VaryMinions

Fortz, Sophie; Temple, Paul; Devroey, Xavier; Heymans, Patrick; Perrouin, Gilles

Published in:
Empirical Software Engineering

Publication date:
2024

Document Version
Peer reviewed version

Empirical Software Engineering manuscript No.
(will be inserted by the editor)

VaryMinions: Leveraging RNNs to Identify Variants in Variability-intensive Systems' Logs

Sophie Fortz · Paul Temple · Xavier Devroey · Patrick Heymans · Gilles Perrouin

Received: date / Accepted: date

Abstract From business processes to course management, *variability-intensive* software systems (VIS) are now ubiquitous. One can configure these systems' behaviour by activating options, *e.g.*, to derive variants handling building permits across municipalities or implementing different functionalities (quizzes, forums) for a given course. These customisation facilities allow VIS to support distinct relevant customer requirements while taking advantage of reuse for common parts. Customisation thus allows realising both scope and scale economies. Behavioural differences amongst variants manifest themselves in event logs. To re-engineer this kind of system, one must know which variant(s) have produced which behaviour. Since variant information is barely present in logs, this paper supports this task by employing machine learning techniques to classify behaviours (event sequences) among variants. Specifically, we train Long Short Term Memory (LSTMs) and Gated Recurrent Units (GRUs) recur-

Sophie Fortz has been partially supported by the EPSRC project on Verified Simulation for Large Quantum Systems (VSL-Q), grant reference EP/Y005244/1, the EPSRC project on Robust and Reliable Quantum Computing (RoaRQ), Investigation 009 Model-based monitoring and calibration of quantum computations (ModeMCQ), grant reference EP/W032635/1, by the Fonds de la recherche scientifique (FRS-FNRS) via a FRIA grant and by the EOS VeriLearn project, under grant No. O05518F-RG03

Gilles Perrouin is an FRS-FNRS Research Associate.

S. Fortz (orcid.org/0000-0001-9687-8587)
King's College London, London, United Kingdom
NADI, Faculty of Computer Science, University of Namur, Namur, Belgium
E-mail: sophie.fortz@kcl.ac.uk

P. Temple (orcid.org/0000-0002-8276-0593)
Univ Rennes, CNRS, Inria, IRISA
E-mail: paul.temple@irisa.fr

X. Devroey (orcid.org/0000-0002-0831-7606) · P. Heymans · G. Perrouin (orcid.org/0000-0002-8431-0377)
NADI, Faculty of Computer Science, University of Namur, Namur, Belgium
E-mail: xavier.devroey@unamur.be · patrick.heymans@unamur.be · gilles.perrouin@unamur.be

rent neural networks to relate event sequences with the variants they belong to on six different datasets issued from the configurable process and VIS domains. After having evaluated 20 different architectures of LSTM/GRU, our results demonstrate that it is possible to effectively learn the trace-to-variant mapping with high accuracy (at least 80% and up to 99%) and at scale, *i.e.*, identifying 50 variants using 5000+ traces for each variant.

Keywords Configurable processes · Recurrent Neural Networks · Variability-Intensive Systems · Variability Mining · Software Product Lines

1 Introduction

Business processes capture the activities of every profit or non-profit, public or private organisation, coordinating humans and software to collectively deliver value. As organisations evolve, new needs appear, *e.g.,* covering electric scooters for an insurance company or handling a change in the law about reimbursing travel expenses at the university. These needs lead to the emergence of *process variants*, differing in their control flow or performance while having commonalities with the original processes. Process variants or *configurations* are specific combinations of the system's options. We consider process executions stored in event logs, where an *event trace* (or trace) is an ordered sequence of events. To explore process reengineering opportunities, it is necessary to identify which variant(s) may have produced a given trace. Existing *variant analysis* [119] techniques do not answer this question but cover the inverse operation, *i.e.,* focusing on the differences between identified variants. This problem is not restricted to business processes and naturally extends to *variability-intensive systems*, which change their behaviour in response to the (de)activation of some *options*. Examples of variability-intensive systems include Software Product Lines (SPLs) [6, 103], operating systems kernels [93, 111], code generators [14, 121], or web-based frameworks [57, 108]. Validating these systems is difficult because enumerating all variants, whose number can grow exponentially with the number of options, is generally infeasible [57]. In this context, locating variations is essential for any reengineering endeavour [9]. Black-box testing techniques can also benefit from this information to *e.g.,* sample which variants should be tested first [57].

To support these activities, in this article, we train Recurrent Neural Networks (RNNs) [107, 110] architectures with different hyperparameters (loss and activation functions among others) to predict the candidate variant(s) that could have produced a given event trace. We make the following contributions:

(i) the first variability-aware approach, which we called VaryMinions, to map execution traces to variants of a system;
(ii) a detailed account on the usage of Long Short Term Memory (LSTMs) [66], and Gated Recurrent Units (GRUs) [21], two RNN architectures, on six different datasets, describing business processes and course management system variants;

(iii) four datasets openly available and based on Claroline [30, 32, 33] and containing $2 * 10$ and $2 * 50$ configurations with $5,000$ traces per configurations;
(iv) a characterisation of the intrinsic learning difficulty for variability-intensive systems.

