would convert object-centric data to a traditional event log as follows. When an event refers to no objects of the considered type, the event is omitted (*deficiency*). When an event has one object reference to the selected type, the event is kept. Finally, if an event is associated with multiple objects of the respective type, the event is duplicated for each object reference (*convergence*). We experience *divergence* when events that reference different objects of the same object type not selected as the case notion, are taken to be causally related. Van der Aalst & Berti [5] mathematically outlines event log flattening as follows.

Definition 2.4 (Flattening an Object-Centric Event Log)**.** Let $L = (E, \preceq_E)$ be an object-centric event log and $ot \in \mathbb{U}_{ot}$ an object type representing a case notion.
The flattened event log, then, is $L^{ot} = \left(E^{ot}, \preceq_E^{ot}\right)$, where:

- $e_i = \left(\left(\pi_{ei}(e), i\right), \pi_{act}(e), \pi_{time}(e), \pi_{omap}(e) \oplus (\ case,\ \{i\}), \pi_{vmap}(e)\right)$ for any $e \in E$ and $i \in \pi_{omap}(e)(ot)$,
- $E^{ot} = \{e_i \mid e \in E \wedge i \in \pi_{omap}(e)(ot)\}$, and
- $\preceq_E^{ot} = \left\{\left(e'_i, e''_j\right) \in E^{ot} \times E^{ot} \mid e' \in E \wedge i \in \pi_{omap}\left(e'\right)(\ ot) \wedge e'' \in E \wedge j \in \pi_{omap}\left(e''\right)(ot) \wedge e' \preceq_E e'' \wedge (e' = e'' \Rightarrow i = j)\right\}$.

A flattened event log is still an event log after removing and duplicating events. See Definition 4.1 in [5].

Convergence, and *divergence*, were first conceptualized by Segers [13]. Van der Aalst [4] adds the concept of *deficiency* and demonstrates the three phenomena through examples. To aid our understanding, we define a use case that indicates the limitations of traditional event logs.

Example 2: Organizing the Complexity of Order to Cash Executions

Consider an order-to-cash (OTC) process of an online retailer. This process *usually* starts with an order placement and ends with a package delivery confirmation.

Execution A. Suppose there is a customer named Tim who places an order, buying a laptop and a digital coupon. Later that day, another order by Tim is recorded, containing only a pair of shoes among the order line items. After that, when all order-picking-related tasks have been performed, the items can be packaged and loaded onto a delivery vehicle. The coupon is not packaged since it is digital, so an email is sent instead of an actual package. Recognizing that the other packages have the same delivery address, they are included in the same delivery. After three days total, Tim receives his orders.

Execution B. At another time, Tim places an order containing a mattress, a shirt, and a packet of lightweight medicine. The mattress, now, is relatively large and is packaged and put into a full truck. A keen employee identifies that the shirt and medicine can be efficiently packed into one package. Meanwhile, the mattress has been returned to the depot as the delivery was unsuccessful (e.g., Tim was not home). Finally, both packages are loaded into the next available truck for delivery. This delivery was later confirmed to be successful.

Data Model. From these fictitious process executions, we can distinguish four object types: **order**, **item**, **package**, and **delivery**. **Order** has a one-to-many relationship with **item**. **Item** has a many-to-zero-or-one relationship with **package** and **package** relates many-to-many with **delivery**. This can be conceptually captured in the diagram below (Figure 3).

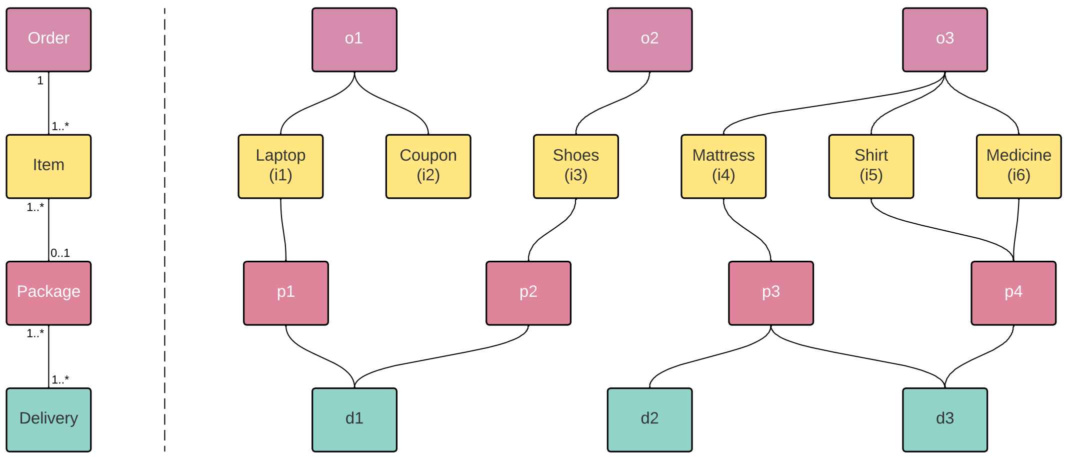

Figure 3. The data model of the example OTC scenario on entity and instance level.

OCEL. An excerpt from a possible OCEL event table arising from Execution A is expressed in Table 1. Table 2 exemplifies the second (conceptual) part of this OCEL, object tables. In the following, we make the connection from Definitions 2.1 and 2.2 to an instance of a log, to clarify their meanings.

- $E = \{e1, e2, \ldots, e18\}$
- $AN = \{resource, Urgency, discount, weight, size, route\ length, no.\ stops\}$
- $AV = \{CloudServiceA, \ldots, 678.0, \ldots, 0.0, \ldots, 0.5, \ldots, small, \ldots, long, \ldots, 18.0\}$
- $AT = \{string, float\}$
- $OT = \{order, item, package, delivery\}$
- $O = \{o1, o2, o3, i1, i2, i3, i4, i5, p1, p2, p3, p4, d1, d2, d3\}$

The functions (denoted π) then, operate like lookup functions in a map. Based on certain keys as parameters, they return the associated values. As an example, event attribute `resource` for event $e3$ can be looked up through $\pi_{vmap}(e3, resource) = CloudServiceB$ and likewise $\pi_{ovmap}(o2, Urgency) = 3.0$.

Table 1. Sample from the event table highlighting Execution A.

ID	Activity	Time	Resource	Order	Item	Package	Delivery
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
e1	Place order	2023-01-30	CloudServiceA	{o1}	{i1, i2}	{}	{}
e2	Pay order	2023-01-30	CloudServiceA	{o1}	{}	{}	{}
e3	Place order	2023-01-30	CloudServiceB	{o2}	{i3}	{}	{}
e4	Pay order	2023-01-30	CloudServiceB	{o2}	{}	{}	{}
e5	Pick item	2023-01-31	WarehouseTeamX	{o1}	{i1}	{}	{}
e6	Pick item	2023-01-31	WarehouseTeamX	{o2}	{i3}	{}	{}
e7	Pack item	2023-01-31	WarehouseTeamX	{o1}	{i1}	{p1}	{}
e8	Pack item	2023-01-31	WarehouseTeamX	{o2}	{i3}	{p2}	{}
e9	Ship package	2023-02-01	WarehouseTeamY	{o1, o2}	{i1, i3}	{p1, p2}	{d1}
e10	Confirm delivery	2023-02-02	PostalServiceP	{o1, o2}	{i1, i3}	{p1, p2}	{d1}
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 2. Object tables for the different object types.

ID	Urgency
o1	1.0
o2	3.0
o3	2.0

ID	Discount
i1	33.0
i2	0.0
i3	25.0
i4	0.0
i5	15.0
i6	0.0

ID	Weight	Size
p1	3.5	medium
p2	3.0	medium
p3	26.0	large
p4	0.5	small

ID	Route length	No. stops
d1	short	5.0
d2	normal	27.0
d3	long	18.0

Flattening. Traditional process discovery algorithms require a single case notion. For the OCEL in Table 1 this would entail flattening the table along a selected object (the case notion). If we were to select **order** as the case notion, we would yield Table 3. Looking at the flat event log, we notice that the activity *Ship package* is duplicated (*convergence*). *Deficiency* is achieved when taking **package** or **delivery** as the trace definition. That is, for both these object types, there are events that have no reference to them (`e1`, `e2`, `e3`, `e4`, and `e5`), implying they will disappear in the process of flattening.

Table 3. Execution A flattened on order. *Note* duplicate events `e9` and `e10`.

ID	Activity	Time	Resource	Order	Item	Package	Delivery
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
e1	Place order	2023-01-30	CloudServiceA	{o1}	{i1, i2}	{}	{}
e2	Pay order	2023-01-30	CloudServiceA	{o1}	{}	{}	{}
e3	Place order	2023-01-30	CloudServiceB	{o2}	{i3}	{}	{}
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
e9	Ship package	2023-02-01	WarehouseTeamY	{o1}	{i1, i3}	{p1, p2}	{d1}
e9	Ship package	2023-02-01	WarehouseTeamY	{o2}	{i1, i3}	{p1, p2}	{d1}
e10	Confirm delivery	2023-02-02	PostalServiceP	{o1}	{i1, i3}	{p1, p2}	{d1}
e10	Confirm delivery	2023-02-02	PostalServiceP	{o2}	{i1, i3}	{p1, p2}	{d1}
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

To explain *divergence*, we direct our attention to Execution B, flattened along **order** (cf. Table 4).

Table 4. Event table excerpt of Execution B exhibiting divergence.

ID	Activity	Time	Resource	Order	Item	Package	Delivery
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
e13	Place order	2023-02-03	CloudServiceA	{o3}	{i4, i5, i6}	{}	{}
e14	Pay order	2023-02-03	CloudServiceA	{o3}	{i4, i5, i6}	{}	{}
e15	Pick item	2023-02-04	WarehouseTeamX	{o3}	{i4}	{}	{}
e16	Pick item	2023-02-04	WarehouseTeamX	{o3}	{i5}	{}	{}
e17	Pack item	2023-02-04	WarehouseTeamX	{o3}	{i5}	{p4}	{}
e18	Pack item	2023-02-04	WarehouseTeamX	{o3}	{i4}	{p3}	{}
e19	Pick item	2023-02-04	WarehouseTeamX	{o3}	{i6}	{}	{}
e20	Pack item	2023-02-04	WarehouseTeamX	{o3}	{i6}	{p4}	{}
e21	Ship package	2023-02-04	WarehouseTeamY	{o3}	{i4}	{p3}	{d2}
e22	Ship package	2023-02-05	WarehouseTeamZ	{o3}	{i4, i5, i6}	{p3, p4}	{d3}
e23	Confirm delivery	2023-02-06	PostalServiceQ	{o3}	{i5, i6}	{p4}	{d3}
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Discovering a directly-follows graph (DFG) from this, we would generate the model given by Figure 4. It showcases incorrect relations between events. For instance, an item cannot be packed before it has been picked.

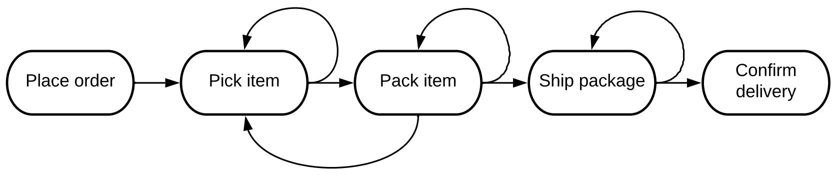

Figure 4. Directly-follows diagram of Execution B experiencing divergence.

Finally, in order to solve these issues, recent works [5, 31] propose object-centric native process discovery techniques. These would construct a model comparable to the one in Figure 5, displaying more accurate causalities between events.

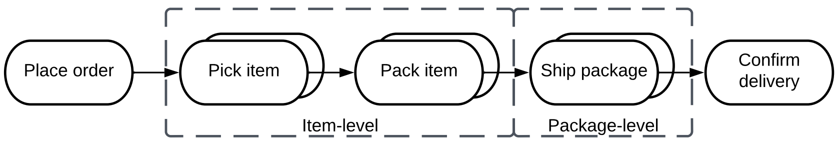

Figure 5. Directly-follows diagram of Execution B with a solution to the divergence issue. Based on Fig. 8 in [4].

2.2 Graph Neural Networks

In short, a graph neural network [32–34] is an optimizable composite function that operates over graph attributes (node, edge, and global features), preserving graph symmetries in order to model a certain target (which may be a graph, vector, or scalar variable) with respect to the data [35].

Let us unfold this into comprehensible parts. Starting with how our data is structured in a graph, we move to some fundamentals that undergird GNNs, after which we outline GNNs mathematically and by example. The following is based on the exposition given by Bronstein *et al.* [36] and Veličković [37].

2.2.1 Graph Data

We assume a graph $G = (V, E)$, such that the set of edges $E \subseteq V \times V$ specify the connectivity of the vertices in V. Furthermore, we load all nodes $u \in V$ with feature vectors $\mathbf{x}_u \in \mathbb{R}^d$, where d indicates the dimension node feature vectors. Accordingly, this is organized in *node feature matrix* $\mathbf{X} \in \mathbb{R}^{|V| \times d}$. With the form

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{|V|} \end{bmatrix} \tag{1}$$

In company with this, the set of edges[1] E is represented using an *adjacency matrix*: $\mathbf{A} \in \mathbb{R}^{|V| \times |V|}$, with

$$a_{uv} = \begin{cases} 1 & (u,v) \in E \\ 0 & (u,v) \notin E \end{cases} \tag{2}$$

2.2.2 Invariant and Equivariant to Permutations

These representations impose an ordering on the nodes, which would obstruct learning a generalized model from graph data, as permuting the order would alter the model's

[1]We refrain from including edge feature information in this explanation. It is however often possible to add and leverage this in a model.

prediction outputs. Therefore, a GNN is constrained by the following rules:

$$f\left(\mathbf{PX}, \mathbf{PAP}^{\top}\right) = f(\mathbf{X}, \mathbf{A})(\text{ Invariance }) \tag{3}$$

$$\mathbf{F}\left(\mathbf{PX}, \mathbf{PAPP}^{\top}\right) = \mathbf{PF}(\mathbf{X}, \mathbf{A})(\text{ Equivariance}) \tag{4}$$

where $\mathbf{P}$ is a permutation matrix and functions f and $\mathbf{F}$ do not affect $\mathbf{A}$.

Besides that, the edges in E permit a locality constraint in f and $\mathbf{F}$. Akin to a convolution operation in convolutional neural networks [38] for images, this means a GNN can operate over node neighborhoods. We define this neighborhood as such:

$$N_u = \{v \mid (u, v) \in E \vee (v, u) \in E\} \tag{5}$$

The multiset[2] of all neighborhood features now, is expressed through:

$$\mathbf{X}_{N_u} = \{\{\mathbf{x}_v \mid v \in N_u\}\} \tag{6}$$

Now we can specify a local function $\phi(\mathbf{x}_u, \mathbf{X}_{N_u})$ that operates on every node and their neighborhood in isolation. We can collect this into $\mathbf{F}$ such that:

$$\mathbf{F}(\mathbf{X}, \mathbf{A}) = \begin{bmatrix} - & \phi\left(\mathbf{x}_1, \mathbf{X}_{N_1}\right) & - \\ - & \phi\left(\mathbf{x}_2, \mathbf{X}_{N_2}\right) & - \\ & \vdots & \\ - & \phi\left(\mathbf{x}_n, \mathbf{X}_{N_n}\right) & - \end{bmatrix} \tag{7}$$

Bronstein *et al.* [36] demonstrate that if ϕ is permutation invariant, then $\mathbf{F}$ is permutation equivariant:

$$\phi \overset{\text{permutation invariant}}{\longrightarrow} \mathbf{F} \overset{\text{permutation equivariant}}{\longrightarrow} \tag{8}$$

2.2.3 Graph Neural Network Layers

In the context of deep learning, we recognize that GNNs implement three types of architectural layers [36].

1. *Equivariant message passing*. This permutation equivariant layer maps a representation of a graph to an updated representation, yielding the same graph (structure) [36].

2. *Local pooling*. This is a downsampling technique that reduces an input graph (with updated node representations) to a smaller sized graph or even a single node. This can result in a better generalized and more performant model [39].

[2]A *multiset*, denoted $\{\{\ldots\}\}$, allows for elements occurring multiple times. This is the case in feature graphs, as features of different nodes may be equal.

3. *Global pooling*. Also called the flat pooling or readout layer, this layer transforms a graph to a fixed-sized representation in one step [39].

We define these in the coming definitions (2.5, 2.6, 2.7), after which an example is worked out showing how information flows through the layers of a GNN.

Definition 2.5 (Message-Passing Layer Types)**.** To learn graph representations, node information is shared across neighborhoods. Going from least to most expressive power at the cost of interpretability, scalability, or learning stability, we recognize three categories of graph neural network layers [36].

$$\mathbf{h}_u = \phi(\mathbf{x}_u, \oplus_{v \in N_u} c_{vu} \psi(\mathbf{x}_v)) \text{ (Convolutional)} \tag{9}$$

$$\mathbf{h}_u = \phi(\mathbf{x}_u, \oplus_{v \in N_u} a(\mathbf{x}_u, \mathbf{x}_v) \psi(\mathbf{x}_v)) \text{ (Attentional)} \tag{10}$$

$$\mathbf{h}_u = \phi(\mathbf{x}_u, \oplus_{v \in N_u} \psi(\mathbf{x}_u, \mathbf{x}_v)) \text{ (Message-passing)} \tag{11}$$

Note that $\mathbf{h}_u$ is a node embedding for node $u \in V$, meaning the above layers are applied per node. Furthermore, ϕ and ψ are differentiable *update* and *message* functions respectively (e.g., neural networks like $\psi(\mathbf{x}) = \mathrm{ReLU}(\mathbf{W}\mathbf{x} + \mathbf{b})$). Also, $\oplus$ is a permutation-invariant aggregation function (i.e., the order of the input does not influence the output). Examples are sum, mean, median, or max.

[40–42] represent noteworthy examples of implementations of convolutional layers. The most popularized of these is the graph convolutional network (GCN) by Kipf & Welling [41]. Proceeding to examples of attentional architectures, [43–45] portray the state-of-the-art. Finally, [46–48] are illustrative exhibitions of message-passing layers.

Definition 2.6 (Local pooling)**.** Let P be a pooling operator that maps a graph $G = (V, E)$ to a pooled graph $G' = (V', E')$:

$$G' = P^{\oplus}_{local}(G) \tag{12}$$

where $|V| > |V'|$ [3] and $\oplus$ is some aggregation function. The main goal of local pooling is to reduce the number of nodes in a graph, at the same time retaining explanatory variance with respect to the prediction target [39].

Definition 2.7 (Global pooling)**.** Again, using G as a data loaded graph, global pooling is defined as:

$$Y = P^{\oplus}_{global}(G) \tag{13}$$

where Y may be a vector space of choice, depending on the task at hand. When doing a regression prediction on a graph-level attribute, this could be a scalar value, whereas with a classification task this might be vector giving a distribution of the class labels. A global pooling layer must be permutation invariant.

[3]In exceptional cases, there exists $|V| \geq |V'|$. A graph is then upscaled by pooling [39].

In the following we present an example that builds upon the OTC example (Execution A specifically, see Table 1). Onto our nodes, we attach feature vectors that include whether the event attribute `resource` refers to a warehouse team, the elapsed time (in hours), and the number of objects that an event is associated to.

Example 3: Graph Neural Network at Work

This example demonstrates the inner workings of a graph convolutional network predicting the graph-level target remaining time.