Methodology. For the first contribution, we showed empirically that VaryMinions can distinguish 50 variants from $5,000+$ event traces per variant. In our second contribution, we successfully determine the variant(s) responsible for generating an event trace with high accuracy ($> 80\%$), regardless of whether the GRU or LSTM model is employed. To measure the learning difficulty, we defined and computed a metric based on the amount of behaviour shared amongst event traces.

Open Science Policy. We also provide a replication package [47] with an implementation of our approach using two common Python frameworks reusing RNNs implementations(namely Tensorflow [29] and Keras [22]) as well as presenting all the results of our experiments.

These contributions extend our preliminary research published at the MaLTeSQuE 2021 workshop [46]. While our previous paper focused solely on business processes, this article adds a new source of datasets issued from the VIS domain: Claroline [30, 32, 33], a course management system that was reverse-engineered from an instance in-use at the University of Namur. We derive four new datasets from this newly added system, forming a much more challenging learning problem (up to 50 variants instead of 5), and we assess the effect of sampling (random uniform vs dissimilarity-based) on the outcome. In addition, we reran all our previous experiments and the new ones at one of the Belgian universities' HPC facilities. We also refactored the VaryMinions source code to ease its reuse and make it more configurable. To summarise, the added value of this extension comes from:

(i) four new and more complex datasets from the VIS domain;
(ii) a discussion about the effect of sampling on this classification task;
(iii) a refactored implementation of VaryMinions.

Section 2 introduces process mining, VIS and RNNs. Section 3 motivates the use of VaryMinions. Section 4 gives an overview of the proposed solution, while Section 5 presents the datasets and the experimental setup with more details. Section 6 gives the results of our evaluation. Section 7 discusses certain factors influencing our experiments, such as hyper-parameter variability and alternate labelling of variants. Section 8 presents related work, and finally, Section 9 wraps up the paper.

2 Background

Our work tackles the problem of tracing back the system variant that produced some event logs. This is an issue common to process variants and variability-

intensive systems. We address it by relying on techniques coming from the Deep Learning community. In the following, we introduce these different concepts.

2.1 Process Variants

Nowadays, many organisations work with multiple (business) processes in parallel that can highly depend on environmental and human factors. For instance, a business process can be influenced by regional laws, available resources, the size of the organisation, *etc.* Most of them share common behaviours meaning that for one general business process, one can define several process variants, each one behaving (slightly) differently from the other variants. Similar process variants gather in process lines or process families and can be modelled using different formalisms [106].

Analysing the specificities and commonalities of process variants allows scale economies and helps practitioners to improve the general business process, define new variants or maintain existing ones [119].

For process understanding and reverse engineering purposes, one commonly inspects execution logs. Indeed, they contain valuable information on the process behaviour in production. If the process owns several variants, one must know which variant(s) are involved in a behaviour of interest. Unfortunately, event logs do not usually contain information about a specific variant (or set of variants) which (could have) produced the sequence of events (*i.e.,* the event trace). This can prevent practitioners from understanding why this behaviour occurs for one variant and not another. In this paper, we address the problem of identifying *process variants* that have (potentially) shown a specific behaviour, based on a given *event trace*. This mapping information is key in various re-engineering activities such as variant process mining [119]. However, these activities are beyond the scope of this paper.

To demonstrate the feasibility of our variant process identification learning approach, we use two datasets that gather execution traces of business processes. They both come from the Business Process Intelligence Challenge, a yearly challenge organised since 2011 to stimulate process mining research on real-life datasets.[1] We selected two editions, modelling process variants: the first one from the year 2015 and the second from the year 2020.

2.2 Variability-Intensive Systems

Process families belong to the vast and heterogeneous category of Variability-Intensive Systems (VISs). These are software-based systems that exist in many variants to address the diversity of customer needs and usage contexts. Structured approaches, like Software Product Lines (SPLs) [104], facilitate the design, development and quality assurance of such systems. They consider a global base of software artefacts for a family of software systems, and allow to

[1] `https://www.tf-pm.org/competitions-awards/bpi-challenge`

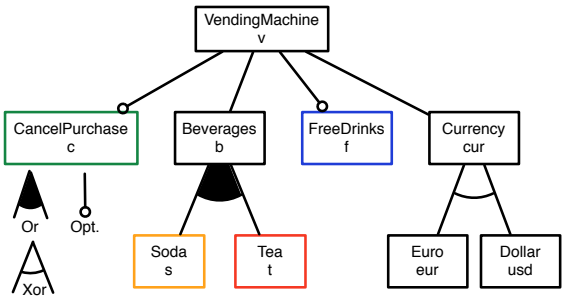


Fig. 1: VIS Feature Model of a beverage vending machine [24, 30].

produce *variants* through the (de)activation of options (also called *features* in the VIS world).[2] Reasoning at the family level rather than at the single system level yields significant economies of scale and quality improvements.

Variability Modelling The variability of a VIS is usually decomposed using a tree-like structure called a (VIS) Feature Model (FM) [72, 109]. An FM represents the common and variable aspects of the system. For instance, Figure 1 presents the FM of a simple configurable beverage vending machine. The machine sells either soda or tea (or both) in euros or dollars, and may optionally support cancelling purchases and providing free drinks. As the number of possible variants increases exponentially with the number of options available, such compact representation of the variability of a VIS enables various kinds of analysis, including counting the number of possible variants, detecting dead options that can never be selected, *etc.* For instance, the vending machine of Figure 1 counts already 24 possible (distinct) configurations. [3] This number is very small compared to real-world VISs. For example, the Claroline case, which we will introduce in Section 5.2.2, has more than 5 million possible configurations for 44 options.

Behavioural Modelling Complementary to FMs, *Featured Transition Systems* (FTSs) [24] are designed to represent compactly the behaviour of a VIS. An FTS is a transition system where each transition is labelled using (VIS) feature expressions (*i.e.,* a Boolean formula referring to its options) to indicate which valid configurations of the VIS can execute the transition. For instance, Figure 2 presents the FTS of the beverage vending machine of Figure 1. As can be seen from the feature expressions, only specific configurations can execute some transitions: *e.g.,* only vending machines with the free (`f`) option enabled can

[2] To avoid any confusion between *VIS feature*, *i.e.,* a functionality of a software system, and *machine learning feature*, *i.e.,* a property characterising an entity, we will refer to the former as *option* (or sometimes *VIS feature*) and to the latter as *feature*.

[3] To avoid any confusion between *VIS configuration*, *i.e.,* a combination of software options, and *neural network configuration*, *i.e.,* a selection of hyperparameters characterising a network, we will refer to the former as *configuration* or *variant* and to the latter as *parameterisation*.

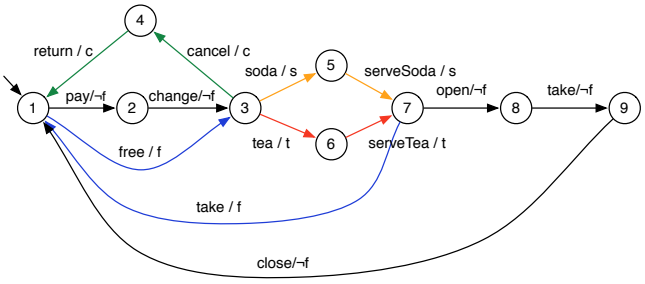


Fig. 2: VIS Feature Transition System of a beverage vending machine [24, 30]

execute the `free` transition from state 1 to state 3. As for FMs, FTSs provide a way to represent compactly the behaviour of all the different configurations of a VIS.

In this paper, we rely on the FM and FTS of Claroline, a highly configurable course management platform previously used at the University of Namur, to simulate executions of different configurations of a real system. This simulation offers a way to generate event logs in a controllable way without requiring running a large number of variants of the system. The Claroline FM and FTS were defined by Devroey *et al.* [30, 32, 33], based on the logs of the implementation used at the University of Namur collected over 9 months.

2.3 Deep Learning and Recurrent Neural Nets

As explained previously, the number of possible variants grows exponentially with the number of VIS options. Similarly, the number of traces a system can generate is supposed to be infinite. These observations command the use of automatic reasoning instead of manual inspections. In particular, we rely on machine learning and deep learning techniques.

Deep Learning (DL) is a subset of machine learning techniques. They remain statistical techniques, but the main difference is that machine learning techniques rely on predefined features (or characteristics) compactly representing data. Historically, domain experts defined the relevant features and the procedures to extract them from raw data. In contrast, DL techniques can infer such features automatically while training but at the cost of more computational resources and time. In the last decade, DL techniques efficiently performed different tasks and new applications such as image processing, assistance in driving for autonomous vehicles, board gaming such as playing Go, sound processing, text processing, automatic translation, *etc.*