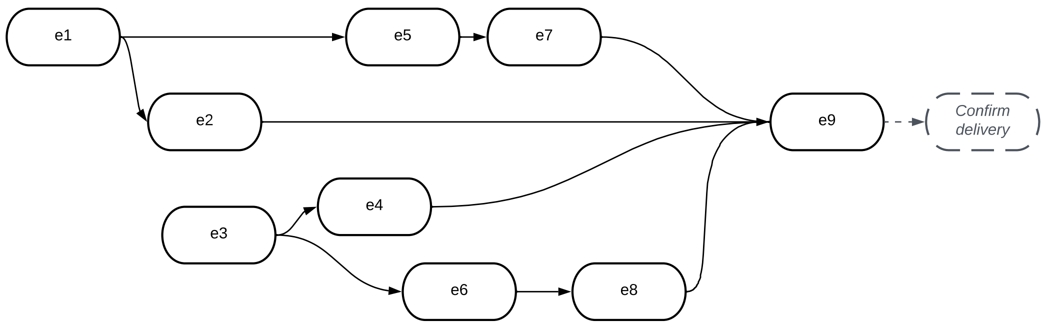

Figure 6. Object-centric directly-follows graph of Execution A (cf. Table 1). The horizontal axis represents time passing, indicating the precedence order of events (not to scale).

Representing the Data. Given the object-centric DFG of Execution A (Figure 6) and event-level features *Warehouse team*, *Elapsed time*, and *No. objects*, we would yield feature matrix **X** and adjacency matrix **A**.

$$\mathbf{X} = \begin{array}{ccc|c} \textit{Warehouse team} & \textit{Elapsed time} & \textit{No. objects} & \\ 0 & 0 & 3 & \text{e1} \\ 0 & 0 & 1 & \text{e2} \\ 0 & 1 & 2 & \text{e3} \\ 1 & 1 & 1 & \text{e4} \\ 1 & 12 & 2 & \text{e5} \\ 0 & 13 & 2 & \text{e6} \\ 1 & 13 & 3 & \text{e7} \\ 1 & 14 & 3 & \text{e8} \\ 1 & 21 & 7 & \text{e9} \end{array} \qquad \mathbf{A} = \begin{array}{ccccccccc|c} \text{e1} & \text{e2} & \text{e3} & \text{e4} & \text{e5} & \text{e6} & \text{e7} & \text{e8} & \text{e9} & \\ 0&0&0&0&0&0&0&0&0 & \text{e1} \\ 1&0&0&0&0&0&0&0&0 & \text{e2} \\ 0&0&0&0&0&0&0&0&0 & \text{e3} \\ 0&0&1&0&0&0&0&0&0 & \text{e4} \\ 1&0&0&0&0&0&0&0&0 & \text{e5} \\ 0&0&1&0&0&0&0&0&0 & \text{e6} \\ 0&0&0&0&1&0&0&0&0 & \text{e7} \\ 0&0&0&0&0&1&0&0&0 & \text{e8} \\ 0&1&0&1&0&0&1&1&0 & \text{e9} \end{array}$$

Note how each row in **X** represents an event, containing features that describe it. In the context of machine learning, this is called a *feature vector*. In a graph context, it may also be referred to as a *node vector* or *node feature vector*. Right from the feature matrix, the adjacency matrix, indicates the nodes' connectivity. The columns in **A** indicate outgoing edges. Take per example, node `e1`. It has directed arcs to `e2` and `e5`, denoted by a 1 in the corresponding column in **A** (cf. Figure 6).

Message-Passing. Kipf & Welling [41] describe the importance of adding self connections to nodes, such that information from a node is included in the same node in the next layer. This is done through the addition of the identity matrix, resulting in updated adjacency matrix:

$$\tilde{\mathbf{A}} = \begin{array}{c} \begin{array}{ccccccccc} \text{e1} & \text{e2} & \text{e3} & \text{e4} & \text{e5} & \text{e6} & \text{e7} & \text{e8} & \text{e9} \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \end{array} \begin{array}{c} \\ \text{e1} \\ \text{e2} \\ \text{e3} \\ \text{e4} \\ \text{e5} \\ \text{e6} \\ \text{e7} \\ \text{e8} \\ \text{e9} \end{array}$$

If we now multiply $\mathbf{X}$ and $\tilde{\mathbf{A}}$, we perform a 'message-passing' operation using the $\sum$ aggregator over the whole graph. The result is the following.

$$\tilde{\mathbf{A}} \cdot \mathbf{X} = \begin{array}{c} \begin{array}{ccc} \textit{Warehouse team} & \textit{Elapsed time} & \textit{No. objects} \end{array} \\ \begin{bmatrix} 0 & 0 & 3 \\ 0 & 0 & 4 \\ 0 & 1 & 2 \\ 1 & 2 & 3 \\ 1 & 12 & 5 \\ 0 & 14 & 4 \\ 2 & 25 & 5 \\ 1 & 27 & 5 \\ 4 & 49 & 15 \end{bmatrix} \end{array} \begin{array}{c} \\ \text{e1} \\ \text{e2} \\ \text{e3} \\ \text{e4} \\ \text{e5} \\ \text{e6} \\ \text{e7} \\ \text{e8} \\ \text{e9} \end{array}$$

Taking `e9` as an example, we observe the feature vectors (rows) being updated with the sum of each of its neighbors' (`e2`, `e4`, `e7`, `e8`, `e9`) features (e.g., $Elapsed\ time = 0 + 1 + 13 + 14 + 21 = 49$). From here, we can add into the equation a (learnable) weight matrix $\mathbf{W}$ and a non-linear function such as ReLU to learn better representations of our data each layer (with respect to the chosen target through the loss function). For the sake of clarity and the example's continuity, we leave this out.

Local pooling. Let us say that we have as our next layer a local pooling layer P^{avg}_{local} that coarsens a graph according to a cluster given as a parameter. Say we cluster the graph on the *Warehouse team* feature (i.e. events that are not related to a warehouse team are pooled into one node, and those that are related to a warehouse team are pooled into another node).

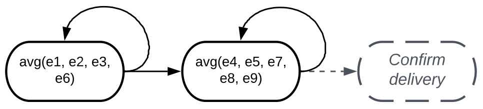

Figure 7. Local pooled graph when pooling grouped on *Warehouse team*.

Applying P_{local}^{avg} produces the graph in Figure 7, mathematically denoted and loaded with adjacency matrix $\tilde{\mathbf{A}} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ and $\mathbf{X}' = \begin{bmatrix} 0 & 3.75 & 3.25 \\ 1.8 & 23 & 6.6 \end{bmatrix}$ as the node feature matrix.

Global Pooling. We set out to predict the remaining time of running Execution A (running, since closing event `e10` is not included). Therefore, let us implement a global maximum pooling layer P_{global}^{max} such that we can return a graph-level output. Putting this into use, we generate the prediction $P_{global}^{max}(\mathbf{X}') = 23$. If we compare this against the ground truth of 24 hours (see Table 1), we achieve a mean absolute error (see. Section 5.3.2) of $\text{MAE} = |24 - 23| = 1$.

Representing a Graph Neural Network Architecture. There are several ways in which we can visualize or model GNN layers in an end-to-end fashion. In this work, we opt for a simple diagram style that shows the data flowing over arrows through matrix transformations. The architecture that has been exemplified in the previous, is modeled in Figure 8.

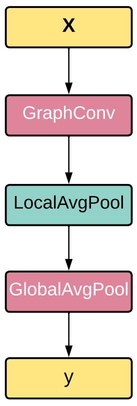

Figure 8. Architecture of the example graph neural network.

2.2.4 Heterogeneous Graph Neural Network Architectures

A graph is a very general expression of data that can embody dynamic structures. We can see images as graphs, where each pixel represents a vertex and is connected via an edge to neighboring pixels. In the same manner, text can be viewed as a graph. Here, characters or words are nodes linked via directed edges. In these examples, each node is loaded with features of the same dimension, making the graph homophilous. An even more expressive data structure is a heterogeneous graph. This allows for capturing node or edge vectors of different dimensions, meaning it can hold multiple data types. This poses a challenge for the earlier proposed GNN, as each $\mathbf{x}_u \in \mathbf{X}$ is multiplied by one weight matrix $\mathbf{W}$ (with fixed dimensions). Excitingly however, recent developments [49–52] show how to learn graph-, edge-, or node-level representations from heterogeneous graphs. In these networks, message-passing mechanisms are adapted to allow for the derivation of *update* and *message* functions (ϕ and ψ, respectively) conditioned on edge or node type. As such, information may flow between the disparate data types attached to nodes and edges in a graph.

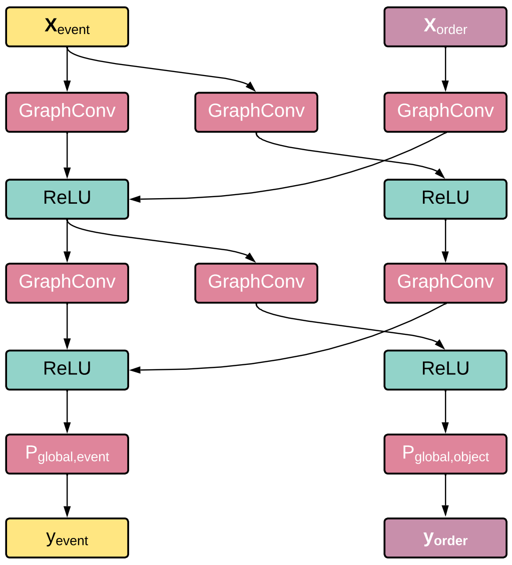

Figure 9. Exemplary heterogeneous graph neural network architecture. Adapted from [53].

Figure 9 presents us with a possible heterogeneous GNN architecture, demonstrating conceptually how information is shared between different node types. Here, the GNN architecture encompasses the two node types: **event**s and **order**s. In the figure, an example network architecture is constructed for each node type (**event** and **order**), which have different feature dimensionalities. Transformation layers are inserted to pass messages over the edges that connect nodes of type **event** with nodes of type **order**. In these layers, a linear projection is applied to each node type's features to project them into the same dimensionality. Regarding the example, it would have multiple weight matrices, $\mathbf{W}_{\mathbf{event}}$ and $\mathbf{W}_{\mathbf{order}}$, that are applied to have consistent dimensions throughout the heterogeneous GNN architecture. Through this strategy, a heterogeneous GNN can learn an optimized combination of different node (and edge) types with respect to the target of the task at hand, while considering their interaction via structural (edge) information.

2.3 Gradient Boosting Decision Trees

Gradient boosting is an ensemble learning technique that fits decision trees by minimizing a gradient loss function. Both regression and classification tasks are possible. In layman's terms, gradient boosting combines multiple weak models into one strong model. When fitting a gradient booster to data, weak models are appended iteratively, each correcting for the error of its predecessors. Each smaller estimator is fitted to the data with respect to the loss of the previous estimator(s). After a new weak model has been learned it is scaled by a learning rate to prevent overfitting. Finally, when a set stopping criterion is met, the algorithm discontinues adding weak learners and we are left with one strong model [54].

The most promising gradient boosting results were achieved using some form of decision trees (as the weak learners). In the context of predictive process mining, the XGBoost [55], Catboost [56], and LightGBM [57] models have shown to be performant, both in accuracy and scalability [58–62].

3 Related Literature

Our work emerges from discoveries, limitations, and results of prior research. Therefore, this section outlines the origins of object-centric thought, explores the current state of object-centric predictive process monitoring (OCPPM), and substantiates the relevance of this research.

3.1 Object-Centric Process Mining Lineage

3.1.1 Proclets

At the beginning of the millennium, Van der Aalst *et al.* [1] first explicitly pointed out the problem of expressing multiple data entities or objects in digital processes. At that time, multiple reports had surfaced observing issues with multiple task instances [63–68] and inter-process interactions [69]. Custom solutions were proposed to deal with the issues. Van der Aalst *et al.* [1] recognize "squeezing cases into a single process definition" as the latent cause of these problems. Example 3.1.1 demonstrates these multiple cases coexist in processes.

Example 4: Hiring Process

Consider a hiring process executed in the human resources (HR) department of an organization. In this process, several objects are at play. The object type **application** takes part in a subset of the activities included in the process. For instance, *Apply*, *Take interview*, or *Sign contract*. Another entity, **vacancy**, may have a role in events like *Open vacancy* and *Apply*. Looking at the process from a **manager**'s perspective, we can detect activities *Define application requirements* and *Sign contract*.

What we can already conclude from Example 3.1.1 is that some events are associated with multiple objects. Each of these objects can be seen as a perspective along which one can view the process. Merging these into one process model creates divergence, convergence, and deficiency issues (See Theoretical Background). Moreover, the hiring process given in Example 3.1.1 might be connected with a recruitment process within the same (or even another) organization. They could share activities or objects. For instance, a manager could be asked by the recruitment agency to give up requirements for the new hire to make a vacancy ready for publication. Here, the processes of the HR department and the recruitment agency interact.

Van der Aalst *et al.* [1] argue that workflow management systems are unable to express this, due to the lacking availability of process modeling languages and discovery algorithms at the time. To fill this gap they designed a lightweight framework, where proclets are used to model multiple objects interacting in workflows. Proclets are lightweight, single-case, processes undergirded by Petri nets describing their behavior. These are contained in a framework that enables multiple proclets to be linked through channels, over which proclets interact via performatives (structured messages).

3.1.2 Artifacts

Hull [15] surveys data-centric process modeling approaches that have become relevant by then [12, 14, 70, 71]. He presents a way of thinking about artifacts living in business processes, defining four dimensions of specifying workflows.

1. *Business artifacts* are information entities (akin to the definition in [72]) that capture business goals and allow for analyzing the progress of those goals over time. Examples include purchase order, invoice, insurance claim, and investigation dossier.

2. *Macro life-cycles* describe the evolvement of artifacts. It holds relevant stages in the evolution of an artifact. For instance, with a loan offer artifact, stages may be: created, drafted, offered, signed, rejected, or closed.

3. *Services* (or tasks) in a business process capture units of work that progress artifacts towards their goals. They exclusively affect business artifacts, i.e. all changes to artifacts take place via services.

4. *Associations* delineate the temporal constraints of the effect services can have on artifacts. Typical constraints are precedence relationships (e.g., service-service: *create item* before *pick item*; service-external event: *receive request*; service-internal event: timeout).

Cohn & Hull [16] make this an actionable framework that is ready to be implemented in a BPMS. These works do not directly address the issues of divergence and convergence. Hull [15] is mainly motivated by new requirements that emerged from implementations of ideas proposed by Nigam & Caswell [12].

Fahland *et al.* [18] then further these concepts approach by showing how to perform conformance checking on artifact-centric process models. Conformance checking is an important process mining analysis capability in which we compare formal process models to actual process executions [73]. Fahland *et al.* [74] view artifact interactions through choreographies, making it possible to analyze complex and tight interactions. Subsequently, Lu *et al.* [19] make the connection with divergence and convergence issues again, showing how artifact-centric process discovery mitigates this. They introduced a family of techniques to discover causal dependencies between artifacts, at both type and event level. Through two case studies in the industry, they made a first step towards a full discovery of artifact-centric process models from relational data sources. Taking the artifact-centric approach they generated valuable insights on the analyzed processes. They focus on the interaction between processes, where they use a single case notion per process at hand. In practice, this results in large and complex models that do not visualize the overall process in one figure. Rather, collections of interconnected process models with various case identifiers are analyzed.

3.1.3 Objects

Li *et al.* [2] suggest Object-Centric Behavioral Constraint (OCBC) models to tackle this problem. OCBC models incorporate different objects into a single diagram, faithfully modeling the relationship between data and behavior. Furthermore, it facilitates popular process mining features such as extraction, discovery, conformance checking, and performance analysis, making OCBC a potent solution to the multi-entity problem within process mining [20]. OCBC models still tend to become highly complex and the corresponding discovery and conformance checking techniques do not prove to be truly scalable. Additionally, the data format supporting OCBC models, XOC, is not exactly compact [20]. For this reason, OCBC models do not meet the usability and scalability requirements of the industry, which is the target audience in the end.

To that end, StarStar and Multiple Viewpoint (MVP) models were developed [22, 75]. These are much more scalable as the relationships between the objects are not computed and displayed in the process model and object life-cycles are not captured as detailed as in OCBC models. MVPs allow for conversion to classical event logs by selecting a view. This approach is similar to features developed by commercial PM tool vendors like Celonis[4], IBM[5], and Mehrwerk[6] that allows importing multiple single case notion event logs. Here, every event is still related to one object. The problem with approaches like these is that, in reality, the same event may refer to multiple objects and there is always a trade-off when choosing only one. Since this choice influences the frequencies shown in the discovered model, one cannot consistently guarantee correct process statistics.

Extending these developments, in 2020, OCEL[7] was put forth as a new object-centric event log standard [23]. Accompanying this, Van der Aalst & Berti [5] provide an object-centric process discovery technique that is (colored) Petri net-based. Their approach set a foundation that enables scalability and usability suited for process mining in practice, as shown by their implementation in `PM4Py`[8].

From here, a wide range of process mining capabilities have been ”reinvented” and extended. Van der Aalst [76] recognizes six capabilities, or types, of process mining.

1. *Process discovery* entails the challenge of uncovering a process model, based on event logs.

2. *Conformance checking* assesses the extent to which a predefined model and a given log correspond.

[4]A leading process mining software vendor. Their tool is called Celonis Execution Management System. See `https://www.celonis.com/ems/platform/`

[5]A tech giant that bought Italian process mining startup myInvenio, which it now integrates into their product suite. See `https://www.ibm.com/products/process-mining`

[6]Founded in 2008, Mehrwerk is one of the earliest vendors selling process mining software. The company has developed MEHRWERK ProcessMining (MPM) on the BI platform Qlik Sense to offer comprehensive Process Mining capabilities to enterprise customers. See `https://mpm-processmining.com/en/execution-suite/`

[7]The respective paper was published in 2021, but the standard was published in 2020. See `https://ocel-standard.org/`

[8]An open source Python library for process mining, found at `https://pm4py.fit.fraunhofer.de/`

3. *Performance analysis* deals with finding root causes of performance-related issues by analyzing indicators such as waiting times, response times, and service times.

4. *Comparative process mining* is mainly used for comparing event logs by a certain window for the goal of inter- or intra-organizational learning and benchmarking. Examples of windows may include locations, periods, or categories of cases.

5. *Predictive process mining* uses past behavior and statistics to answer ML questions. This can result in time and cost savings by anticipating operational issues.

6. *Action-oriented process mining* turns diagnostics into actions to improve operations.

The first three of these are enabled in an object-centric setting by works like [5, 8, 10, 11, 31, 77, 78]. Celonis has announced their tool Process Sphere which promises to feature fully object-centric process mining capabilities. Integrated with their action-oriented functionalities, this empowers the industrial impact of object-centric process mining [24, 79]. Mehrwerk and IBM are also working on integrating object-centricity into their platforms, but are less overt about this.

Advancements have also been made in the field of predictive process mining. These are discussed in the subsequent section.

3.2 Predictive Object-Centric Process Monitoring

Concerning predictive machine learning tasks performed on object-centric event logs, the current process mining literature provides several innovative methods. The following outlines the extant research on predictive process monitoring that uses object-centric event data as a starting point. We recognize a division between non-native and native object-centric predictive process monitoring approaches. The former uses an interfacing layer between the OCEL and traditional PPM methods to cope with the challenges that OCELs present. The latter applies techniques to directly learn and predict from OCEL-based features. Works in these categories are summarized in the coming paragraphs and subsequently characterized.

3.2.1 Single Case Projection Approaches

A first attempt to do any predictive task on OCEL data was by Rohrer *et al.* [26] in 2020. They propose an approach using a Generative Adversarial Network that leverages OCELs by including object attributes into event vectors. They used two synthetic datasets: the *RoboCup Logistics League* factory logistics simulation and the *OTC example* process, also used in [5, 23, 29]. Taking remaining events as their prediction target they performed predictions per selected object type. Importantly, they demonstrated that including object attributes of other (related) objects improves prediction results. Limiting to this work is that the datasets are relatively small, synthetic, and not both publicly available. Besides that, structural features that OCEL introduces were not leveraged or coped with. That is, the graph or sequential structure of process data was not specifically included in the prediction model via engineered or learned features. The state of events was encoded through one-hot encoding of activity names, losing context of where an event is in a trace. Lastly, to cope with many-to-many relationships between objects and events in the data, Rohrer *et al.* [26] project the log onto a single case by selecting one object type. This enables their addition of object attributes in predictive process monitoring. Most notably, their results indicate that the inclusion object attributes aids model accuracy.