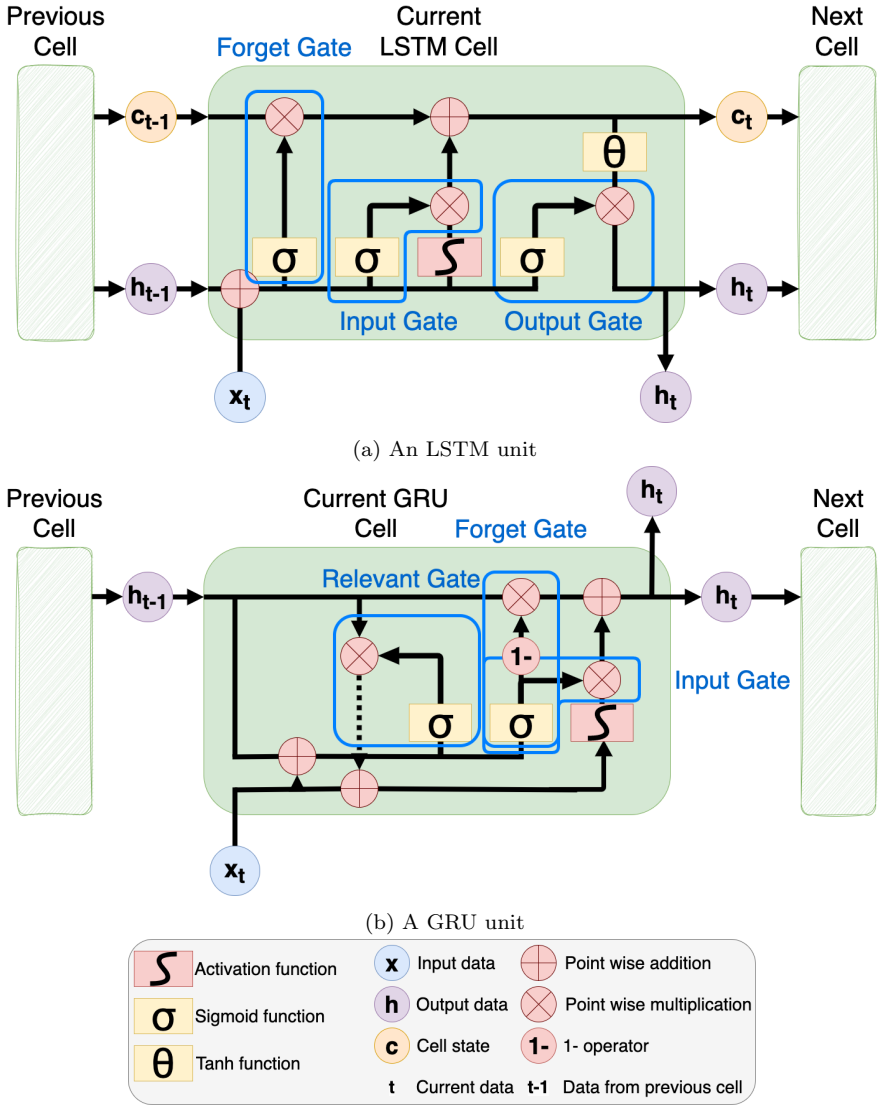
Different families of machine learning algorithms exist: decision trees or random forests, support vector machines, linear regressors, neural networks, *etc.* Thanks to their capability to model and handle complex relations, neural networks are at the centre of attention of DL techniques. There are various neural network architectures, each adapted to a specific task. For instance,

convolutional neural networks excel at image processing while recurrent neural networks (RNNs) [107, 110] handle data sequences (such as text or speech).

Previous works applied RNNs to execution traces to predict the next event or the final execution state. [40, 118]. When data sequences are too long, *vanilla* RNNs may face the so-called *vanishing or exploding gradient* problem [65]. Indeed, weights from the first layers may rarely be adjusted since, during training, the back-propagation mechanism re-injects prediction errors backwardly in the network starting from its output layer so that it can ultimately provide the right outcome. Because one injects errors from the output, they tend *to vanish* and never reach the first layers leaving them unchanged. Conversely, the gradient can grow exponentially, yielding intractable computations. Two RNN architectures deal with longer sequences and long-term dependencies: Long-Short Term Memory (LSTM) [66] and Gated Recurrent Unit (GRU) [21]. These architectures alleviate gradient issues [23, 65] by using gates to regulate the data flow and keep specific long-term data in memory. RNNs are composed of multiple *units* (sometimes referred to as cells), which can convey data from one to another. Typically, RNNs start with an embedding layer that transforms input data into multi-dimensional vectors.

Figure 3 depicts an example of an LSTM unit (left) and a GRU unit (right). Inside one unit, gates regulate the data flow, deciding what data to keep and what to forget. Mathematically, gates are functions (*e.g.,* sigmoid) expressing the amount of data to keep. We can define several types of internal gates for different purposes. An LSTM unit (Figure 3a) is composed of three different state variables and three different gates. The variables represent respectively the input of the unit (*i.e.,* the matrix computed by the embedding layer, called x_t in the figure), the output (called h_t), and the unit state (called c_t). The latter acts as the long-term memory of the network, registering data from previous units to pass through the next ones. Forget gates (on the left of the Figure) are used to convey data from the previous unit directly to the next one. In particular, it may set some values to 0, making the network forget this data. The input gate (in the middle) defines how much data should be treated in the current unit. The final output of a unit travels through the output gate (on the right of the Figure). To avoid gradient explosion, LSTM units use a *tanh* function (above the output gate) to keep data in a small range of value (*i.e.,* between -1 and 1). In GRU (Figure 3b), input and forget gates are merged (on the right of the Figure) and there is no output gate. Consequently, the output and unit state variables are also combined into a unique variable (named h_t in the Figure). GRU also offers a new type of gate (in the middle of the figure) expressing how relevant data from the previous unit is for the current unit.

LSTMs and GRUs are efficient text classifiers, *e.g.,* [75, 85]. In this work, we want to create a mapping between execution traces that are a succession of events occurring in a specific order and configurations of a system that, supposedly, can produce them. In this context, an event does not appear randomly but depends on the previous succession of events. Sometimes directly from the few previous ones, sometimes because of an event that occurs way earlier in the trace. Thus, using LSTM and GRU architectures seems appropriate. Further-



(a) An LSTM unit

(b) A GRU unit

Fig. 3: A unit of LSTM versus a unit of GRU

more, because of this dependency in the sequence of events, we consider traces as text, *i.e.,* an ordered sequence of symbols that follows a given grammar.

While RNNs are usually good fits to work on Natural Language Processing (NLP) tasks, there is little work trying to use RNNs in the context of technical documents or software specifications. Li *et al.* conducted a systematic literature review on extracting variants from text specifications [80].

None of the reviewed works relied on RNNs but used other classification models (decision trees, association rules, *etc.*). Recently, Arganese *et al.* investigated ambiguity in natural requirements as variability points [7], but the mapping concerns words rather than complete sequences.

3 Motivation: Behaviour-driven VIS Reverse-engineering via Black-box Learning

Over the past two decades, researchers have been focused on modelling the behaviour of SPLs for design and analysis purposes. Various paradigms for modelling SPL behaviour, such as Featured Transition Systems [24] and Featured Finite State Machines [56], have been defined. However, engineers typically manually create these models, which is time-consuming, error-prone, and not suitable for complex VISs. Recent efforts [26, 27] have attempted to automate this model creation process, but it is still in its early stages.

Most approaches to learning VIS behaviour rely on and extend Dana Angluin's seminal L^* algorithm from 1987 [5]. This algorithm aims to infer a single system's behaviour in a black-box and active manner, relying solely on execution traces obtained on the fly. In this case, access to the source code is unnecessary, but interaction with the System Under Learning (SUL) is essential. L^* follows a simple metaphor. The *Learner* constructs hypothesis models of the system by posing queries to the *Teacher*, who serves as a middleware between the Learner and the SUL. The Teacher can either validate the hypothesis or provide a counterexample if it is invalid, helping the Learner to update its hypothesis.

To adapt Angluin's algorithm and accommodate variability, existing approaches [26, 27] introduce post-processing steps. For instance, learning each product variant and progressively merging them is one approach. However, this approach becomes impractical when dealing with a large number of variants. Another approach, instead of considering variants individually, would be to consider learning the VIS in a family-based fashion [44, 45]. In both cases, it is essential to relate Angluin's queries and counterexamples to configurations. Existing mappings are incomplete, as they rely on partial observations of the system. We assume that the Teacher only possesses knowledge of previously observed SULs, with all new configurations being unknown. Hence, in this scenario, a configuration prediction technique is required.

Existing SPL reverse engineering techniques usually assume the presence of an accurate FM. This is a strong hypothesis. Usually, FMs are built from requirements which are known to be ambiguous and partly implicit. FM reverse engineering approaches also have limitations in terms of completeness and soundness. It is much easier to assume a set of configurations especially in reverse engineering scenarios. Therefore, we aim for a solution that does not require an FM. In our context, we can only rely on the list of features (but without explicitly stating the constraints between them) and a list of the configurations used for classification.

To map configurations to variant products, white-box approaches rely on the source code or use a combination of software implementation artifacts and logs [91, 92]. We place ourselves in a strict black-box context in which the source code is not available. This is the case for the business processes we analysed. Therefore, we focus on execution logs only. These logs do not directly relate event traces with variants. Indeed, Cândido *et al.* [19] have pointed out that preemptively logging detailed information would result in enormous log files, reaching several terabytes, which would impede effective analysis.

Since extensive mapping information is not available, we propose to employ supervised machine learning to tackle the following challenge: *how can we classify new incoming traces (previously unseen) to multiple variants?*

4 VaryMinions Overview

Figure 4 provides an overview of VaryMinions' architecture. The input data *(a)* are a set of available execution traces. For training, traces are associated to the set of system variants that can produce them. The inputs first pass through an Embedding layer *(b)* that transforms the sequences of events into a vector of indexes (to make the representation more compact and to ease their processing). The embedding layer creates a structured space in which indexes that occur in a similar context are close. In this new representation space, indexes become vectors, and initial traces become tensors composed of numerical weights. This homogeneous representation allows performing mathematical operations on those weights through the rest of the network.

The embedding layer is also configurable, *e.g.,* we need to specify the number of dimensions of the representation space and the number of dimensions in the output tensors. We keep the number of dimensions the same in input and output to avoid combining different input dimensions into one output dimension. We then link this layer to the RNN layer *(c)*, which is instantiated with either LSTM or GRU units to learn the relationships between elements of the tensors. Again, this layer is configurable, in particular with the number and usable kinds of units (detailed in Section 5.3).

There exist unidirectional and bidirectional [110] units. Unidirectional layers only consider the processing of the sequence in one direction (from start to end). In contrast, bidirectional layers also handle the other direction (from end to start), which can be helpful in language processing. In our case, traces are fully available at training time. Reading them forward and backward can help grasp long-term relations between events. Because of our analogy with text, we use bidirectional units [110] only.