Galanti *et al.* [28] come from a business perspective, taking various key performance indicators (KPIs) as target variables. The main innovation of their approach is the inclusion of object interactions through aggregations. They suggest taking viewpoints on the OCEL per object type and defining KPIs of interest per viewpoint. The process at hand here operates within a large Italian utility provider and is akin to an OTC workflow. Their prediction procedure starts with defining a KPI associated with an object type. Then, the OCEL is first unfolded along **order**, similar to the concept of viewpoints proposed by Berti & Van der Aalst [22]. Next, the obtained object-centric-like event log is enriched with aggregation features that capture a derivative of object interactions. They propose four interaction feature types, which we have abstracted as the following.

1. *Aggregated previous object attribute*. This feature aggregates a selected object attribute (and respective object type) until the current event.

2. *Conditionally aggregated previous object attribute*. For categorical object attributes they employ specific conditional functions with percentage count as the aggregate function.

3. *Event object type count*. A count of distinct objects of a selected object type associated with the previous (or current) events in the current process execution.

4. *Aggregated object type activity relation*. A feature mapping an aggregation function over the event long per given activity and object type. For each object (of selected object type) in previous events (including the current), it aggregates on a given activity. See Table 5 for an implementation with percentage as the aggregator.

To explain these features, an example is given, putting to display the innovations of Galanti *et al.* [28].

Example 5: Galanti's Features at Work

In this example, Table 5 implements each of the four interaction feature types on our running example. `Avg No. stops` is an implementation of *Aggregated previous object attribute*, `% normal delivery route length` of *Conditionally aggregated previous object attribute*, `No. items` of *Event object type count*, and *Aggregated object type activity relation* is instantiated in `item, % Pack item`.

Table 5. Object-centric event feature table demonstrating custom instances of features suggested by Galanti *et al.* [28]. *Note* how both Execution A and B are present (cf. Table 1, 2, and 4).

ID	Activity	Time	Res…	Order	Item	Package	Delivery	Avg No. stops	% normal delivery route length	No. items	item, % Pack item
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
e1	Place order	2023-01-30	Clo…	{o1}	{i1, i2}	{}	{}	0.0	0.0	2	0.0
e2	Pay order	2023-01-30	Clo…	{o1}	{}	{}	{}	0.0	0.0	2	0.0
e3	Place order	2023-01-30	Clo…	{o2}	{i3}	{}	{}	0.0	0.0	3	0.0
e4	Pay order	2023-01-30	Clo…	{o2}	{}	{}	{}	0.0	0.0	3	0.0
e5	Pick item	2023-01-31	War…	{o1}	{i1}	{}	{}	0.0	0.0	3	0.0
e6	Pick item	2023-01-31	War…	{o2}	{i3}	{}	{}	0.0	0.0	3	0.0
e7	Pack item	2023-01-31	War…	{o1}	{i1}	{p1}	{}	0.0	0.0	3	33.3
e8	Pack item	2023-01-31	War…	{o2}	{i3}	{p2}	{}	0.0	0.0	3	66.7
e9	Ship package	2023-02-01	War…	{o1, o2}	{i1, i3}	{p1, p2}	{d1}	5.0	0.0	3	66.7
e10	Confirm delivery	2023-02-02	Pos…	{o1, o2}	{i1, i3}	{p1, p2}	{d1}	5.0	0.0	3	66.7
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
e21	Ship package	2023-02-05	War…	{o3}	{i4}	{p3}	{d2}	27.0	100.0	3	100.0
e22	Ship package	2023-02-05	War…	{o3}	{i4, i5, i6}	{p3, p4}	{d3}	22.5	50.0	3	100.0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

The approach comes with several limitations. An obvious one is that it is unable to handle object attributes when the attribute refers to an object type that has multiple object instances in an event. For instance, in Table 5, `e9` contains two object references of type **package**. This implies that **package** attribute `weight` cannot be included in the event feature table, as we are not able to determine whether we refer to `p1` or `p2`. Another limitation is that activity relations are not captured by any established PPM trace encoding (e.g., aggregation, index encoding [80]) or bucketing technique (e.g., clustering [81], prefix-length [82], transition system [83]) that embed context. Importantly, they demonstrate that their interaction features do improve prediction quality when compared to flattened and object-centric-like event logs that do not include these features.

Lastly, the most recent work on object-centric PPM took on the problems of next activity, next event time, and remaining time prediction [29]. This work operated on one, rather small dataset, the *OTC example*. This OCEL was also used for prediction by Rohrer *et al.* [26] and as a sample log in [5, 23]. Gherissi *et al.* [29] posit their work as a conservative initial attempt to take on the challenge of object-centric predictive process monitoring. They take a weak cognate of object interactions into their approach. That is, they first flatten the OCEL based on a selected object type. Next, categorical variables (including activity) are one-hot encoded, disregarding established trace encoding or bucketing techniques. Subsequently, per event, a count of related objects in previous events (of the current trace) is added as an enrichment feature, akin to the *Event object type count* feature by Galanti *et al.* [28]. After that, to cope with the varying number of object references per event, they define a maximum trace length, which becomes the fixed input shape for a long short-term memory (LSTM) architecture. This LSTM then performs the said prediction tasks. Already evident from their work is that learning from object relations does lift predictive performance.

3.2.2 Native Object-Centric Approaches

Recognizing the limitations of conversion from case-agnostic logs to single case notion or viewpoint event logs, Adams *et al.* [27] postulate a framework for extracting and encoding features from OCELs. The work extends [84] to an object-centric setting by defining how to calculate their suggested features for OCELs, presenting new features (defined in [10]), and producing a Python library that enables the application of their ideas in practice on any OCEL. Utilizing an object-centric case definition, either connected components or leading type (Def. 5 and 6 in [8], respectively), they extract object-centric process executions in the form of directed event graphs. To each node in the graphs are attached equal-length feature vectors. They describe either characteristics of the event in relation to previous events in its process execution, or the whole log up to that event's time. In total, they list 27 features that can be grouped into five perspectives: **C**ontrol-Flow, **D**ata-Flow, **R**esource, **P**erformance and **O**bjects. Figure 10 illustrates what such a graph-based feature structure looks like for our running example (the *Ship package* event specifically).

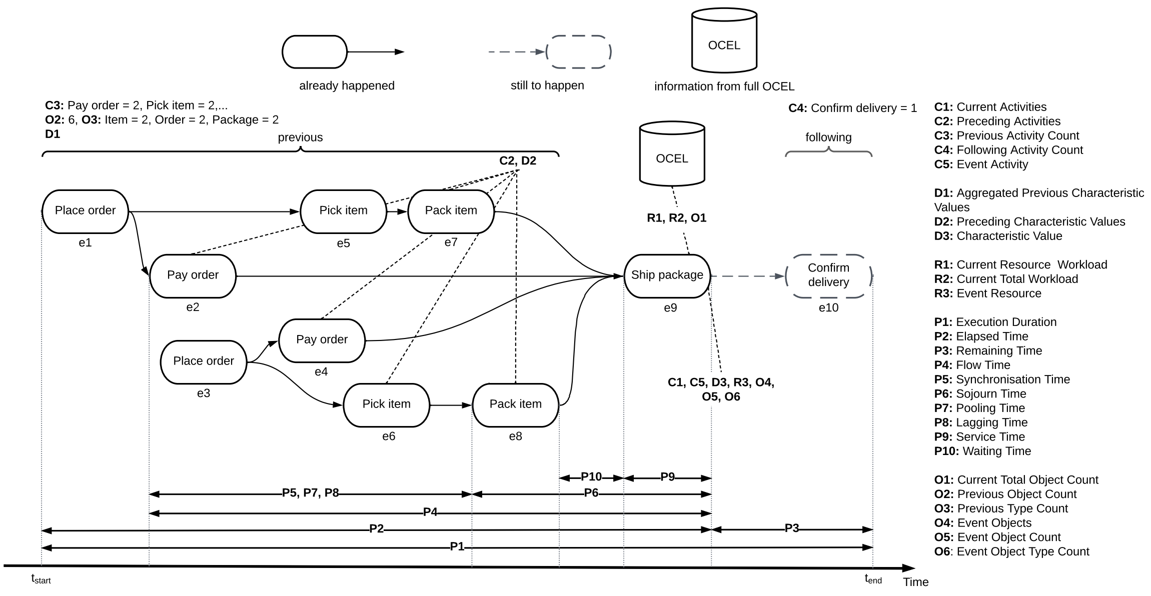

Figure 10. Object-centric features for event-level extraction proposed by Adams *et al.* [27] (cf. Fig. 4 in [27]). Features that can be extracted for event `e9` are illustrated (see Table 1 for the log).

From here, three encodings may be generated. The feature graph structure, as it stands. A sequential structure, in which each feature graph is collected into a sequence, where each event node is only connected to its preceding and succeeding event nodes (based on the complete timestamp) (Def. 7, [27]). And a tabular encoding, which does not take any constraints pertaining to the connectivity or ordering of nodes, generates a feature matrix by concatenating all event vectors (in arbitrary order, Def. 6, [27]). The graph-based encoding is their main innovation. The others were already known in PPM. They assess their work on a loan application event log, which is a classical one that has been transformed to an OCEL [85]. For each of the encodings, two use cases are given, one performing remaining time prediction and one demonstrating visualization capabilities. Crucially, using Shapley values [86] and model performance results, they illustrate that information is increasingly lost going from graph-based to sequential and finally tabular encoding. Limiting their observations is the fact that hyperparameter tuning of the models used for evaluation was not part of the published research. Linear regression was used to model the tabular data structure, and default LSTM and GCN architectures were used for sequential and graph-based encoding respectively. Moreover, the model evaluation scores are presented in their standardized form, making differences (per encoding/model) difficult to interpret. A consequence of this is that we cannot rule out whether the tabular structure is still inferior to sequential and graph structures when another model is employed. Even so, this is an essential work in object-centric predictive process monitoring, as it enables object-centric trace encoding for machine learning based on OCELs. Not to mention, their approach is replicable, highly configurable, and applicable for future work in both academic and industrial spheres through `ocpa`[9], a Python library they published alongside it [87]. In the remainder of this thesis, we may refer to their approach as EFG, which stands for event feature graph.

Berti *et al.* [30] extend the possibilities for machine learning on OCELs by introducing the concept of object-level feature extraction via object graphs (cf Def. 4 in [30]). In a general sense, object graphs capture the interaction objects have through events. They give three example graphs that can be useful.

1. *Object interaction graphs* that connect pairs of objects if they co-occur in some event of the log. Such an undirected graph could be used to derive features capturing expected relations (e.g., every invoice should be connected to an order).

2. *Object creation graphs* that link objects via directed edges if one already exists and the other starts to exist in a certain event. As an example, features capturing a temporal ordering could be calculated from here (e.g., payment before invoice).

3. *Object continuation graphs* relate objects via directed edges that terminate their life-cycle with the given event to all objects that start their life-cycle in that event. In the context of HR processes, we could extract a feature about objects being birthed whenever a contract offer has been signed (e.g., the number of objects in the onboarding process triggered).

Consequently, examples of feature maps are given to extract useful information as input for prediction or anomaly detection algorithms. They demonstrate in their visual tool called OCPM[10] an implementation of anomaly detection per object via isolation forests [88]. Furthermore, they have extended `PM4Py` to enable their feature extraction methods in Python. This library furthers these ideas through *object co-birth* and *object co-death graphs*.

[9]An open source Python library for object-centric process analysis, found at `https://github.com/ocpm/ocpa`

[10]OCPM is an object-centric process mining web application that supports discovery, conformance checking, and exploration techniques on OCELs. See `https://www.ocpm.info/`

These undirected graphs connect objects whose life-cycle is started and ended in the same event, respectively. It gives rise to even more graph-based features for object-centric data.

3.3 Synthesis and Current Limitations

When synthesizing these related works, we observe that OCEL and the corresponding discovery techniques strike a balance between accuracy in capturing a process' behavior and scalability in algorithmic model discovery. This also results in a model being more easily interpreted and analyzed by process analysis practitioners. Concerning predictive monitoring, this implies that less information is captured when comparing OCEL to XOC or artifact-centric data. However, when dealing with large amounts of data, this is likely nullified. Adding to this, OCEL allows for a more scalable ML pipeline as discovery techniques are involved in feature extraction and encoding, making lighter underlying data formats critical.

Regarding the current state-of-the-art artificial intelligence based on object-centric event data, we observe promising results. Three works argue that OCELs allow for better predictions in predictive process monitoring when compared to traditional event logs [26, 28, 29]. Interestingly enough, these efforts all exhibit some form of flattening, incorporating lossy data preprocessing techniques. Therefore, more attention is drawn to the two approaches that enjoy OCEL-native feature extraction techniques [27, 30], meaning OCELs are not flattened along an object type during preprocessing or feature extraction. Both these works take a more methodic approach, describing general features that can be applied to any OCEL. They can be divided into two perspectives (that have not been compared yet). Adams *et al.* [27] design their features on an event basis, enriching event vectors. Berti *et al.* [30], on the contrary, extract features per object, generating object vectors. Both give a generic open source Python implementation, bringing true object-centric predictive process monitoring closer to industrial application.

Finally, to structure the discussion on the related works in OCPPM, we characterize each approach along several dimensions (see Table 6). The table is horizontally divided into native and non-native approaches. These works are then characterized on six dimensions pertaining to how the approaches encode different information contained in OCELs. *Encoding Granularity* indicates what perspective is taken on the OCEL. We can either take the object tables (cf. Table 2) and enrich each row with information from the events table (cf. Table 1), or vice versa. *Process Behavior Encoding* relates to how the activities in the event log are encoded, such that context (where an activity is in the process) is considered. *Event Attribute Encoding* concerns the way an approach deals with event attributes. *Event-object Interaction Encoding* refers to what features are proposed to incorporate the interaction between events and objects. *Object-object Interaction Encoding* is a dimension that considers relations between objects as features (see the discussion on Berti *et al.* [30] for examples). Lastly, *Object Attribute Encoding* pertains to how an approach handles object attributes (e.g., are they allowed by the approach? And, to what extent did the approach take in object attributes?).

Table 6. Characterization of the related works in predictive process monitoring on object-centric event logs.

Approach		Encoding Granularity	Process Behavior Encoding	Event Attribute Encoding	Event-object Interaction Encoding	Object-object Interaction Encoding	Object Attribute Encoding
Native	Adams *et al.* [27]	Event-level	OC-DFG as data structure	Event-based features[1] Execution-based features	Current total object count Previous object count Event objects Event object count Event object type count	None	None
	Berti *et al.* [30]	Object-level	None	None	Activity count[2] Related events count[2]	Object graph features	None[3]
Non-native	Rohrer *et al.* [26]	Event-level	One-hot	None[1]	None	None	Limited[4]
	Galanti *et al.* [28]	Event-level	CatBoost algorithm	Categorical event attributes	Aggregated previous object attribute Conditionally aggregated previous object attribute Event object type count Aggregated object type activity relation	None	Limited[5]
	Gherissi *et al.* [29]	Event-level	One-hot	None[1]	Event object type count	None	None

[1] Event attributes were not mentioned, but are allowed by the approach.
[2] This feature may also be viewed as an encoding of process behavior but from an object-level encoding point of view.
[3] Berti *et al.* [30] did not include any object attributes in their work, however, their approach does support object attribute integration.
[4] Only object attributes of the object type that is selected as the case notion are included.
[5] Only for object types that are referred to once per event at maximum.

Upon analyzing Table 6, we observe that none of the approaches take full advantage of the six dimensions presented. We see that a choice in one dimension can exclude options in the other dimensions. For instance, Adams *et al.* [27] take an event-level perspective (*Encoding Granularity*), excluding the possibility to encode object-object interactions or object attributes without aggregation techniques (as in [28]). The approach by Berti *et al.* [30] on the other hand, can encode this without compromise due to its object-level view on OCELs, but it suffers on the *Process Behavior Encoding* dimension. Taking the object tables as a basis, a lossless encoding of events (process behavior) is inaccessible.

These limitations are produced by the inherently complex structure of OCELs. That is, as the metamodel in Figure 2 illustrates, objects and events have a many-to-many relationship. This means that an event may refer to multiple objects and an object may refer to many events. When preparing feature matrices for ML algorithms, we usually need to define fixed dimensions. Each vector (row) must have the same length (amount of columns).

When creating a feature vector per event (event-level granularity), the event vector can refer to multiple objects that have attributes. To obtain an equal size vector for each event, either none, a manually filled, or an aggregate of the object attributes (or other object-based features) may be taken into the vector. An aggregate could for example be the mean of a certain object attribute per object type.

Contrarily, when taking an object perspective, one creates a feature vector per object. Each object refers to one or more events. Thus, to acquire a fixed-size feature matrix, we can take either none or an aggregate of each of those events referred to. Examples of aggregates would be the maximum value of a certain event attribute (for events related to the respective object) or a count of related events (like in [30]).

In both perspectives, we experience a loss of information due to either the unavailability of attributes or taking aggregates. Besides this, we can note that relational information is lost of either

1. object-object interactions (via an object graph) when taking the event perspective, or

2. event-event interactions (via the OC-DFG) when taking the object perspective.

Concluding this chapter, we see that the current efforts in object-centric predictive process monitoring are promising, but have limitations that motivate either deserting OCELs by flattening them into traditional event logs or remaining object-centric and taking aggregates of features contained in OCELs. In reference to our research objective (cf. Section 1), this thesis sets out to design an encoding approach that captures object-centric event data without flattening events (Definition 2.4), filling unavailable object features, or performing aggregations on objects (as in [28] or features **O1-O6** in [27], see Table 5 and Figure 10 respectively). The observed limitations in related literature lead to a set of requirements that incentivize a more comprehensive, native (i.e. without flattening on object type) approach to predictive tasks on OCELs that handles events *and* objects without taking aggregates. In the following section (Section 4), we outline these requirements and present our proposed approach that incorporates these requirements, satisfying the potential of OCEL for predictive process monitoring by providing full support for the six dimensions presented in Table 6.

4 Approach

In the previous section, we reviewed the current literature in the field of process mining and predictive process monitoring in an object-centric setting. This review revealed crucial insights into the state of the art and the direction in which advancements have been made. However, it also unearthed notable limitations that need addressing. They therefore serve as pivotal guideposts for the formulation of our novel solution.

First, we reify these limitations into requirements or characteristics to assess a solution. Following this, we introduce our approach. We elucidate our proposed idea on an instance level, based on the running example both conceptually and applied to deep learning. Concluding, we present a capability assessment of our approach.

4.1 Assessment Criteria for an Object-Centric Feature Encoding Approach

Object-centric process mining has emerged as a reaction and solution to flattening event logs in traditional process mining. Therefore, this work's goal is to extend the elimination of flattening procedures to predictive process monitoring by obviating flattening on events, filling unavailable object features, and aggregating object attributes. Section 3.3 shed light on how current PPM approaches have utilized OCELs for prediction tasks, demonstrating that information loss still occurs in both *non-native*[11] and native approaches. This was shown to be caused by choosing an *Encoding Granularity*, either event-level or object-level. Taking one perspective excludes the possibilities and benefits that come with the other perspective, and vice versa. Optimally, one wants all options on all six dimensions presented in Table 6 to be taken up into one encoding. The list of options derived from the table can be summed up in the following six assessment criteria by which we can assess the encoding capabilities of an object-centric predictive process monitoring approach.

C1. *Encoding Granularity*: event-level and object-level.

C2. *Process Behavior Encoding*: Execution graph structure, feature-based trace encoding, and execution-based features.

C3. *Event Attribute Encoding*: Numerical attributes, numerically encoded categorical attributes, and event-based features.

C4. *Event-object Interaction Encoding*: Any calculable interaction feature.

C5. *Object-object Interaction Encoding*: Object graph structure (and features).

C6. *Object Attribute Encoding*: Numerical attributes, numerically encoded categorical attributes, and object-based features.

[11]Approaches that flatten the OCEL on an object type to get a traditional event log.

A solution that aligns with our research objective then, should not be confined to either an event-level or an object-level perspective, but have both perspectives. When not making a choice on *Encoding Granularity*, but encoding both objects and events in their original (graph) structure, one can facilitate all six dimensions to the fullest. The proceedings of this chapter give an exposition of our novel solution that stays true to the data structure presented in the original OCEL.

4.2 Heterogeneous Object Event Graph Encoding

We propose a heterogeneous object event graph encoding (HOEG). This encoding incorporates both event-level and object-level perspectives into a graph with different node (and edge) types. In a heterogeneous graph, each node type may have its own shape, equal to how OCELs contain data entities that each have a different shape. This factoid can be derived from the metamodel in Figure 2 and directly seen on an instance level in Tables 1 and 2 in Example 2.

4.2.1 Conceptual Design

To illustrate this heterogeneous graph encoding, let us revisit Execution A from the OTC example given in Table 1. Given the events and objects present in this table, Figure 11 visualizes its heterogeneous object event graph structure.

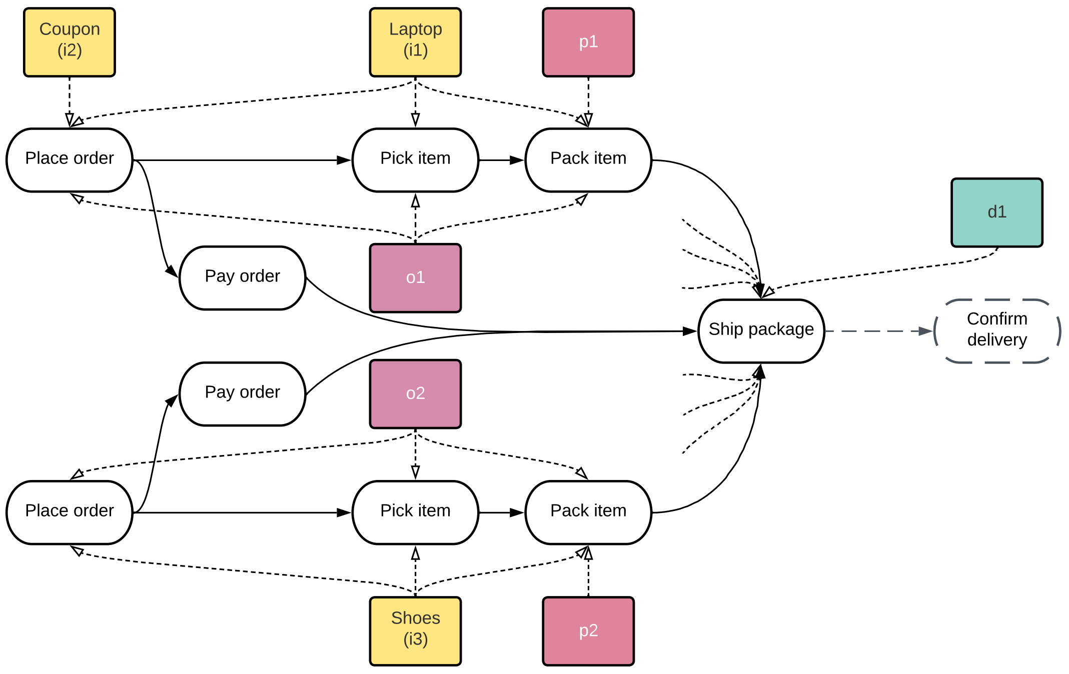

Figure 11. Heterogeneous object event graph for Execution A (derived from Table 1) of the running OTC example. *Note* the trimmed dotted edges going out from *Ship package*. They are connected with objects `o1`, `o2`, `i1`, `i3`, `p1`, `p2`, but trimmed for readability considerations. The dashed arrow and activity node are used to signify future event *Confirm delivery*.

As depicted, the encoding uses an object-centric discovery technique to construct the OC-DFG (like in [27]), to which it attaches object information. The event node type has an oval shape and is linked to the objects that it refers to in the events table of the OCEL (Table 1 in this case). Object types are separate node types in the graph structure. These different node types are denoted by the node colors per object type. In the illustration, there are five node types in total: event nodes, order nodes, item nodes, package nodes, and delivery nodes. Generally speaking, the HOEG encoding has a node type per object type and one node type to encode events.[12]

Contrasting HOEG to EFG via Figure 11 and 10 respectively, it becomes evident that HOEG explicitly encodes objects and their features, whereas EFG resorts to aggregating object information into event features (**O1-O6**, Figure 10).

4.2.2 Deep Learning on the HOEG Encoding

The HOEG encoding structure lends itself to heterogeneous graph neural network architectures when performing deep learning. In such architectures, we can define different edge relation types over which the message-passing mechanism transfers information during a forward pass of a neural network. It is through this that we are able to learn on both event and object features simultaneously. The network learns how to best combine and transform the information from the different node (or edge) types to predict a certain target.

From a graph neural networks perspective, the HOEG encoding can be understood as a set of semantically sensible node and edge type names, according adjacency matrices, and feature matrices. Building upon the visual representation introduced in Figure 9, we now detail the underlying data driving the visualization. Doing so yields a better understanding of how a heterogeneous graph neural network architecture would learn patterns from HOEG.

Consider a HOEG instance based on Execution A (cf. Tables 1 and 2), including (numerically encoded) event attributes, object attributes, event-based feature *Elapsed time*, and implicitly encoded object-object interactions (via events, as seen in Figure 11).

This would have the five node types: `event`, `order`, `item`, `package`, and `delivery`. Besides that, it contains edge type triples:

- `(event, follows, event)`

- `(order, interacts with, event)`

[12]`https://github.com/TKForgeron/OCPPM` hosts an open source implementation of HOEG as a general technique.

- `(item, interacts with, event)`
- `(package, interacts with, event)`
- `(delivery, interacts with, event)`
- `(order, updates, order)`
- `(item, updates, item)`
- `(package, updates, package)`
- `(delivery, updates, delivery)`

Each of these edge types has a separate adjacency matrix indicating the heterogeneous object event graph's connectivity. The latter four edge types are not visible in Figure 11. They are introduced here to facilitate the inclusion of self-loops, where edges connect a node to itself. This practice is a standard technique in graph deep learning. By enabling self-loops, the GNN can enhance its node representations. This enhancement is achieved through the message-passing mechanism, which allows nodes to exchange information with themselves across multiple layers.

Furthermore, each node type has a distinct feature matrix with the following[13] shapes:

- $(9, 2)$ for `event` with features *Resource* and *Elapsed time*.
- $(2, 1)$ for `order` with feature *Price*.
- $(3, 1)$ for `item` with feature *Discount*.
- $(2, 2)$ for `package` with features *Weight* and *Size*.
- $(1, 2)$ for `delivery` with features *Route length* and *No. stops*.

Additionally, HOEG allows for encoding interaction information as edge features, which would yield another nine feature matrices at maximum (one for each edge type).

[13]Where *Resource*, *Size*, and *Route length* are numerically encoded using some categorical encoding technique.

4.2.3 HOEG Encoding Assessment

We assess the capabilities of the HOEG encoding conceptually in Table 7 through the assessment criteria given in Section 4.1. Here, we observe that HOEG is a flexible and expressive encoding approach as it embraces the inherent structure of OCELs as its data structure. Note the addition of *event-object interaction graph structure*, a novel feature.

Table 7. Heterogeneous object event graph encoding capability assessment.

Criterium	Support	Remark
C1. Encoding Granularity		
Event-level	Full	
Object-level	Full	
C2. Process Behavior Encoding		
Execution graph structure	Full	
Feature-based trace encoding	Full	When traces are encoded using features, they can be appended as event features to event nodes.
Execution-based features	Limited	May be used as target variable. When used as predictor variable, it is duplicated across each event node in an execution graph.
C3. Event Attribute Encoding		
Numerical attributes	Full	
Numerically encoded categorical attributes	Full	
Event-based features	Full	
C4. Event-object Interaction Encoding		
Event-object interaction graph structure	Full	The interaction between events and objects is naturally encoded into its graph structure. Not seen before in related works.
Event-object interaction features	Full	May either be appended as node features (to either event or object nodes), or as edge features.
C5. Object-object Interaction Encoding		
Object interaction graph structure	Full	Any interaction discovery algorithm presented by Berti *et al.* [30] may be used here to create direct object-object edges. It is otherwise also possible to let object-object interaction be implicitly encoded via the event-object interaction graph structure.
Object interaction features	Full	May either be appended as object node features, or as edge features if direct object-object edges are present.
C6. Object Attribute Encoding		
Numerical attributes	Full	
Numerically encoded categorical attributes	Full	
Object-based features	Full	

In order to evaluate the viability of the HOEG encoding for predictive process monitoring on object-centric event logs, we run experiments on three datasets. The following chapters elaborate on this.

5 Experimental Setup

Drawing upon the previous chapters, this section sets a structure for evaluating the proposed HOEG encoding. First, we distinguish EFG as a graph-based encoding against which the HOEG encoding is mainly compared. Then, both EFG and HOEG are elaborated upon in terms of configuration. Second, graph neural network architecture design choices are given and the process of tuning hyperparameters is outlined. Third, evaluation methods are presented by which the graph-based feature encodings are compared and analyzed. Fourth, the object-centric event logs used in the experiments are described in terms of preprocessing and content. Lastly, we provide an overview of the workflow of the experiments in a modular fashion.

5.1 Feature Encodings

Taking into account Chapters 3 and 4, we recognize two perspectives concerning object-centric feature extraction: object-level and event-level feature encoding. Section 4 introduced us to an approach that aims to solve the limitations of taking either perspective by integrating the object- and event-level views into one heterogeneous object event graph encoding. Consequently, to evaluate the efficacy of this integrated approach, it must be juxtaposed against its single-perspective counterparts that emphasize either objects or events individually. However, event-based prediction tasks have been the focus of predictive process monitoring as an academic field, and object-based predictions are not yet established in research. Besides that, from an objects perspective, remaining time cannot be a target variable. Therefore, we evaluate our proposed HOEG encoding predicting on an event level only. The upcoming subsections elaborate on the configuration of the EFG and HOEG encodings.

5.1.1 Event Feature Graph Configuration

Regarding the configuration of the EFG encoding for our experiments, we initialize the graphs using a feature set nearly identical to Adams *et al.* [27]. That is, we include in the event feature vectors a one-hot encoding of *Preceding Activities* (**C2**), the *Elapsed Time* (**P2**), the *Synchronization Time* (**P5**), and the *Previous Type Count* (**O3**). Comparing this to Adams *et al.* [27], we recognize that *Aggregated Previous Characteristic Values* (**D1**) has been dropped, and **P5** has been added. **D1** has been removed due to it requiring a flattening procedure (aggregation), which is in their use case specifically caused by the adoption of an object attribute as an event attribute[14]. Furthermore, we include event attributes. Whenever an event attribute is a categorical variable, we encode it using the count encoding technique, which replaces the names of the groups with the group counts. Finally, we employ the graph-based feature structure. Primarily to stay as true to the original data structure as possible, but also as the literature recommends so. That is, Adams *et al.*

[14]`RequestedAmount` in the loan application log, to be exact.

[27] have shown promising results using the graph-based encoding and Galanti *et al.* [28] suggest exploring graph-based approaches for predictive analytics on object-centric processes. Ultimately, this encoding type results in homogeneous graphs representing process executions, where each node holds a feature vector that describes one event.

5.1.2 Heterogeneous Object Event Graph Configuration

The most natural way of incorporating both object and event perspectives is a heterogeneous graph as described in the previous chapter (Section 4). When employing this encoding, there are many configuration options, demonstrated by Table 7. In our HOEG setup, event node types exactly correspond to the EFG encoding configuration specified in Section 5.1.1. Event-object interaction is encoded in the structure only (not as features). Object-object interaction is implicitly encoded via event-object edges. Objects are never direct neighbors in our HOEG configuration. Finally, object node types contain feature matrices based on numerically encoded object attributes only.

5.2 Models and Hyperparameters

To leverage the multi-dimensional information contained in OCELs, we utilize graph-based deep learning models for each of the given scenarios to demonstrate how the graph-based models exploit their respective structures.

First, we design our model architectures based on qualitative graph neural network knowledge. When performing predictions based on event data, the context of an event is of high importance. This is demonstrated by the abundance of research into ways of capturing the state of an event as either features (i.e. trace encoding techniques [89]) or as an inherent mechanism of a predictive model (annotated transition systems [83]). Therefore, we employ k-dimensional GNN layers proposed by Morris *et al.* [90] in our architecture design. Their message-passing layer captures higher-order graph structures at multiple scales. Higher-order graph structures refer to patterns or relationships that exist between groups of nodes in a graph, beyond merely the pairwise connections between individual nodes. This fits well with capturing the state of an event given an execution graph.

Then, via guidance from literature, several hyperparameters are fixed to recommended values and two are chosen to undergo a tuning process to find the best value within a specified range.

The first, the number of *hidden dimensions* (*hd*), is optimized across eight settings. The higher the number, the more complex patterns the model could potentially detect. The disadvantage of increasing this number is that it entails increased model complexity, which results in lower scalability. Also, at some point increasing *hd* no longer decreases model error, as more complex patterns might not be present in the data or might not generalize to unseen data. The number of *hidden dimensions* is tuned because it enables insight into learning capability facilitated by different encodings.

The second hyperparameter that undergoes tuning is the *learning rate* (*lr*). This determines the step size at which the model's parameters are updated during training via the specified optimizer function (Adam in this case) [91]. *Learning rate* influences convergence speed and performance. A lower *lr* might contribute to a slower convergence but could cause the optimizer to find better optima. We tune this hyperparameter as it may give insights into an encoding's overall stability.

Both hyperparameters can interact. For example, it is possible that a combination of a steep *learning rate* and a high number of *hidden dimensions* proves to be optimal, while in contrast, only a high number of *hidden dimensions* results in more error.

In Table 8, we outline the hyperparameters that are considered. Each row indicates the selected value or tunable value range.

Table 8. The hyperparameters used in the experiments.

Hyperparameter	**Value(s)**	**Source(s)**	**Remark**
Batch size	16		Dependent on available resources.
No. epochs	30	Adams *et al.* [27]	
Early stopping criterion	4		Dependent on available resources.
No. pre-message-passing layers	0	Adams *et al.* [27]	We use preprocessed features.
No. message-passing layers	2	Adams *et al.* [27]	
No. post-message-passing layers	1	Adams *et al.* [27]	
Drop-out rate	0.0	You *et al.* [92]	
Activation function	PReLU	You *et al.* [92]	
Optimizer	Adam	Adams *et al.* [27], You *et al.* [92]	Using default settings.
No. hidden dimensions	$\{8, 16, 24, 32, 48, 64, 128, 256\}$		Adams *et al.* [27] used 24.
Learning rate	$\{0.01, 0.001\}$	Adams *et al.* [27], You *et al.* [92]	You *et al.* [92] recommend 0.01, which is also used in [27].

5.3 Evaluation Methods

We implement several methods to evaluate the encodings and their respective models and hyperparameters. First, a set of informative and representative baselines is given. Next, performance metrics are elaborated on. Additionally, generalizability is expounded. Lastly, we discuss how scalability is evaluated.

5.3.1 Baseline

Three baselines are used for determining the eminence of models or encodings. Baselines aid in indicating how to interpret performance metrics.

Median serves as our first uninformed baseline. As our prediction task is regression-based, the central tendency measure median is a suitable baseline to give context to the results of complex learners.

LightGBM is used as a second baseline. This is a lightweight, yet performant gradient booster model which has been shown to be promising for PPM tasks [62]. Gradient boosting models require a tabular data structure with fixed-size dimensions. This entails transforming the graph encodings (Section 5.1) to tabular feature matrices, losing structural information. For HOEG, fixed-size feature matrices cannot be attained, as these encode heterogeneous data that may have different dimensions per node type. Therefore, the LightGBM baseline takes only a tabular version of the event node types as input data. We refer to this as the event feature table (EFT) encoding.

EFG$_{ss}$ is the third baseline model. We replicate the GCN architecture used by Adams *et al.* [27]. The configuration of this GCN is reconstructed to be as similar as possible to their model and input data structure. This entails that it runs on an adapted version of the EFG configuration. That is, subgraph sampling[15] of size four is applied to the EFG encoded datasets before passing data to the GCN. The GCN performs graph-level predictions, pooling information from the four nodes into one prediction. This is different from our EFG-based and HOEG-based models, which make predictions per event (node). Furthermore, the GCN is configured with the hyperparameters found in Table 8 that refer to Adams *et al.* [27]. Hyperparameters not referred to are set to: $batch\ size = 64$, $early\ stopping$ is disabled, $drop\text{-}out\ rate = 0.0$, $activation\ function = PReLU$, $hidden\ dimensions = 24$, and $learning\ rate = 0.01$. Finally, as this baseline runs on EFG (as specified in Section 5.1.1), but with subgraph sampling enabled, it is referred to as EFG$_{ss}$ in the remainder of this work.

5.3.2 Performance

Regarding the performance evaluation of models, we aim to depict a complete picture. Therefore, three metrics are reported. Considering that the outputs of our models are continuous, we include only regression performance measures. The first, the mean absolute error (MAE) takes the average of the sum of absolute errors. For this reason, it is a highly interpretable measure.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y_i}| \tag{14}$$

We additionally include the mean squared error (MSE), which yields a positive error number by taking the average square of the errors. As such, larger errors are weighted heavier, resulting in a greater error score for models that exhibit overfitting.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2 \tag{15}$$

Finally, the mean absolute percentage error (MAPE), is an intuitive relative error score in terms of interpretation. It represents the average difference between the predicted and actual values expressed as a percentage of the actual values. For example, a MAPE value of 7% implies that, on average, the model's prediction deviates from the actual value by 7%.

$$\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}|\frac{y_i - \hat{y_i}}{y_i}| \tag{16}$$

In our context, this measure has two drawbacks. To start, if the actual value is zero ($y_i = 0$) the score gives an undefined value. Second, since the percentage is used, a prediction

[15]Consecutive nodes (ordered by time) of size four are sampled.

lower than the true value is favored compared to a higher predicted value (e.g., for $y_i = 1, \hat{y}_i = 2$ we yield $|\frac{1-2}{1}| = 1$, while $y_i = 2, \hat{y}_i = 1$ is scored $|\frac{2-1}{2}| = 0.5$).

To conclude, for all metrics listed above, a lower value means a better-performing model.

5.3.3 Generalizability

When developing any machine learning model, we always navigate the bias-variance trade-off in some way. The goal of building a predictive model is to make it generalizable beyond the data it was trained on to provide value in an unknown data context. An estimator that has high bias has not learned the relevant relations between features with respect to the target variable (underfitting). Opposing this is a model with high variance. Such a model has fitting to the random noise in its training data, disregarding the general patterns that exist in the full population (including data outside the training set) [93]. In supervised learning, we strive to develop a predictor that has a good balance between high bias and high variance, learning as much as possible from training data, while generalizing well outside the training set.

We want to be able to indicate where a model lies on this spectrum. Therefore, we split each data set into three splits: the training, validation, and testing split. A learner then is fitted to the training set, while the validation set is used to compare against when looking for optimal parameter and hyperparameter [16] combinations. From here, the best-performing model is chosen. Naturally, this results in a bias toward the validation set. For that reason, we evaluate the selected model using a final test set.

In the case of deep learning models, the training process of the final estimators is visualized and interpreted to rectify the choice for those final models. The results of all models on each of the splits are reported using the previously defined measures.

5.3.4 Scalability

The final dimension along which the presented approaches in this work are evaluated is scalability. We do this by reporting on the training and prediction times. Each experiment runs on the same hardware: Intel$^{(R)}$ Core$^{(TM)}$ i5-7500 @ 3.40GHz (4x) with 48GB memory and NVIDIA GeForce GTX 960 with 4GB memory.