Then the network continues with one Dense layer *(d)* preparing for classification. We made the number of units in this layer the same as the number of classes (*i.e.,* configurations). The output of the network *(e)* is a vector of 1s and 0s whose number of elements is equal to the number of configurations of the system. This vector classify the trace into one or more configuration(s).

In this vector, 1s state that associated configurations can generate the input trace and 0s that they cannot.

For instance, let us take a simple system with three configurations. The output vector is thus of size three. If our prediction model outputs the vector $[1, 1, 1]$, it predicts that all the configurations can execute the input trace. In another case, the output vector is $[0, 1, 0]$. Then, only the second configuration is able to produce the input trace, *etc.*. One should note that our models cannot provide the output vector $[0, 0, 0]$ since the RNN selects at least the configuration with the highest score.

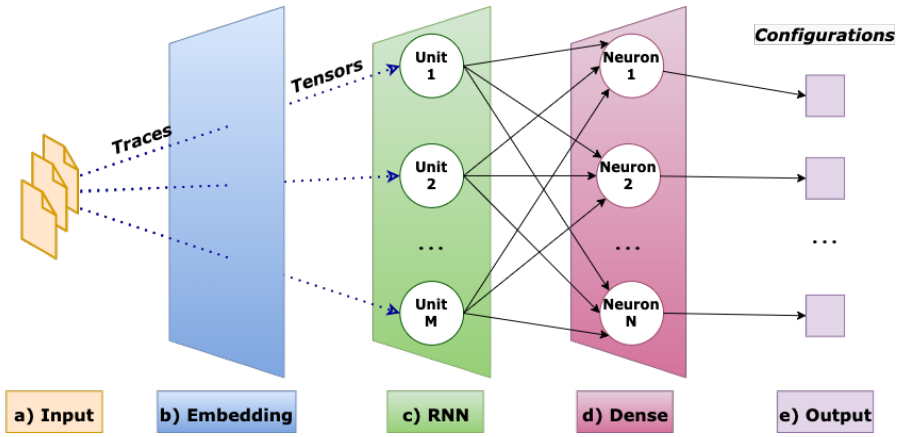


Fig. 4: Description of the VaryMinions architecture

5 Evaluation Protocol

In the following, we describe our evaluation protocol to validate that we can learn which variants may have produced an execution trace. First, we state the research questions that drive this experimentation before describing the creation and annotation of our datasets. Then, we explain how we instantiated VaryMinions regarding our specific context. Finally, we present the running setup and the evaluation metrics.

5.1 Research Questions

We state the following research questions concerning the multi-classification of execution traces among the different VIS variants:

RQ1 *How accurately can we identify process variants based on their traces?*
This question addresses the efficiency of our approach. To the best of our

knowledge, this is the first attempt to use RNNs to learn such a mapping. Thus, we cannot compare it with the state of the art. Instead, we expect the RNNs to be at least better than random classifiers (accuracy higher than $> 50\%$).

RQ2 *What is the performance of LSTMs versus that of GRUs for process traces classification?* We would like to know which model architecture is the most appropriate for this task, if any.

5.2 Datasets selection and preprocessing

We use six different datasets that we divide into two groups. The first group contains the 2015 and 2020 editions of the Business Process Intelligence Challenge (BPIC). Each dataset contains event logs, describing different executions of configurable processes:

- *BPIC15 (DS1)* represents building permit applications in five municipalities, each one corresponding to a process variant [38]; and
- *BPIC20 (DS2)* gathers data from the travel reimbursement process at the Eindhoven University of Technology (TU/e), where variants correspond to different kinds of documents to be managed [37].

The second group consists of four datasets containing event logs describing executions of different variants of *Claroline* [30, 32], an online course management system used at the University of Namur until 2018. Claroline was the main communication channel between students and lecturers, with approximately $7,000$ users. Its architecture is plugin-based. Depending on needs, one can deploy new variants at runtime.

5.2.1 Business Process Intelligence Challenge (BPIC)

The original BPIC datasets (from [37, 38]) contain only valid and complete traces and other information. We prune the logs to keep only the process variant ID, the trace ID and the sequence of events. To cope with different trace lengths, we apply padding (*i.e.,* filling traces with other meaningless events and using a mask to know where the processing should stop). Trace duplicates are removed, and since multiple variants can produce the same trace, we encode the variants into a binary vector (where the size matches the number of variants) that serves as a label. A value of one at the i-th index of the vector denotes that we observed at least once the trace associated with variant i. Traces associated with all variants have thus a vector full of ones. In the end, each trace is associated with one or more variants (*i.e.,* classes). We expect the RNN models to learn these associations to predict the variant(s) for an unlabelled trace. We wrote this preprocessing procedure in Python as part of VaryMinions' implementation [47].