5.4 Object-Centric Datasets

The experiments are run on three datasets. The first, the loan application OCEL, is a traditional event log transformed into an object-centric one. The second is an object-centric dataset from an order management process. This OCEL was originally generated for demonstration purposes [5, 23], but is now being used for predictive process mining experiments as well [26, 29]. The last is taken from a real-life operational system at a large

[16] A parameter is internal to a model and can be optimized from data, while a hyperparameter is external to a model, being configurable by the modeler [94].

financial organization. For each of the OCELs, we outline the data preprocessing steps and describe the dataset.

5.4.1 Loan Application Log

The Business Process Intelligence Challenge is an initiative that brings together academia and industry by yearly publishing a process mining-related contest on a real-life dataset. In 2017, the corresponding event log [85] came from a loan application process, where events can relate to one **application** and multiple **offer** objects. We adapt this to the OCEL standard, drawing upon the work done by Adams *et al.* [27] [17]. The resulting data model is drawn in Figure 12.

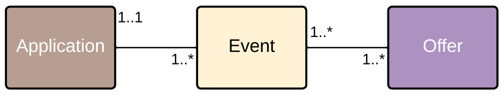

Figure 12. Loan application OCEL data model. Note that this data model is an instantiation of the contents of the *Log* class in the OCEL metamodel given in Figure 2.

The log contains an attribute (`EventOrigin`) that indicates from which object type events originate. This is used to create object references per event. Regarding attributes, the original log contains only event attributes. Some of these show signs that they are transformed to be an event attribute, suggesting that they might fit as an object attribute when transforming the log into an object-centric event log. The HOEG feature encoding might leverage object attributes for remaining time prediction. Therefore, we include several attributes present in the dataset originally presented, as object attributes. Determining which attributes should describe objects and which events, is done using domain knowledge, basic semantics, and data exploration. In doing so, the event log is made to conform to the OCEL v1.0 standard [23]. Table 9 shows a summary of relevant characteristics of this event log, which is referred to as BPI17 in the remainder of this work. The table displays four dimensions: ***Cases***, ***Events***, ***Objects*** and ***Event-Object Interaction***. Worth noting is that on the ***Events*** dimension it becomes apparent that there is relatively little standard deviation in events per case. Additionally the mean and median are close to each other for events per case as well as throughput time. This might suggest that the traces have a similar distribution, with little outliers that draw the average (trace length or throughput time) away from the center (median). Regarding objects in the OCEL, there are 10 additional attributes available for machine learning when HOEG is employed. The first three of object attributes listed belong to **application** and the others to the **offer** object type.

Going into the experiments, this dataset is split into training, validation, and testing sets based on process executions. For comparability reasons, we replicate the split used

[17]The exact dataset can be found within the source code of "A Framework for Extracting and Encoding Features from Object-Centric Event Data": `https://github.com/niklasadams/OCELFeatureExtractionExperiments`.

Table 9. Summary of the BPI17 OCEL. *Note*: a case is defined by the connected components execution extraction technique ([8], Def. 5).

Characteristic	Value
Cases	
Statistics	
Total	31,509
Minimum throughput time (s)	201.1
Maximum throughput time (s)	14,604,259.8
Median throughput time (s)	1,646,735.9
Mean throughput time (s)	1,887,853.9
Standard deviation throughput time (s)	1,119,596.8
Events	
Statistics	
Total	393,931
Minimum per case	6
Maximum per case	41
Median per case	12
Mean per case	12.5
Standard deviation per case	3.5
Attributes	
Action	Categorical
EventOrigin	Categorical
OrgResource	Categorical
Objects	
Statistics	
Total applications	31,509
Total offers	42,995
Attributes	
RequestedAmount	Numerical
ApplicationType	Categorical
LoanGoal	Categorical
OfferedAmount	Numerical
CreditScore	Numerical
Selected	Binary
MonthlyCost	Numerical
NumberOfTerms	Numerical
Accepted	Binary
FirstWithdrawalAmount	Numerical
Event-Object Interaction	
Statistics	
Event<>Application	328,894
Event<>Offer	201,006

by Adams *et al.* [27], which sets aside 30% of the traces for model testing, 0.14% for validation, and 0.56% for model training. To prevent information leakage, we apply Z-score normalization to the full dataset based on the training set.

5.4.2 Order Management Log

The order management dataset (OTC) is a small OCEL that has been generated to serve as an example in works by Van der Aalst & Berti [5] and Ghahfarokhi *et al.* [23]. Originally it contains five object types: **order**, **item**, **package**, **product**, and **customer**. As the latter two contain few unique instances, the log produces one large trace (via connected components extraction) containing all events. As the computational complexity of many object-centric features (from [27]) increases quadratically with trace length, these cannot be calculated within a feasible timeframe. Therefore, we exclude **product** and **customer** as object types and use **item** as the leading type for execution extraction, so that we have multiple and shorter traces. The data model, then, is represented in Figure 13.

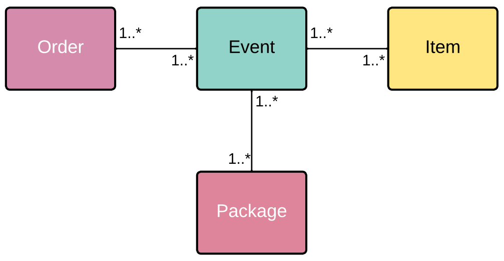

Figure 13. Order management OCEL data model. Note that this data model is an instantiation of the contents of the *Log* class in the OCEL metamodel given in Figure 2.

Concerning object attributes, those that describe a **customer**, are adapted to be event attributes. This can be done without data duplication or loss, as events in the log have a one-to-one relationship with **customer**. Furthermore, the remaining object types: **order**, **item**, and **package**, do not contain object attributes in the initially generated OCEL. The HOEG encoding does require at least one object attribute per object type. Therefore, we encode the object identifiers as numerical object attributes. We assume these do not contain relevant information concerning our prediction target, therefore the object attributes included in the prepared OCEL could be seen as noise.

After preprocessing, the OTC OCEL can be characterized via Table 10. Evident from these summary statistics is the limited size of the dataset, with $8,159$ traces and $22,367$ events. Additionally, in contrast to the BPI17 OCEL (Table 9, this table reports a modest standard deviation throughput time in proportion to the mean or median throughput times. Otherwise worth noting is the relatively large number of events per case, which is caused by the high level of event-object interaction in the log. That is, the log contains a high number of object references per event, as indicated by the many-to-many relationships in Figure 13 and the high values under ***Event-Object Interaction*** in comparison to Tables 9 and 11. These values indicate how many references there are from events to specific object types.

After encoding this log into EFG or HOEG, we split it into training, validation, and testing sets of traces, using a $0.7/0.15/0.15$ split. Finally, based on the set of traces used for training, we apply Z-score normalization to all splits.

5.4.3 Financial Institution Log

The third dataset included in the experiments is an object-centric event log from a large financial institution (FI[18]) that is kept anonymous. Therefore, details are withheld throughout this paper. As the OCEL concerns a critical process of the organization, the dataset cannot be made publicly available. The data originates from a workflow management

[18]This abbreviation is used in the remainder of this paper to refer to the OCEL originating from the financial institution.

Table 10. Summary of the OTC OCEL. *Note*: a case is defined by leading type extraction ([8], Def. 6), with **item** as the leading type.

Characteristic	**Value**
Cases	
Statistics	
Total	8,159
Minimum throughput time (s)	447,864.0
Maximum throughput time (s)	12,109,961.0
Median throughput time (s)	3,335,901.0
Mean throughput time (s)	3,594,196.7
Standard deviation throughput time (s)	1,638,634.3
Events	
Statistics	
Total	22,367
Minimum per case	8
Maximum per case	155
Median per case	56
Mean per case	57.9
Standard deviation per case	22.8
Attributes	
Weight	Numerical
Price	Numerical
Age	Numerical
Bankaccount	Numerical
Objects	
Statistics	
Total orders	2,000
Total items	8,159
Total packages	1,325
Attributes	
ObjectID	Numerical
Event-Object Interaction	
Statistics	
Event<>Order	895,990
Event<>Item	1,761,874
Event<>Package	145,490

system (WFMS) operational in the partnering financial institution. In the WFMS process executions have been loosely standardized. That is, activities are named and connected with a digital dossier. The system, therefore, produces well-defined case identifiers that enable traditional process mining. Each dossier is viewed as a case identifier. There are three dossier types: **KRS**, **KRV**, and **CV**, with a respective progression in complexity. If further work is required at the end of a **KRS** dossier, a **KRV** dossier is opened in the WFMS. The same holds for a **KRV** that evolves into a **CV**. While work has started in a new, more complex dossier, activities can still be executed for the preceding dossier(s). This results in divergence, to which object-centric process mining provides a solution.

The data resulting from the described process is modeled in Figure 14. From the data model, we observe that an OCEL can be constructed. In Figure 15, the specific data pipeline is given.

This pipeline takes data from the WFMS application logs and enriches this with attributes from a dataset containing multi-source combined enrichment data (e.g. location, number of responsible employees). All categorical attributes used for enriching are encoded using count encoding with normalization, such that group names are replaced by percentage counts of the group name occurrence. This was done before extraction from

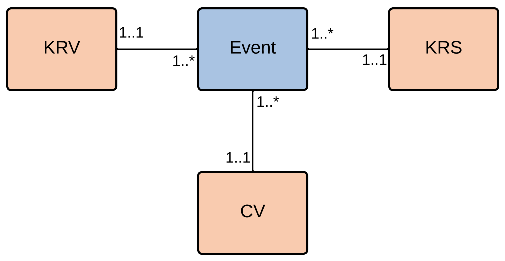

Figure 14. Financial institution OCEL data model. Note that this data model is an instantiation of the contents of the *Log* class in the OCEL metamodel given in Figure 2.

the *Output Layer*, as an anonymization strategy.

Most crucial to attaining an OCEL, is the *Derive Linked Dossiers* step in the processing layer. Here, based on dossier lineage information from the *Combined Enrichment Database*, an activity *Link Objects* is inserted between each dossier promotion (e.g. from **KRV** to **CV**). *Link Objects* refers to both object types, creating an object-centric link in the event data.

After the *Processing Layer* we yield an events table and an objects table (containing the dossiers). Events are taken from January 1st, 2023 until May 1st, 2023, and objects are viewed as static entities.

Table 11 gives an initial characterization of the data, distinguishing four axes: cases, events, objects, and event-object interaction. Regarding cases and events, we observe a positive skew in the distribution of the case length. Besides that, we note a relatively high standard deviation for case length and throughput time when compared with the other datasets (Tables 9 and 10). This might be indicative of a highly complex process. Furthermore, this is the largest event log included in this work, with $695,694$ events, which have 3 event attributes each. As for objects, the log contains $96,444$ objects of three types: **KRS**, **KRV**, and **CV**. These object types typically have a sequential order. That is, usually a **KRS** evolves into a **KRV**, which can evolve into a **CV**. It is also possible for a **KRS** to be directly linked to a **CV**. Each follow-up object type implies a more complex process execution in terms of work performed (and seniority required) by the process practitioners. This predictable progression can be seen in the relatively little event-object interaction. Table 11 reports events referring to the different object types in comparison to the other logs (cf. Tables 9 and 10). This can be explained by the fact that the dataset has been extracted from a workflow-oriented information system rather than a service-oriented system. The latter lends itself to higher levels of object interaction as cases are not predefined at the configuration or deployment of the system [95].

When preparing event-based graphs (for EFG and HOEG) from the OCEL, only the activities occurring in more than 4% of the events are included (in feature **C2**). Besides that, we follow the configuration procedure presented in Section 5.1.1. The execution graphs are split into a training, validation, and testing set, with a $0.7/0.15/0.15$ split. Lastly, based on

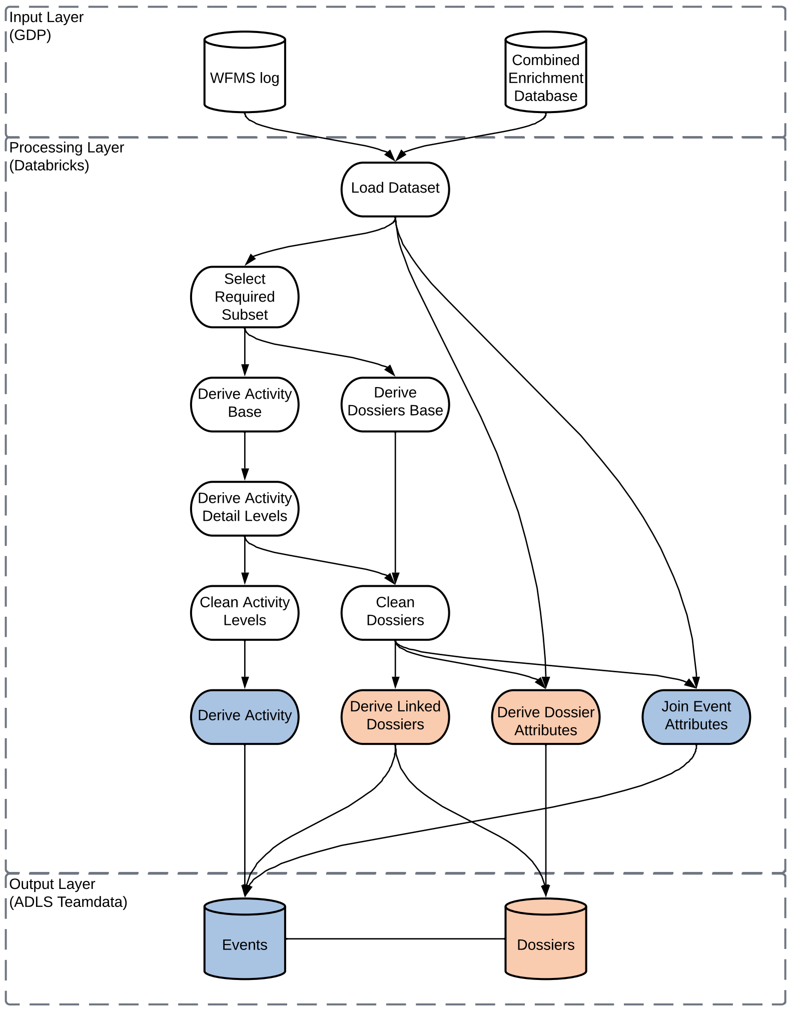

Figure 15. The financial institution OCEL pipeline visually indicating the steps required to arrive at the events and objects tables. *Note*: arrows denote dependencies and data flows.

the training split, each graph's features are standardized via Z-score normalization.

5.5 Machine Learning Pipeline

The previous sections can be broadly summarized in the pipeline depicted in Figure 16. The figure shows the components of an experiment instance. We distinguish five such components: *data loading*, *feature encoding*, *model training*, *model evaluation*, and *tracking*.

Traversing the pipeline, we observe that an experiment can become a complex system to maintain. Hence, we have opted to divide it into segregated modules that each have their responsibility.

The *data loading* module is responsible for parsing and validating a dataset following the OCEL v1.0 standard. Note how we may enter any OCEL into the system.

The *feature encoding* component then takes this OCEL and transforms it into the four different encoding types. Here, we select the features given in Section 5.1 to fill the encod-

Table 11. Summary of the Financial Institution OCEL. *Note*: a case is defined by the connected components execution extraction technique ([8], Def. 5).

Characteristic	Value
Cases	
Statistics	
Total	31,277
Minimum throughput time (s)	1.1
Maximum throughput time (s)	10,111,075.0
Median throughput time (s)	2,432,327.0
Mean throughput time (s)	2,957,048.3
Standard deviation throughput time (s)	2,354,827.1
Events	
Statistics	
Total	695,694
Minimum per case	4
Maximum per case	158
Median per case	20
Mean per case	22.2
Standard deviation per case	13.0
Attributes	
Categorical	3
Numerical	0
Objects	
Statistics	
Total KRS	31,513
Total KRV	31,357
Total CV	31,278
Attributes	
Categorical	13
Numerical	1
Event-Object Interaction	
Statistics	
Event<>KRS	235,756
Event<>KRV	466,965
Event<>CV	56,137

ing structures. Furthermore, this component requires split ratios for determining the train, validation, and test sets.

Model training, then, serves to optimize each of the four models (including specified hyperparameters). The graph-based models are implemented in `PyTorch Geometric`, an established Python library based on `PyTorch` [96] to enable GNN development [97]. In the figure, the Median baseline is excluded, as it does not learn or train to fit complex statistical patterns in the data.

The models under research are then passed to the *model evaluation* component, where their predictions are scored on each split based on the metrics presented in Section 5.3.2. From these metrics, we generate a report that is used to analyze each model's performance.

Lastly, the *tracking* module, retrieves run time information from the *feature encoding*, *model training*, and *model evaluation* modules and monitors the training process of each model. The first facilitates scalability reports on preprocessing methods and model training and prediction. The second enables us to navigate the bias-variance trade-off by visually analyzing the training process of our deep learning models. Via such visualizations, we are able to assess model stability and learning capacity during training.

Note how at most phases in the experiment flow, we can specify certain configurations (e.g., datasets, features, hyperparameters, evaluation metrics). We instantiate each experi-

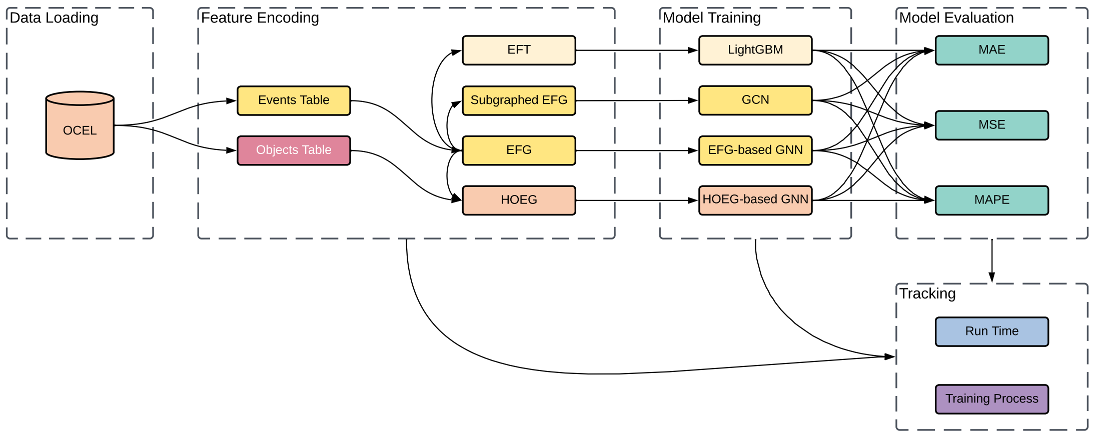

Figure 16. The machine learning pipeline of each experiment.

ment employing the same settings (except dataset and split ratio), as discussed in previous sections of this chapter.

6 Results

The subsequent section presents the results obtained through three experiments. In the first, we experiment with different hyperparameter settings, producing one best performing model per encoding. The second experiment compares model performance across the encodings. Lastly, we elaborate on the baseline experiment, where we evaluate our models against different baselines. All three experiments involve the datasets introduced in Section 5.4, highlighting the generalizability of the results.

6.1 Hyperparameter Tuning Experiment

The goal of hyperparameter tuning is to find a set of hyperparameters that result in performant models across different datasets. Furthermore, it can give insight into model stability by inspecting the performance of the different settings for *learning rate* and *hidden dimensions*. Recall that *learning rate* influences convergence speed and performance and *hidden dimensions* affects a model's learning capacity.