As described in Table 1, DS1 contains $5,542$ traces after preprocessing, with a maximum of 154 events per trace. The five process variants are fairly equally

represented since they contain 1,108 traces on average, with a minimum of 828 and a maximum of 1,350. Therefore, DS1 is well-balanced. DS2 contains 2,074 traces after preprocessing, with 5 process variants and a maximum of 90 events per trace. The least and most represented process variants contain 89 and 1,478 traces respectively, with an average of 415 traces per variant. Therefore, the dataset is imbalanced, suggesting it is harder to learn from accurately.

Table 1: Overview of the preprocessed datasets used in our experiments. Class-specific metrics (cols 3–5) represent (i) the number of traces per class, (ii) the percentage of traces assigned specifically to this variant in the dataset, and (iii) the percentage of traces shared by this variant and at least another one.

Dataset	Class Id	# Traces	% Variant-specific behaviour	% Shared behaviour
DS1 (BPIC15) 5,542 traces	**Munic. 1**	1170	99.658	0.342
	Munic. 2	828	99.638	0.362
	Munic. 3	1350	99.778	0.222
	Munic. 4	1049	99.905	0.095
	Munic. 5	1153	99.827	0.173
DS2 (BPIC20) 2,074 traces	**Int'l Decl.**	753	30.013	69.987
	Dom. Decl.	99	100	0.0
	Permit Req.	1478	64.344	35.656
	Prepaid	202	90.099	9.901
	Req. For Pay.	89	77.528	22.472
DS3 (Clar. Dis. 10) 50,000 traces	**Variant 1**	5000	100	0
	...	...	...	...
	Variant 10	5000	100	0
DS4 (Clar. Rand. 10) 50,000 traces	**Variant 1**	5000	100	0
	...	...	...	...
	Variant 10	5000	100	0
DS5 (Clar. Dis. 50) 250,000 traces	**Variant 1**	5000	100	0
	...	...	...	...
	Variant 50	5000	100	0
DS6 (Clar. Rand. 50) 250,000 traces	**Variant 1**	5000	100	0
	...	...	...	...
	Variant 50	5000	100	0

To better characterise the learning complexity, Table 1 shows the number of traces per class (*i.e.,* variant) and the overlap (*i.e.,* percentage of variant-specific and shared behaviour) between classes. The number of traces provides a first indication of the learning difficulty: more traces generally yield a more accurate network once trained. DS1 contains equally represented classes with limited overlap ($< 0.5\%$ in the last column), while DS2 is less balanced in how classes are represented and how they are interleaved, denoting a shared behaviour between multiple variants. In particular, for DS2, there is a big overlap between the *International Declaration* and the *Permit Request* variants, and between the *Prepaid Travel Cost* and the *Request For Payment* variants, while the *Domestic Declaration* variant is completely separated.

5.2.2 Claroline

Claroline is a highly configurable web-based system whose behaviour depends on a set of activated options. In total, Claroline contains 44 options leading to more than $5,406,700$ unique variants. Handling such a large configurable system is not trivial as it requires deriving different variants and executing them in various ways to trigger different behaviours and collect, format, and process the corresponding event logs. Setting up such pipelines is hard and outside the scope of this paper. For those reasons, we decided, instead of executing the actual system, to simulate executions of different variants using a Featured Transition System (FTS) capturing the behaviours of different configurations of Claroline. The FTS was reverse-engineered by Devroey *et al.* [30, 32] from a 5.26 Go Apache webserver log containing $45,210,987$ entries collected from January 2013 to September 2013 using a bigram inference method. The final FTS consists of 106 states and $2,053$ transitions.

Simulations. The simulation of a given Claroline configuration works as follows. First, the FTS is *projected on the configuration* (*i.e.,* pruned) to keep only the subset of behaviours that can effectively be executed by the configuration. The result of that process is a classical transition system, describing a subset of the behaviours of Claroline. Second, the traces associated with the configuration are produced using random walks in the transition system. We generated $5,000$ traces per configuration. To avoid infinite traces (*e.g.,* in case of a loop in the transition system), we also limited the size of a trace to 300 events. We relied on VIBeS [31, 34], a model-based testing tool for highly-configurable systems, to project the FTS and generate the traces.

We relied on two different strategies to select the different simulated Claroline configurations: random selection and dissimilarity-based selection. The random selection consists in selecting a set of (valid) configurations using a dedicated generator ensuring a random distribution of the selection. In our case, we used CMSGen [54], a fast uniform-like sampler. CMSGen comes with a default parameterisation, which we reused as is.[4] Unlike random, dissimilarity-based selection [62] picks configurations in such a way that they are as dissimilar as possible when considering their selected options. For our evaluation, we used PLEDGE [63], a search-based dissimilarity-driven configuration selection tool. We selected the default parameterisation of PLEDGE, with one minute per generation. We have set the number of configurations to simulate to 10 and 50. This way, we can go beyond the difficulty provided by the BPIC datasets and check that our method can run when the number of configurations is higher. While 50 is still small compared with the number of possible unique variants of Claroline (*i.e.,* $> 5,000,000$), it is closer to a realistic setting.