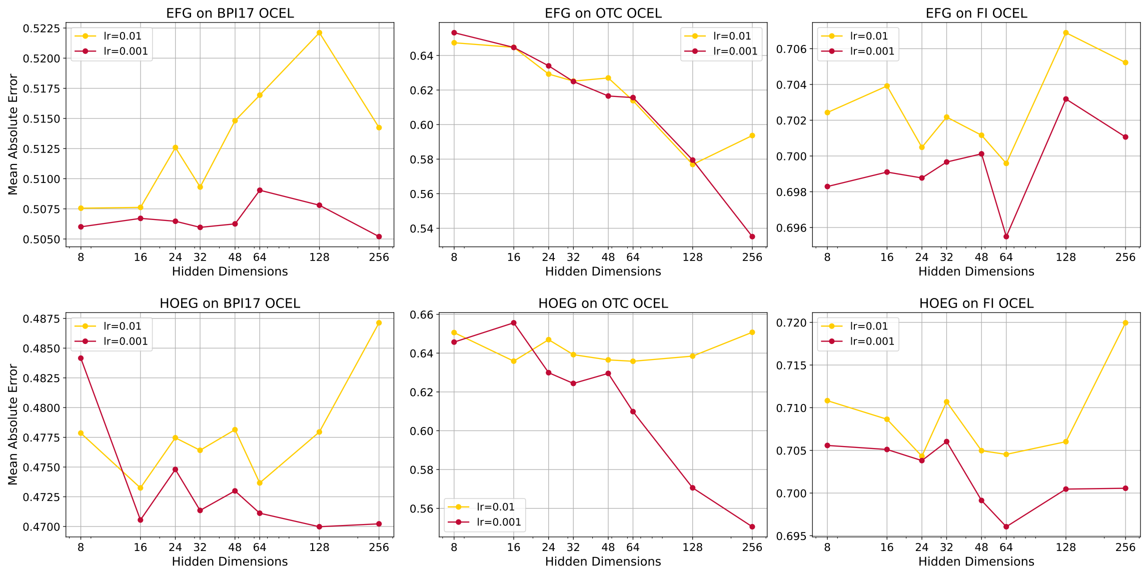

Figure 17. Test MAE scores for the EFG and HOEG encoding per hyperparameter setting. *Note:* scales (y-axis) are not aligned, as we intend to compare hyperparameter settings within each encoding and dataset.

Figure 17 shows the results of tuning *learning rate* and *hidden dimensions* simultaneously. We first tune the EFG-based model. A first observation shows that the models are stable for both BPI17 and FI, as the MAE score range is relatively small. We generally observe that a lower *learning rate* of 0.001 scores better. Looking at *hidden dimensions*, 256 seems to work best for BPI17 and OTC, while *hidden dimensions* $= 64$ yields the lowest MAE for EFG on the FI OCEL. Besides that, we observe a pattern for the OTC dataset suggesting that increasing model complexity improves the learning capacity of the EFG-based model. For the BPI17 and FI OCELs, Figure 17 does not indicate a distinct pattern concerning the *hidden dimensions* hyperparameter, as the various configurations exhibit relatively similar performance.

For the HOEG encoding, we observe approximately the same model stability per dataset as with EFG in terms of performance variance between hyperparameter combinations. Again *learning rate*=0.001 yields the best model, in combination with 64, 128, and 256 *hidden dimensions* for the FI, BPI17, and OTC OCELs respectively. Akin to the EFG performance on OTC, Figure 17 suggests that increasing the number of *hidden dimensions* aids model performance.

Table 12. HOEG compared against EFG in terms of model complexity on all three datasets.

Hidden	**BPI17**		**OTC**		**FI**		**Average**	
Dimensions	**Model Parameters**		**Model Parameters**		**Model Parameters**		**Model Parameters**	
	EFG	**HOEG**	**EFG**	**HOEG**	**EFG**	**HOEG**	**EFG**	**HOEG**
8	587	1,857	475	1,924	395	3,092	**486**	**2,291**
16	1,427	4,985	1,203	5,628	1,043	7,964	**1,224**	**6,192**
24	2,523	9,393	2,187	11,124	1,947	14,628	**2,219**	**11,715**
32	3,875	15,081	3,427	18,412	3,107	23,084	**3,470**	**18,859**
48	7,347	30,297	6,675	38,364	6,195	45,372	**6,739**	**38,011**
64	11,843	50,633	10,947	65,484	10,307	74,828	**11,032**	**63,648**
128	40,067	183,177	38,275	245,644	36,995	264,332	**38,446**	**231,051**
256	145,667	694,025	142,083	950,028	139,523	987,404	**142,424**	**877,152**
Average Scale	1	4.0	1	5.5	1	7.4	**1**	**5.6**

Table 12 highlights the inherent higher complexity of the HOEG encoding. The same number of *hidden dimensions* implies a different number of model parameters for EFG and HOEG. On average, our HOEG configurations have 5.6 times the number of model parameters of an EFG (on equal datasets). This is due to the HOEG model being a heterogeneous GNN, which duplicates a homogeneous GNN architecture across each node type and inserts message-passing layers per edge type (as explained in Section 2.2.4). By this, HOEG is able to leverage object (interaction) information.

Concluding the first experiment, we observe comparable model stability for both encodings. Drawing definitive conclusions about a universally optimal dataset-agnostic hyperparameter setting for an encoding seems to be premature. Additionally, we cannot observe an overall superiority of one encoding type over the other. The next experiment delves more into comparing the encoding types.

6.2 Encoding Type Experiment

In this experiment, we contrast the EFG and HOEG encoding types. Distributions are given of the model performance of all hyperparameter combinations per split (train, validation, and test). In addition, the learning curves of the best models for both EFG and HOEG are plotted per dataset, providing insight into the training process and model performance during training. Using both visualization types, we directly compare the two encoding types in terms of model stability and learning capacity.

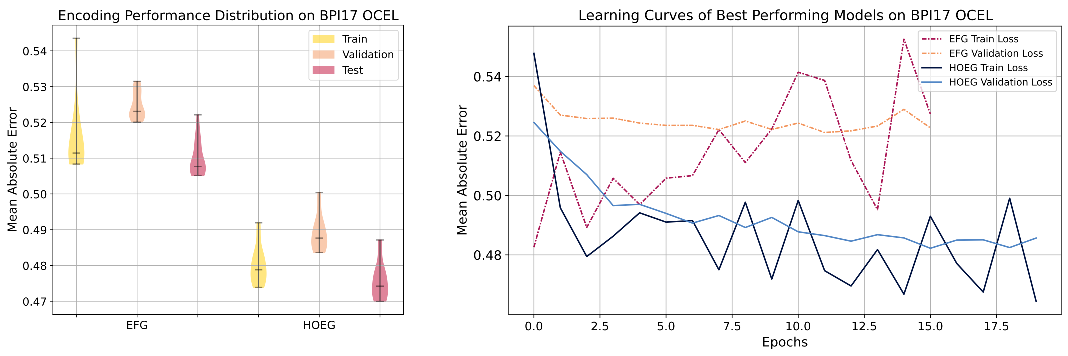

Figure 18. Violin plot of the MAE score distribution over different hyperparameter settings per split for EFG and HOEG encodings on BPI17 OCEL (left). Training and validation loss learning curves of tuned EFG and HOEG models on BPI17 OCEL (right).

Figure 18 provides an initial impression of the potential prevalence of HOEG over EFG in terms of learning capacity on the loan application OCEL. Looking at HOEG in the violin plot, there is notable interquartile overlap among scores for all three splits, which suggests stable model performance. In contrast, the EFG exhibits some indications of instability, particularly on the training split. This is primarily attributed to a few outliers in the MAE scores on the training split. However, when considering most MAE scores across the three splits, EFG's performance appears relatively consistent.

Assessing the learning curves in Figure 18, we note that EFG (depicted in orange and burgundy) shows indications of model instability. Its training loss curve starts at a relatively low value, after which it increases in an erratic manner. In contrast, the validation loss curve demonstrates only marginal improvement, suggesting that the model struggles to learn meaningful patterns from the data. HOEG, on the other hand, presents a more favorable learning curve. The training curve drops over the course of the epochs, indicative of effective learning. While fluctuations are evident, the overall trajectory points to the model's capacity to adapt to the training data. Importantly, the validation loss curve follows a steady downward trend, signifying that the model generalizes well to unseen data.

The violin plot in Figure 19 highlights the performance distribution on the order management demo OCEL. Notably, both encodings display significantly lower MAE scores on the training split compared to the validation and testing splits. This discrepancy suggests the models are overfitting, as both EFG and HOEG excel in learning the training data's patterns but exhibit challenges in generalizing to the validation and holdout sets. When assessing the encoding performances across the splits, it becomes apparent that EFG shows a slight advantage over HOEG, although we cannot assert definitive conclusions based on these observations.

The learning curves reinforce the suggestion that both encodings highly overfit the order management event data, as both encodings show diverging training and validation losses. Regarding model stability, all loss curves are in steady decline, which is indicative

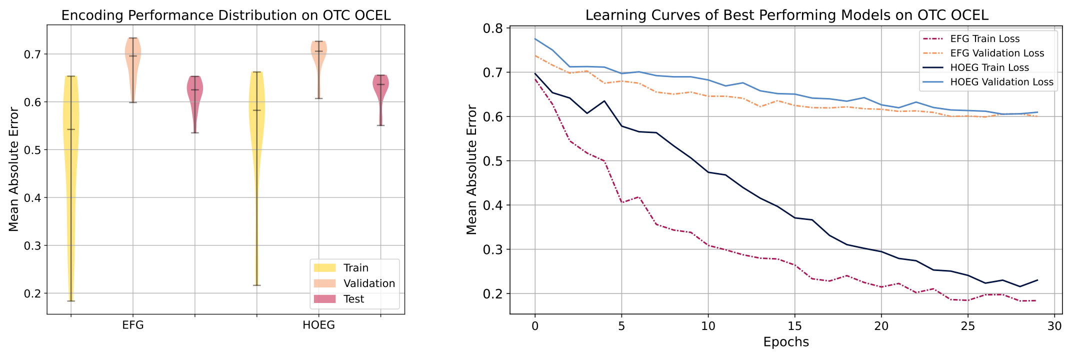

Figure 19. Violin plot of the MAE score distribution over different hyperparameter settings per split for EFG and HOEG encodings on OTC OCEL (left). Training and validation loss learning curves of tuned EFG and HOEG models on OTC OCEL (right).

of stable learning.

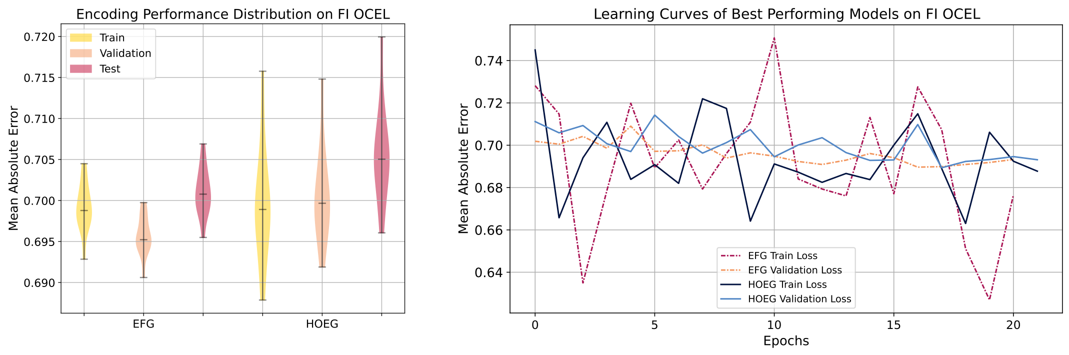

Figure 20. Violin plot of the MAE score distribution over different hyperparameter settings per split for EFG and HOEG encodings on FI OCEL (left). Training and validation loss learning curves of tuned EFG and HOEG models on FI OCEL (right).

Figure 20 presents us with the MAE score distribution of the various configurations of EFG and HOEG, along with a plot depicting the learning curves of the best model configurations on the FI dataset. A preliminary observation of the left plot suggests that EFG exhibits more favorable score distributions compared to HOEG. However, further examination, including interquartile overlap and insights from the learning curves plot, indicates that HOEG may possess greater stability. The learning curves on the right demonstrate that HOEG's training loss is less volatile, whereas EFG's validation loss maintains a more stable pattern throughout training. While neither EFG nor HOEG appears to be overfitting, both encoding types struggle to effectively capture the OCEL patterns required for accurate remaining time predictions.

In general, the results of the second experiment are not conclusive about which encoding facilitates the most expressiveness of OCELs to predict process remaining time. Looking at the learning curves, it might be suggested that HOEG trains in a more stable manner than

EFG. Given the nuances observed in this second experiment regarding the performance of HOEG and EFG, the following experiment delves into a comparative analysis against established baselines to provide more context for evaluating the encodings explored.

6.3 Baseline Experiment

This baseline experiment provides context and helps us understand the relative impact of the results of the previous experiments. Besides EFG, HOEG is now contrasted with Median, LightGBM, and EFG$_{ss}$ (by Adams *et al.* [27]) baseline models per dataset (see Section 5.3.1). Recall that LightGBM takes tabular input data, meaning that structural information is not included. Furthermore, EFG$_{ss}$ refers to a GCN architecture running on EFG with subgraph sampling (subgraph size: $k = 4$). Tables 13, 14, and 15 enable comparative analysis of model performance, measured by MAE, MSE, and MAPE, as well as model scalability in terms of fitting time and prediction time. The prediction time column gives the time in seconds that it takes a model to predict on a complete dataset (all three splits). Finally, per column, the most favorable value is emphasized for each metric.

Table 13. HOEG compared against EFG and different baseline models on performance and scalability on BPI17 OCEL. *Note* that the best performing hyperparameters have been selected: $lr = 0.001$, $hd = 128$ for both HOEG and $lr = 0.001$, $hd = 256$ for EFG.

Model	**Train Score**			**Validation Score**			**Test Score**			**Fitting**	**Prediction**
	MAE	**MSE**	**MAPE**	**MAE**	**MSE**	**MAPE**	**MAE**	**MSE**	**MAPE**	**Time (s)**	**Time (s)**
Median	0.7854	1.0802	4.9764				0.7746	1.0472	4.7171	**0.0042**	**0.0005**
LightGBM	0.5282	0.5730	8.3854				0.5282	0.5664	8.2184	15.9230	0.8351
EFG$_{ss}$[1]	**0.4414**	**0.5481**	16.9989	**0.4519**	**0.5627**	15.6191	**0.4377**	**0.5322**	14.1622	547.1785	86.1564
EFG	0.5084	0.6010	**2.9345**	0.5209	0.6211	**3.7560**	0.5052	0.5855	**2.7618**	122.0836	20.5225
HOEG	0.4739	0.5745	5.3590	0.4836	0.5878	5.4609	0.4700	0.5610	5.1711	536.0755	68.5030

[1] GCN model running on EFG with subgraph sampling of size four and hyperparameters: $lr = 0.01$, $hd = 24$.

Upon looking at the results for the loan application log (BPI17), we notice that the EFG$_{ss}$ scores best on MAE and MSE (on all splits), meaning its predictions are closest to the actual values on average and do not deviate significantly from the true remaining times compared to the other models. The highest test MAE is 0.7746 (Median), while the lowest is 0.4377. In contrast with the Median baseline model, HOEG achieves an MAE score improvement of $\frac{0.4700-0.7746}{0.7746} \times 100\% = -0.39\%$. EFG achieved the best MAPE score (on all splits), which indicates that its predictions tend to be closer to the true values in relative terms. Crucially, from strongest to weakest we observe the following ordering in model performance: EFG$_{ss}$, HOEG, EFG, LightGBM, Median.

Interestingly, on the scalability perspective, Table 13 reports the reverse. The Median takes the least time training and predicting, while the EFG$_{ss}$ is the slowest. This is expected, as the Median model does not learn patterns like the others, there are no parameters to be fit. Therefore, when comparing the actual learning algorithms, it is important to note that the LightGBM significantly outperforms the graph-based models. Also interesting is the fact that the HOEG model is more scalable than the EFG$_{ss}$, even while having a much higher parameter count and a lower learning rate. That is, the EFG$_{ss}$ has

$1,297$ model parameters and a learning rate of 0.01, compared to $183,177$ parameters and a learning rate of 0.001 for HOEG.

Table 14. HOEG compared against EFG and different baseline models on performance and scalability on OTC OCEL. *Note* that the best performing hyperparameters have been selected: $lr = 0.001$, $hd = 256$ for both HOEG and EFG.

Model	**Train Score**			**Validation Score**			**Test Score**			**Fitting**	**Prediction**
	MAE	**MSE**	**MAPE**	**MAE**	**MSE**	**MAPE**	**MAE**	**MSE**	**MAPE**	**Time (s)**	**Time (s)**
Median	0.7379	0.9888	4.4810				0.7175	0.8762	**3.3845**	**0.0064**	**0.0021**
LightGBM	0.5422	0.5021	5.6673				0.6060	**0.5980**	5.2201	1.9556	0.7211
EFG$_{\text{ss}}$[1]	0.6359	0.7769	11.2216	0.7338	1.0440	13.3283	0.6585	0.7560	11.4388	750.0486	150.5996
EFG	**0.1835**	**0.1601**	11.0881	**0.5985**	0.8779	13.6788	**0.5352**	0.6951	38.7516	83.8779	6.6364
HOEG	0.2163	0.1804	**2.2094**	0.6069	**0.8723**	**5.8325**	0.5505	0.6952	5.5826	305.0114	25.1399

[1] GCN model running on EFG with subgraph sampling of size four and hyperparameters: $lr = 0.01$, $hd = 24$.

For the baseline results on the demo object-centric event log (OTC), performance scores are rather different. The test MAE ranges from 0.5352, achieved by EFG, to 0.7175, attained by Median. The EFG seems to be prominent when the MAE and MSE metrics are viewed. HOEG closely follows and additionally performs better with the MAPE metric on the training and validation set. Unexpectedly, on the holdout set, the Median model achieves the best MAPE. Conspicuously, in line with the figures in the previous experiments, for EFG and HOEG there is a steep increase in both MAE and MSE scores going from the train to the validation or test split.

In terms of scalability, the same pattern as with the BPI17 OCEL appears. That is, the Median takes the least time, followed by the LightGBM, EFG, HOEG, and finally the EFG$_{\text{ss}}$.

Table 15. HOEG compared against EFG and different baseline models on performance and scalability on FI OCEL. *Note* that the best performing hyperparameters have been selected: $lr = 0.001$, $hd = 64$ for HOEG and EFG.

Model	**Train Score**			**Validation Score**			**Test Score**			**Fitting**	**Prediction**
	MAE	**MSE**	**MAPE**	**MAE**	**MSE**	**MAPE**	**MAE**	**MSE**	**MAPE**	**Time (s)**	**Time (s)**
Median	0.7673	1.2763	4.9429				0.7702	1.2881	2.5284	**0.0071**	**0.0005**
LightGBM	0.7167	**0.8035**	**4.6391**				0.7286	**0.8334**	**1.8010**	15.8959	1.1631
EFG$_{\text{ss}}$[1]	0.7250	1.0537	11.4524	0.7222	1.0338	**9.0335**	0.7310	1.0579	8.3716	645.9569	749.5316
EFG	0.6928	0.9487	20.1893	**0.6906**	**0.9328**	13.9496	**0.6955**	0.9518	7.8566	211.5477	21.3843
HOEG	**0.6879**	0.9911	22.4544	0.6919	0.9898	14.6175	0.6961	1.0037	15.1977	884.2844	82.1531

[1] GCN model running on EFG with subgraph sampling of size four and hyperparameters: $lr = 0.01$, $hd = 24$.

The object-centric event data from the complex process at the financial institution yields different scores again. Here, EFG marginally outperforms HOEG. Both set themselves apart from the baseline models, but not by much, as our main regression metric, MAE, ranges from 0.6955 (EFG) to 0.7702 (Median).

When considering model scalability, the fitting times differ from the other datasets. Here, HOEG takes the longest to train, while EFG$_{\text{ss}}$ comes in second. Prediction times display the same order as the other datasets. Worth noting, is that compared to HOEG, the EFG$_{\text{ss}}$ took significantly longer to predict on the train, validation, and test set, taking nearly 12.5 minutes (749.5316 seconds).

On the whole, the HOEG outperforms the baseline models in all datasets based on MAE. Considering all three metrics and OCELs, the results could be used as evidence that hints at the GNN based on our configuration of the EFG encoding standing out across the datasets. However, the characteristics of the three datasets appeal to a more nuanced interpretation and discussion.

7 Discussion

This chapter discusses interpretations, limitations, and implications of the results previously presented. There, we learned that specific characteristics of the datasets might influence the capacity of the models to learn relevant patterns and relationships from the encoding structures. Therefore, the following is structured by dataset, along with a section synthesizing the results and answering the research questions posed in the Introduction.

7.1 Loan Application Log

The BPI17 OCEL is relatively structured, having only two object types and originating from a workflow management system. Each event is always related to one **application**, and only the latter events in a trace are related to one or more **offer**s. Regarding the experimental results, this may be why the machine learning models, and especially the graph-based ones, learn so well on this dataset. The results of the hyperparameter tuning experiment may be used to support this, since for all hyperparameter settings, the HOEG-based model performs relatively well.

Based on MAE and MSE, EFG$_{\text{ss}}$ outperforms EFG by a fair margin and HOEG by a slight margin. This might seem odd, as the EFG$_{\text{ss}}$ is derived from the EFG encoding, which is a large component of the HOEG structure. This may be explained by the differing preprocessing pipelines of the graph-based features driving the EFG$_{\text{ss}}$-based GCN and the features undergirding the higher-order GNNs running on our EFG and HOEG configurations.

The EFG$_{\text{ss}}$-based GCN performs graph-level predictions on subgraphs of size 4, sampled from EFG process executions. The HOEG-based and EFG-based models give a prediction per event node. This difference has two implications in favor of the EFG$_{\text{ss}}$.

First, the EFG$_{\text{ss}}$ receives more examples, as the total events in the OCEL are duplicated $\sum\limits_{t \in L}(t_{size} - (k-1)) \times 1_{\{t_{size} \geq k\}}$[19] times via subgraph sampling. For the BPI17 OCEL, this means the EFG$_{\text{ss}}$ trained on $672,120$ events, whereas our EFG and HOEG configurations received the original $220,965$ events (considering the train split).

Secondly, subgraph sampling in combination with graph-level prediction implies that the first $k-1$ events in a trace are not predicted. This could greatly affect regression loss, especially as the first set of events in a trace are the most difficult to predict. Crucially, remaining times for the first $k-1$ events are always higher, and usually much higher than the events following. An exclusion of these events from predictive models might then yield a misleadingly good regression score. Particularly MSE (but MAE as well) is affected in a seemingly positive manner, as MSE punishes outliers. For the loan application log, subgraph sampling excludes $\sum_{t \in L}(k-1) \times 1_{\{t_{size} \geq k\}} + t_{size} \times 1_{\{t_{size} < k\}} = 94,527$ events[20]

[19] Where L is any OCEL, t indicates a trace, t_{size} implies number of events in a trace, k equals the subgraph size, and $1_{\{t_{size} \geq k\}}$ represents an indicator function that equals 1 if t_{size} is greater than or equal to k and 0 otherwise.

[20] $1_{\{t_{size} < k\}}$ represents an indicator function that equals 1 if t_{size} is less than k and 0 otherwise.

(24% of the total, cf. Table 9).

Besides affecting regression scores, in some process mining contexts where traces are short, not predicting the first $k - 1$ events of a trace might be unfavorable. In addition, when scalability is crucial, subgraph sampling is not recommended as it requires the global pooling operation which is expensive (as demonstrated by the last two columns of Table 13).

Subgraph sampling does, on the other hand, seem to yield more stable learning curves. This was observed in our trials and with subgraph sampling (see Supplementary Figures, Figure A.1).

Considering these lines of thought, HOEG seems to be the best-performing model when looking at MAE and MSE.

7.2 Order Management Log

As described in Section 5.4.2, the order management OCEL is a small synthetic log, containing $22,367$ events and $8,159$ traces (using lead type extraction based on **item**). Moreover, the object types used (**order**, **item**, and **package**), do not contain object attributes. Besides that, this dataset contains much object interaction, implying individual traces are relatively long.

Upon analyzing the hyperparameter tuning results for OTC we observe a seemingly clear pattern, higher model complexity increases model performance (for both EFG and HOEG). This could be due to the size of the OTC OCEL. When a dataset is too small, a less complex model might not converge fast enough when learning complex patterns from the data. A model with more hidden dimensions might use the extra hidden dimensions to find the relevant patterns earlier in the training process.

Concerning EFG and HOEG in the second experiment, we observe strong signs of overfitting, in both the performance distribution plots and the learning curves. This could be attributed to how this OCEL was generated. It is unknown which probability distributions (e.g. uniform, Gaussian, Poisson) were used to produce the timings of events or values of attributes, neither is it known how traces and relationships were built up. The learning curves and violin plot indicate that the models struggle to find general patterns in the training data that exist in the validation and test data as well. They seem to fit the noise in the training split. This might suggest that the dataset does not contain realistic object-centric event data or contains too few examples.

The results of the baseline experiment (cf. Table 14), could serve as further support for this since the LightGBM and EFG_{ss} models were unable to fit the data as well.

When comparing HOEG and EFG with the baselines, we observe a jump in performance. This suggests that the graph structure does aid the predictive performance. This may be associated with the many object interactions that this log contains. That is, object interactions lead to more complex graph structures, where nodes have multiple incoming or outgoing edges, which are leveraged by the HOEG-based and EFG-based GNNs.

For the OTC OCEL, EFG_{ss} performs worse than HOEG and EFG. In some cases, subgraph sampling removes relevant interactions, resulting in a series of unconnected events. For instance, in our running example (i.a. depicted in Figure 10), taking a subgraph sample of size 4 produces many samples of events that are unconnected (e.g. $\{< e1, e2, e3, e4 >, < e2, e3, e4, e5 >, ...\}$). This could well be the case for the OTC dataset. Therefore, the EFG_{ss} might find difficulties in leveraging information in related events for predicting remaining time of process execution subgraphs, explaining its lower performance.

Interpreting whether the addition of object attributes and explicit modeling of object interactions lifts performance remains challenging. At first, it seems that including objects explicitly introduces noise, considering the slight drop in performance displayed by Figure 19. However, upon further reasoning, one could recognize the questionable choice of event attributes as interference. Specifically, the log contains event attributes `weight`, `price`, which can be argued to be object attributes for any of the object types (**order**, **item**, or **package**) from an ontology point of view. That is, these attributes might not be relevant for every event, with *Payment reminder* as a real example from the OTC log. It would be more fitting to consider `weight` and `price` as attributes of **item**, for instance. Given that these attributes are taken as event attributes, implies they are duplicated across events, implicitly encoding object information into events. This means our EFG might unintentionally leverage object information, depending on the relevance of these attributes.

7.3 Financial Institution Log

The FI OCEL contains $695,694$ events, $31,277$ traces, 3 event attributes, and 14 object attributes (recall Section 5.4.3). The object-centric event data was extracted from a real-life operational workflow management system that supports a complex process that has many external dependencies.

Our first experiment suggested that there is no one hyperparameter setting that works best, for neither HOEG or EFG. This is consistent with the results of the second experiment. Here, the learning curves signify that our higher-order GNN models struggle to learn relevant complex patterns with respect to process remaining time. Though HOEG's learning curve presents itself more stable, the validation loss is not going down by much over the course of the epochs. This could be attributed to the additional data that HOEG includes compared to EFG (i.e. the 14 object attributes). Nonetheless, HOEG does not seem to leverage this information to its advantage, as the EFG attains the best-performing model by a negligible margin (test MAE difference of $0.6961 - 0.6955 = 0.0006$). This might be explained by two factors. First, a meager amount of object interactions is present in the process (compared to the other OCELs). The reason for this is the design of the workflow management system, which defines one case notion. Due to the nature of the process at hand, a business case was formed for connecting these cases to attain an object-centric event log. Ultimately, this implies there is less structural information that the HOEG can leverage. The second factor worth considering is the limited predictive value of both the event attributes and, to a greater extent, the object attributes. It would be reasonable to assume including 14 additional features, lifts the predictive performance of the HOEG-based model. However, as the additional findings (seen in Table 15) suggest, including more features does not improve model performance by much. This suggestion mainly becomes apparent by contrasting the metric scores of the Median model against the others. In doing so, we observe a predictor that does not learn complex patterns from input features approaching the performance of the complex models (test MAE of Median: 0.7702, LightGBM: 0.7286). This indicatively substantiates the potentially limited predictive value of the object attributes.

Furthermore, the baseline experiment demonstrated the EFG-based and HOEG-based models modestly outperform the baselines. Interestingly, the Median baseline approximates the more complex baselines, without having learned complex patterns. This could be attributed to case deadlines that are in place at the respective organization. These have a normalizing effect on the process remaining times. These deadlines are set to feasible dates, based on case complexity, priority, and central tendency measures (e.g. mean, median, or mode) of process execution times. Therefore, it could well be that the Median model performs well on the events of most traces, but lacks the complexity to fit to the events in the traces that exceeded their deadline. The findings support this explanation, as Median achieved a relatively favorable MAPE score, while the test MSE score is the highest (cf. Table 15).

7.4 Heterogeneous Object Event Graph Configuration Recommendations

Configuring HOEG for predictive process monitoring enables integral encoding of OCELs, but it also entails certain risks that warrant careful consideration. Specifically, when modeling a heterogeneous object event graph, it is not immediately apparent which direction event-object edges should take. Therefore, it seems sensible to make them undirected. Upon first thought, this seems perfectly fine. However, when considering the message-passing mechanism in a GNN with two or more message-passing layers, it becomes clear that information is passed from future events to the current. This is problematic, as in a real-life scenario, features about events from the future are not available. Via objects, we are able to leak information from many events in the future to the current one, as objects tend to be related to many events. Therefore, we recommend modeling object node types as source nodes with outgoing edges to event nodes, which are then the destination nodes.

Unexpectedly, this property that HOEG provides introduces an appealing advantage. That is, we can estimate how predictable a process is based on how well a model can learn to predict remaining time while leaking future event information (with respect to the current event). Table 16 highlights, model performance when future event features are used in predicting remaining time for events. We observe a drastic decrease in regression loss for the BPI17 OCEL compared to the experimental results, where the best model achieved a test MAE of 0.4377. For the order management log, the information leaking HOEG configuration mainly shows an increase in performance on the train split. For the test set, a modest loss decrease of $0.5352 - 0.4930 = 0.0422$ is obtained, again showing the model's inability to generalize well to unseen data. For the FI OCEL, the performance increase is negligible ($0.6955 - 0.6951 = 0.0004$ on the test set), compared to the best model on the FI log. Even when information about future events may be used to predict remaining time per event, the higher-order GNN struggles to make accurate predictions. Therefore, these findings suggest the FI process may be highly unpredictable, strengthening previous arguments. Finally, for the FI process, this could be used as an argument to prioritize structuring the process over introducing predictive techniques to aid its operational efficiency.

Table 16. Mean absolute error scores for each dataset encoded using a leaky HOEG configuration.

OCEL	**Mean Absolute Error**		
	Train	**Validation**	**Test**
BPI17	0.0934	0.0927	0.0934
OTC	0.0938	0.5465	0.4930
FI	0.6854	0.6881	0.6951

Throughout our experiments, the same GNN architecture has been instantiated on both EFG and HOEG. As the two encodings are different, there is every hope that an interaction might be at play between GNN design and encoding type. For instance, the higher-order message-passing layers may be favorable for EFG, disregarding the enhanced capabilities of HOEG. Conceivably, other GNN designs, employing different message-passing layers, or a different set of hyperparameters (e.g. fixed ones from Table 8), might be needed to learn useful patterns from the HOEG encoding. Besides that, there is a bias towards a well-suited design for EFG, as our preliminary experiments, used to assemble our initial GNN, were performed only on the EFG encoding. From there, one final architecture was put forward to serve as the deep model in both encodings. Moreover, we utilized a functionality that automatically converts a homogeneous model to a heterogeneous one. This might not be necessarily optimal, as there are at least three other techniques provided by Fey & Lenssen [97] to do this. All three provide more options to include domain-specific configuration optimizations, which will likely yield a better-fitting model for the HOEG encoding.

Also worth noting, is the inherent complexity of HOEG compared to EFG-based deep learning models, as demonstrated by Table 12. This should be considered when drafting designs based on the HOEG encoding.

7.5 Synthesis

The interpretation of the experimental results indicate that HOEG is a promising encoding technique. Compared to EFG, HOEG generally exhibits favorable learning curves. This does not seem to translate into the final model performance per se, as the HOEG encoding only outperforms EFG on the BPI17 log, and closely follows EFG in the OTC and FI OCELs. These results constrain us from being able to draw definitive conclusions on the superiority of either encoding type. Nevertheless, the above sections presented arguments that incite a nuanced answer attributing differing encoding performances to dataset characteristics. BPI17 is a rather structured OCEL, with a decent amount of object interactions, and clear event and object attributes. This likely facilitates good learning capabilities for HOEG, explaining its edge over EFG. OTC is a small dataset, where the mechanisms and probability distributions used for its generation are unknown. Moreover, it could be argued that its event attributes should actually be object attributes, which obscures assessing the superiority of either HOEG or EFG. Finally, the findings suggest that FI OCEL contains noisy attributes and relatively few object interactions, making it difficult to learn useful patterns with respect to remaining time prediction, especially for the HOEG encoding. In light of our experimental results, there is some indication that HOEG may offer a slight advantage over EFG in terms of remaining time prediction on OCELs. This is mainly attributed to its capability to learn patterns from object interactions and object attributes, as this is the chief difference compared to EFG (cf. Chapter 4).

8 Conclusion

In this thesis, we set out to address the central research question: *How to leverage the multi-dimensional data structure given in object-centric event logs in order to predict process remaining time?*. To answer this, four sub-questions aided the discussion. After providing theoretical background, we answer the first two sub-questions through Section 3.2. Existing approaches perform machine learning tasks on OCELs by either projecting the log onto a single case definition, or utilizing object-centric execution extraction to obtain true object-centric traces (*SRQ1*). OCELs host an enhanced data structure compared to traditional event logs. Extant literature [27, 28, 30] takes advantage of this through encoding object-centric process behavior as graphs, including features that describe event-object interactions, capturing object-object interactions, and incorporating object attributes (*SRQ2*). None of the extant OCPPM implementations enjoy this full set of capabilities provided by OCELs. This gave rise to requirements for a solution that addresses the shortcomings of individual approaches. Combining the work of Adams *et al.* [27] and Berti *et al.* [30], we integrated the different capabilities of OCELs into one novel class of feature encodings: heterogeneous object event graphs (*SRQ3*). Through a series of three experiments performed on three datasets, we obtained insight into the performance and scalability of HOEG when predicting process remaining time. HOEG was mainly contrasted with EFG, an encoding type proposed by Adams *et al.* [27] that considers OC-DFGs as its main structure to attach features onto. In terms of performance, HOEG-based GNNs perform better than EFG-based models for well-structured object-centric processes like the loan application process. Additionally, the HOEG encoding type could be expected to excel when attributes in the OCEL are well-defined, and object interactions are prolific, due to event-object interaction being an explicit part of its graph encoding structure (*SRQ4*). Regarding scalability, the results consistently demonstrated that models based on HOEG are more complex than EFG-based models, having more model parameters. Therefore, they tend to be less scalable. Considering this, we propose the heterogeneous object event graph encoding type as a promising technique to leverage the multi-dimensional data structure given in object-centric event logs for predicting process remaining time.

Future work might look into different configurations of HOEG, as we have only evaluated one possible implementation. Other options may include edge features or different heterogeneous graph neural network architectures. Besides that, subgraph sampling techniques could provide better model results, as hierarchical patterns may then be learned better through pooling operations. Another avenue for further research would be exploring different applications of the HOEG encoding, as it is flexible to a variety of tasks. Via HOEG one can make predictions (regression or classification) on any component of the heterogeneous graph: events, objects, edges, or a full graph. This could, for instance, provide interesting applications of outlier detection, extending the work by Berti *et al.* [30], which demonstrates an initial attempt at outlier detection for objects in OCELs. Lastly, the HOEG encoding might be utilized to assess the predictability of an object-centric process, giving possible insights into process maturity.

A Supplementary Figures

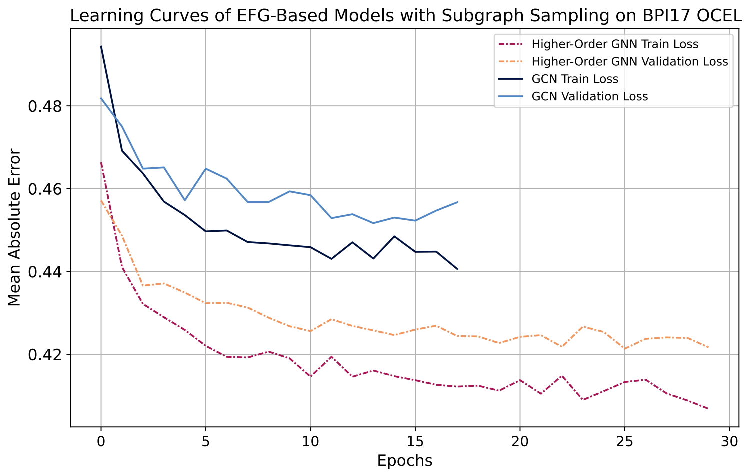

Figure A.1. Training and validation loss learning curves of GCN [27] and higher-order GNN models running on the subgraphed EFG with subgraph size four.

References

1. Van der Aalst, W. M., Barthelmess, P., Ellis, C. A. & Wainer, J. Proclets: A Framework for Lightweight Interacting Workflow Processes. *Int. J. Cooperative Inf. Syst.* **10,** 443–481 (2001).
2. Li, G., de Carvalho, R. M. & Van der Aalst, W. M. P. *Automatic Discovery of Object-Centric Behavioral Constraint Models* in *Business Information Systems* (ed Abramowicz, W.) (Springer International Publishing, Cham, 2017), 43–58. ISBN: 978-3-319-59336-4.
3. Jans, M. & Soffer, P. *From Relational Database to Event Log: Decisions with Quality Impact* in *Business Process Management Workshops* (eds Teniente, E. & Weidlich, M.) (Springer International Publishing, Cham, 2018), 588–599. ISBN: 978-3-319-74030-0.
4. Van der Aalst, W. M. P. *Object-Centric Process Mining: Dealing with Divergence and Convergence in Event Data* in *Software Engineering and Formal Methods* (eds Ölveczky, P. C. & Salaün, G.) (Springer International Publishing, Cham, 2019), 3–25. ISBN: 978-3-030-30446-1.
5. Van der Aalst, W. & Berti, A. Discovering Object-centric Petri Nets. *Fundamenta Informaticae* **175.** 1-4, 1–40. ISSN: 1875-8681. `https://doi.org/10.3233/FI-2020-1946` (2020).
6. Esser, S. & Fahland, D. Multi-Dimensional Event Data in Graph Databases. *Journal on Data Semantics* **10,** 109–141. ISSN: 1861-2040. `https://doi.org/10.1007/s13740-021-00122-1` (June 2021).
7. Waibel, P., Pfahlsberger, L., Revoredo, K. & Mendling, J. *Causal Process Mining from Relational Databases with Domain Knowledge* 2022. arXiv: `2202.08314 [cs.DB]`.
8. Adams, J. N., Schuster, D., Schmitz, S., Schuh, G. & Van der Aalst, W. *Defining Cases and Variants for Object-Centric Event Data* in *2022 4th International Conference on Process Mining (ICPM)* (Aug. 2022), 128–135.
9. Fahland, D. *Multi-dimensional Process Analysis* in (eds Remco, del Río Ortega Adela, Claudio, R.-M. S. D. C. & Dijkman) (Springer International Publishing, 2022), 27–33. ISBN: 978-3-031-16103-2.
10. Park, G., Adams, J. N. & Van der Aalst, W. M. P. *OPerA: Object-Centric Performance Analysis* in *Conceptual Modeling* (eds Ralyté, J., Chakravarthy, S., Mohania, M., Jeusfeld, M. A. & Karlapalem, K.) (Springer International Publishing, Cham, 2022), 281–292. ISBN: 978-3-031-17995-2.
11. Berti, A. & Van der Aalst, W. M. P. OC-PM: analyzing object-centric event logs and process models. *International Journal on Software Tools for Technology Transfer* **25,** 1–17. ISSN: 1433-2787. `https://doi.org/10.1007/s10009-022-00668-w` (Feb. 2023).
12. Nigam, A. & Caswell, N. S. Business artifacts: An approach to operational specification. *IBM Systems Journal* **42,** 428–445 (2003).

13. Segers, I. *Investigating the application of process mining for auditing purposes* 2007.

14. Bhattacharya, K., Gerede, C., Hull, R., Liu, R. & Su, J. *Towards Formal Analysis of Artifact-Centric Business Process Models* in *Business Process Management* (eds Alonso, G., Dadam, P. & Rosemann, M.) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2007), 288–304. ISBN: 978-3-540-75183-0.

15. Hull, R. *Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges* in *On the Move to Meaningful Internet Systems: OTM 2008* (eds Meersman, R. & Tari, Z.) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2008), 1152–1163. ISBN: 978-3-540-88873-4.

16. Cohn, D. & Hull, R. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull.* **32,** 3–9 (2009).

17. Piessens, D. *Event log extraction from SAP ECC 6.0* 2011.

18. Fahland, D., de Leoni, M., van Dongen, B. F. & Van der Aalst, W. M. P. *Behavioral Conformance of Artifact-Centric Process Models* in *Business Information Systems* (ed Abramowicz, W.) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2011), 37–49. ISBN: 978-3-642-21863-7.

19. Lu, X., Nagelkerke, M., Wiel, D. v. d. & Fahland, D. Discovering Interacting Artifacts from ERP Systems. *IEEE Transactions on Services Computing* **8,** 861–873 (2015).

20. Li, G., de Murillas, E. G. L., de Carvalho, R. M. & Van der Aalst, W. M. P. *Extracting Object-Centric Event Logs to Support Process Mining on Databases* in *Information Systems in the Big Data Era* (eds Mendling, J. & Mouratidis, H.) (Springer International Publishing, Cham, 2018), 182–199. ISBN: 978-3-319-92901-9.

21. Fahland, D. in *Process Mining Handbook* (eds Van der Aalst, W. M. P. & Carmona, J.) 274–319 (Springer International Publishing, Cham, 2022). ISBN: 978-3-031-08848-3. `https://doi.org/10.1007/978-3-031-08848-3_9`.

22. Berti, A. & Van der Aalst, W. *Extracting Multiple Viewpoint Models from Relational Databases* in *Data-Driven Process Discovery and Analysis* (eds Ceravolo, P., van Keulen, M. & Gómez-López, M. T.) (Springer International Publishing, Cham, 2020), 24–51. ISBN: 978-3-030-46633-6.

23. Ghahfarokhi, A. F., Park, G., Berti, A. & Van der Aalst, W. M. P. *OCEL: A Standard for Object-Centric Event Logs* in *New Trends in Database and Information Systems* (eds Bellatreche, L. *et al.*) (Springer International Publishing, Cham, 2021), 169–175. ISBN: 978-3-030-85082-1.

24. Detwiler, B. Celonis announces next-generation, MRI process mining technology with Process Sphere™. `https://www.celonis.com/blog/celonis-announces-next-generation-mri-process-mining-technology-with-process-sphere/` (2023) (Nov. 9, 2022).

25. Meyer, J. & Reimold, J. *Associative Intelligence for Object-Centric Process Mining with MPM (Extended Abstract) 1* in (2021).

26. Rohrer, T., Ghahfarokhi, A. F., Behery, M., Lakemeyer, G. & Van der Aalst, W. *Predictive Object-Centric Process Monitoring* 2022. arXiv: `2207.10017 [cs.AI]`.

27. 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* (eds Troya, J. *et al.*) (Springer Nature Switzerland, Cham, 2022), 36–53. ISBN: 978-3-031-20984-0.

28. Galanti, R., de Leoni, M., Navarin, N. & Marazzi, A. Object-centric process predictive analytics. *Expert Systems with Applications* **213,** 119173. ISSN: 0957-4174 (Mar. 2023).

29. Gherissi, W., El Haddad, J. & Grigori, D. *Object-Centric Predictive Process Monitoring* in *Service-Oriented Computing – ICSOC 2022 Workshops* (eds Troya, J. *et al.*) (Springer Nature Switzerland, Cham, 2023), 27–39. ISBN: 978-3-031-26507-5.

30. Berti, A., Herforth, J., Qafari, M. & Van der Aalst, W. Graph-Based Feature Extraction on Object-Centric Event Logs. *Research Square (Research Square)* (Dec. 2022).

31. Barenholz, D. *et al. There and Back Again* in *Application and Theory of Petri Nets and Concurrency* (eds Gomes, L. & Lorenz, R.) (Springer Nature Switzerland, Cham, 2023), 37–58. ISBN: 978-3-031-33620-1.

32. Gori, M., Monfardini, G. & Scarselli, F. *A new model for learning in graph domains* in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* **2** (2005), 729–734.

33. Scarselli, F. *et al. Graph neural networks for ranking Web pages* in *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)* (2005), 666–672.

34. Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. & Monfardini, G. The graph neural network model. *IEEE transactions on neural networks* **20,** 61–80 (2008).

35. Sanchez-Lengeling, B., Reif, E., Pearce, A. & Wiltschko, A. B. A Gentle Introduction to Graph Neural Networks. *Distill.* https://distill.pub/2021/gnn-intro (2021).

36. Bronstein, M. M., Bruna, J., Cohen, T. & Veličković, P. *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges* 2021. arXiv: `2104.13478 [cs.LG]`.

37. Veličković, P. Everything is connected: Graph neural networks. *Current Opinion in Structural Biology* **79,** 102538. ISSN: 0959-440X. `https://www.sciencedirect.com/science/article/pii/S0959440X2300012X` (2023).

38. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86,** 2278–2324 (1998).

39. Liu, C. *et al. Graph Pooling for Graph Neural Networks: Progress, Challenges, and Opportunities* 2022. arXiv: `2204.07321 [cs.LG]`.

40. Defferrard, M., Bresson, X. & Vandergheynst, P. *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering* in *Advances in Neural Information Processing Systems* (eds Lee, D., Sugiyama, M., Luxburg, U., Guyon, I. & Garnett, R.) **29** (Curran Associates, Inc., 2016). `https://proceedings.neurips.cc/paper_files/paper/2016/file/04df4d434d481c5bb723be1b6df1ee65-Paper.pdf`.

41. Kipf, T. N. & Welling, M. *Semi-Supervised Classification with Graph Convolutional Networks* 2017. arXiv: `1609.02907 [cs.LG]`.

42. Wu, F. *et al. Simplifying Graph Convolutional Networks* in *Proceedings of the 36th International Conference on Machine Learning* (eds Chaudhuri, K. & Salakhutdinov, R.) **97** (PMLR, June 2019), 6861–6871. `https://proceedings.mlr.press/v97/wu19e.html`.

43. Monti, F. *et al. Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs* in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017).

44. Veličković, P. *et al. Graph Attention Networks* 2018. arXiv: `1710.10903 [stat.ML]`.

45. Brody, S., Alon, U. & Yahav, E. *How Attentive are Graph Attention Networks?* 2022. arXiv: `2105.14491 [cs.LG]`.

46. Battaglia, P., Pascanu, R., Lai, M., Jimenez Rezende, D. & kavukcuoglu koray, k. *Interaction Networks for Learning about Objects, Relations and Physics* in *Advances in Neural Information Processing Systems* (eds Lee, D., Sugiyama, M., Luxburg, U., Guyon, I. & Garnett, R.) **29** (Curran Associates, Inc., 2016). `https://proceedings.neurips.cc/paper_files/paper/2016/file/3147da8ab4a0437c15ef51a5cc7f2dc4-Paper.pdf`.

47. Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O. & Dahl, G. E. *Neural Message Passing for Quantum Chemistry* in *Proceedings of the 34th International Conference on Machine Learning* (eds Precup, D. & Teh, Y. W.) **70** (PMLR, Aug. 2017), 1263–1272. `https://proceedings.mlr.press/v70/gilmer17a.html`.

48. Battaglia, P. W. *et al. Relational inductive biases, deep learning, and graph networks* 2018. arXiv: `1806.01261 [cs.LG]`.

49. Schlichtkrull, M. *et al. Modeling Relational Data with Graph Convolutional Networks* in *The Semantic Web* (eds Gangemi, A. *et al.*) (Springer International Publishing, Cham, 2018), 593–607. ISBN: 978-3-319-93417-4.

50. Liu, Z. *et al. Heterogeneous Graph Neural Networks for Malicious Account Detection* in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (Association for Computing Machinery, Torino, Italy, 2018), 2077–2085. ISBN: 9781450360142. `https://doi.org/10.1145/3269206.3272010`.

51. Zhang, C., Song, D., Huang, C., Swami, A. & Chawla, N. V. *Heterogeneous Graph Neural Network* in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Association for Computing Machinery, Anchorage, AK, USA, 2019), 793–803. ISBN: 9781450362016. `https://doi.org/10.1145/3292500.3330961`.

52. Wang, X. *et al. Heterogeneous Graph Attention Network* in *The World Wide Web Conference* (Association for Computing Machinery, San Francisco, CA, USA, 2019), 2022–2032. ISBN: 9781450366748. `https://doi.org/10.1145/3308558.3313562`.

53. PyTorch Geometric. *Heterogeneous Graph Learning* `https://pytorch-geometric.readthedocs.io/en/latest/notes/heterogeneous.html` (2023).

54. Friedman, J. H. Stochastic gradient boosting. *Computational Statistics & Data Analysis* **38.** Nonlinear Methods and Data Mining, 367–378. ISSN: 0167-9473. `https://www.sciencedirect.com/science/article/pii/S0167947301000652` (2002).

55. Chen, T. & Guestrin, C. *XGBoost: A Scalable Tree Boosting System* in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Association for Computing Machinery, San Francisco, California, USA, 2016), 785–794. ISBN: 9781450342322. `https://doi.org/10.1145/2939672.2939785`.

56. Dorogush, A. V., Ershov, V. & Gulin, A. *CatBoost: gradient boosting with categorical features support* 2018. arXiv: `1810.11363 [cs.LG]`.

57. Ke, G. *et al. LightGBM: A Highly Efficient Gradient Boosting Decision Tree* in *Advances in Neural Information Processing Systems* (eds Guyon, I. *et al.*) **30** (Curran Associates, Inc., 2017). `https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf`.

58. Teinemaa, I., Dumas, M., Rosa, M. L. & Maggi, F. M. Outcome-Oriented Predictive Process Monitoring: Review and Benchmark. *ACM Trans. Knowl. Discov. Data* **13.** ISSN: 1556-4681. `https://doi.org/10.1145/3301300` (Mar. 2019).

59. Verenich, I., Dumas, M., Rosa, M. L., Maggi, F. M. & Teinemaa, I. Survey and Cross-Benchmark Comparison of Remaining Time Prediction Methods in Business Process Monitoring. *ACM Trans. Intell. Syst. Technol.* **10.** ISSN: 2157-6904. `https://doi.org/10.1145/3331449` (July 2019).

60. Lee, S., Lu, X. & Reijers, H. A. *The Analysis of Online Event Streams: Predicting the Next Activity for Anomaly Detection* in *Research Challenges in Information Science* (eds Guizzardi, R., Ralyté, J. & Franch, X.) (Springer International Publishing, Cham, 2022), 248–264. ISBN: 978-3-031-05760-1.

61. Galanti, R. *et al.* An explainable decision support system for predictive process analytics. *Engineering Applications of Artificial Intelligence* **120,** 105904. ISSN: 0952-1976. `https://www.sciencedirect.com/science/article/pii/S095219762300088X` (2023).

62. Pourbafrani, M., Kar, S., Kaiser, S. & van der Aalst, W. M. P. *Remaining Time Prediction for Processes with Inter-case Dynamics* in *Process Mining Workshops* (eds Munoz-Gama, J. & Lu, X.) (Springer International Publishing, Cham, 2022), 140–153. ISBN: 978-3-030-98581-3.

63. Casati, F., Ceri, S., Pernici, B. & Pozzi, G. *Conceptual modeling of workflows* in *OOER'95: Object-Oriented and Entity-Relationship Modeling: 14th International Conference Gold Coast, Australia, December 13–15, 1995 Proceedings 14* (1995), 341–354.

64. Casati, F., Grefen, P., Pernici, B., Pozzi, G. & Sánchez, G. *WIDE workflow model and architecture* tech. rep. Accessed April 2023 (Technical Report 96-19, University of Twente, 1996). https://www.academia.edu/download/42342330/WIDE_Workflow_model_and_architecture20160207-25096-4cxmnt.pdf.

65. Barthelmess, P. & Wainer, J. *Workflow systems: a few definitions and a few suggestions* in *Proceedings of conference on Organizational computing systems* (1995), 138–147.

66. Van der Aalst, W. M. P., Barthelmess, P., Ellis, C. A. & Wainer, J. *Workflow Modeling using Proclets* in *Cooperative Information Systems* (eds Scheuermann, P. & Etzion, O.) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2000), 198–209. ISBN: 978-3-540-45266-9.

67. Swenson, K. D., Maxwell, R. J., Matsumoto, T., Saghari, B. & Irwin, K. A business process environment supporting collaborative planning. *Collaborative computing* **1,** 15–34 (1994).

68. Bandinelli, S., Braga, M., Fuggetta, A. & Lavazza, L. *Cooperation support in the spade environment: a case study* in *Proceedings of the Workshop on Computer Supported Cooperative Work, Petri nets, and Related Formalisms (14th International Conference on Application and Theory of Petri Nets)* (1993).

69. Hagen, C. & Alonso, G. *Beyond the black box: Event-based inter-process communication in process support systems* in *Proceedings. 19th IEEE International Conference on Distributed Computing Systems (Cat. No. 99CB37003)* (1999), 450–457.

70. Deutsch, A., Hull, R., Patrizi, F. & Vianu, V. *Automatic verification of data-centric business processes* in *Proceedings of the 12th international Conference on Database Theory* (2009), 252–267.

71. Fritz, C., Hull, R. & Su, J. *Automatic construction of simple artifact-based business processes* in *Proceedings of the 12th international conference on database theory* (2009), 225–238.

72. Chen, P. P.-S. The entity-relationship model—toward a unified view of data. *ACM transactions on database systems (TODS)* **1,** 9–36 (1976).

73. Rozinat, A. & Van der Aalst, W. Conformance checking of processes based on monitoring real behavior. *Information Systems* **33,** 64–95. ISSN: 0306-4379. https://www.sciencedirect.com/science/article/pii/S030643790700049X (2008).

74. Fahland, D., De Leoni, M., Van Dongen, B. F. & Van der Aalst, W. M. Many-to-Many: Some Observations on Interactions in Artifact Choreographies. *ZEUS* **705,** 9–15 (2011).

75. Berti, A. & Van der Aalst, W. *StarStar Models: Process Analysis on top of Databases* 2018. arXiv: `1811.08143 [cs.DB]`.

76. Van der Aalst, W. M. P. in *Process Mining Handbook* (eds Van der Aalst, W. M. P. & Carmona, J.) 3–34 (Springer International Publishing, Cham, 2022). ISBN: 978-3-031-08848-3. `https://doi.org/10.1007/978-3-031-08848-3_1`.

77. Adams, J. N. & Van der Aalst, W. M. P. *OCπ: Object-Centric Process Insights* in *Application and Theory of Petri Nets and Concurrency* (eds Bernardinello, L. & Petrucci, L.) (Springer International Publishing, Cham, 2022), 139–150. ISBN: 978-3-031-06653-5.

78. Adams, J. N. & Van der Aalst, W. *Precision and Fitness in Object-Centric Process Mining* in *2021 3rd International Conference on Process Mining (ICPM)* (2021), 128–135.

79. Van der Aalst, W. *Object-Centric Process Mining* tech. rep. Whitepaper (Munich, de, Jan. 2023). `https://assets.ctfassets.net/zmrtlfup12q3/3ocxcGPuVbPOCR1ZFKQ0ON/12e0a52c2c07fcb826e443f9b669fb76/OBJECT-CENTRIC-PROCESS-MINING_WHITEPAPER_FINAL_LD.pdf`.

80. Di Francescomarino, C. & Ghidini, C. Predictive process monitoring. *Process Mining Handbook. LNBIP* **448,** 320–346 (2022).

81. Di Francescomarino, C., Dumas, M., Maggi, F. M. & Teinemaa, I. Clustering-based predictive process monitoring. *IEEE transactions on services computing* **12,** 896–909 (2016).

82. Leontjeva, A., Conforti, R., Di Francescomarino, C., Dumas, M. & Maggi, F. M. *Complex symbolic sequence encodings for predictive monitoring of business processes* in *Business Process Management: 13th International Conference, BPM 2015, Innsbruck, Austria, August 31–September 3, 2015, Proceedings 13* (2015), 297–313.

83. Van der Aalst, W. M. P., Schonenberg, M. H. & Song, M. Time prediction based on process mining. *Information Systems* **36,** 450–475. ISSN: 03064379 (2011).

84. de Leoni, M., Van der Aalst, W. & Dees, M. A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. *Information Systems* **56,** 235–257. ISSN: 0306-4379. `https://www.sciencedirect.com/science/article/pii/S0306437915001313` (2016).

85. Van Dongen, B. *BPI Challenge 2017* en. 2017. `https://data.4tu.nl/articles/_/12696884/1`.

86. Lundberg, S. M. & Lee, S.-I. A unified approach to interpreting model predictions. *Advances in neural information processing systems* **30** (2017).

87. Adams, J. N., Park, G. & Van der Aalst, W. M. ocpa: A Python library for object-centric process analysis. *Software Impacts* **14,** 100438. ISSN: 2665-9638. https://www.sciencedirect.com/science/article/pii/S2665963822001221 (2022).

88. Mensi, A. & Bicego, M. *A Novel Anomaly Score for Isolation Forests* in *Image Analysis and Processing – ICIAP 2019* (eds Ricci, E. *et al.*) (Springer International Publishing, Cham, 2019), 152–163. ISBN: 978-3-030-30642-7.

89. Barbon Junior, S., Ceravolo, P., Damiani, E. & Marques Tavares, G. *Evaluating Trace Encoding Methods in Process Mining* in *From Data to Models and Back* (eds Bowles, J., Broccia, G. & Nanni, M.) (Springer International Publishing, Cham, 2021), 174–189. ISBN: 978-3-030-70650-0.

90. Morris, C. *et al.* Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence* **33,** 4602–4609. https://ojs.aaai.org/index.php/AAAI/article/view/4384 (July 2019).

91. Kingma, D. P. & Ba, J. *Adam: A Method for Stochastic Optimization* 2017. arXiv: 1412.6980 [cs.LG].

92. You, J., Ying, Z. & Leskovec, J. *Design Space for Graph Neural Networks* in *Advances in Neural Information Processing Systems* (eds Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. & Lin, H.) **33** (Curran Associates, Inc., 2020), 17009–17021. https://proceedings.neurips.cc/paper_files/paper/2020/file/c5c3d4fe6b2cc463c7d7ecba
Paper.pdf.

93. James, G., Witten, D., Hastie, T. & Tibshirani, R. *An introduction to statistical learning* (Springer, 2013).

94. Brownlee, J. *What is the Difference Between a Parameter and a Hyperparameter?* July 2017. https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/.

95. Li, G., Medeiros de Carvalho, R. & van der Aalst, W. M. *Configurable Event Correlation for Process Discovery from Object-Centric Event Data* in *2018 IEEE International Conference on Web Services (ICWS)* (2018), 203–210.

96. Paszke, A. *et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library* in *Advances in Neural Information Processing Systems* (eds Wallach, H. *et al.*) **32** (Curran Associates, Inc., 2019). https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

97. Fey, M. & Lenssen, J. E. *Fast Graph Representation Learning with PyTorch Geometric* 2019. arXiv: 1903.02428 [cs.LG].