Event logs datasets. We have derived the four different event logs datasets based on the following sets of configurations of Claroline:

[4] See `https://github.com/meelgroup/cmsgen` for details.

Claroline Dissimilar 10 **(DS3)** regroups execution traces of 10 different configurations of Claroline, selecting the most dissimilar sets of options. This dataset should lead to more discriminated traces and better classifications.

Claroline Random 10 **(DS4)** gathers traces from 10 different instances of Claroline, randomly chosen to have a more realistic dataset.

Claroline Dissimilar 50 **(DS5)** is similar to DS3, but with 50 configurations to allow more diversity.

Claroline Random 50 **(DS6)** is similar to DS4, but with 50 configurations.

For each of these datasets (DS3 to DS6), the output of this generation process is a file containing $5,000$ traces per configuration that we can use as an input for VaryMinions. In our case, we thus have either $50,000$ traces per file (for 10 configurations) or $250,000$ traces per file (for 50 configurations), as shown in Table 1. The last two columns of this table show systematically 100% of variant-specific behaviour and 0% of shared behaviour for Claroline datasets, meaning that for each trace, at least one action is specific to one variant of Claroline. This is due to the use of a sampler for selecting the configurations, giving very little control over the traces overlap. Due to the huge amount of possible variants (*i.e.,* $> 5,000,000$), the chance to find any shared behaviour between multiple variants is almost zero.

5.3 RNN Parameterisations

As we said before, because we use sequences of events, we investigate the use of RNNs to learn to which configuration(s) we can associate a trace. More specifically, we focus on LSTMs and GRUs. As for many DL models, hyperparameters must be defined. Because there are so many, we decided to vary only a few of them to try to understand how much impact they may have on learning. We focused on the functions that are used inside the networks and that may impact the quality of the predictions. We also manually selected a subset of hyperparameters that we fixed to a specific standard value. Hyperparameters and their values are described in detail hereafter and summed up in Table 2.

Number of hidden layers. One specific aspect that impacts the learning capabilities of neural networks is their topology. Since the traces are short compared to text documents, we decided to use networks with only one hidden layer. It may avoid potential overfitting, that can emerge from more complex structures (*e.g.,* auto-encoder) while offering satisfactory prediction performance.

Units. In our previous work [46], our experiments used different numbers of units regarding the RNN layer *(c)*. This number affects the topology of the network and may help to grasp more complex concepts if this number increases. Yet, having too many units on a layer may lead to dealing with redundant information that will deteriorate the final prediction performances of the network [49]. On the contrary, a layer with a smaller number of units may not have the capability to grasp interesting information which may also

harm the prediction performances [49]. Based on our previous experiences, we decided to set the number of units to 30 which has shown relatively good performances while limiting the training time.

Training set, batch size and epochs. Other hyperparameters can be set affecting the training time and the optimisation of the many different parameters (*e.g.,* weights between layers and units) of the networks. Common hyperparameters to set are the ratio of data used to train the model and those used to evaluate the performance of the model; the size of the batch of data that the model will have to deal with during training, which may mitigate overfitting; and the number of time the model will optimise parameters over the whole training set (*i.e.,* the number of epochs). Each of these hyperparameters was set as follows:

(i) the percentage of the data used for training is set to 66% of the whole dataset which is a common value in the ML community, the remaining traces are used in the test set to assess the generalisation performances of the trained models;
(ii) we set the batch size to 128, which is adapted to the dataset size;
(iii) we set the number of epochs to 20 to avoid overfitting. In our preliminary evaluations (evaluated between 10 and 50 epochs), a plateau was reached after approximately 15 epochs. We finally set the number of epochs to 20, to allow for small increases in accuracy.

Activation functions. Activation functions are defined at the level of units (*i.e.,* neurons) and respond to an input signal. If the signal is strong enough, the neuron is activated and the output is also high. Though different activation functions can be used for each neuron, it is usual to define an activation function for an entire layer. We have used a Rectified Linear Unit (ReLU) function on the hidden layer RNN layer (*i.e., (c)* in Figure 4) to alleviate the vanishing gradient problem. Regarding the Dense layer *(d)*, we experimented with two common activation functions that are sigmoid and hyperbolic tangent (tanh). Both are shown in Figure 5. The main difference between both is their definition domain which affects how they handle negative input values. The sigmoid function is defined over $[0; 1]$ meaning that as the values get closer to $-\infty$ the neuron is closer to being non-activated at all (*i.e.,* the output signal is 0) while as the input values are getting larger the response is also getting larger. When the input value is 0, the response is 0.5. On the other hand, tanh is defined over $[-1; 1]$. It may be useful to take into account negative correlations and when the input value is 0, the response is also 0. Using one or the other may affect the "strength" of the signal that will reach the last layer for classification in turn affecting which class (*i.e.,* configuration) will be recognised.

Loss functions. Loss functions are used during training to optimise the weights of the networks by back-propagating errors. We have used three loss functions already implemented in tensorflow [5], namely Binary Cross-Entropy (with and

[5] `https://www.tensorflow.org/api_docs/python/tf/keras/losses`

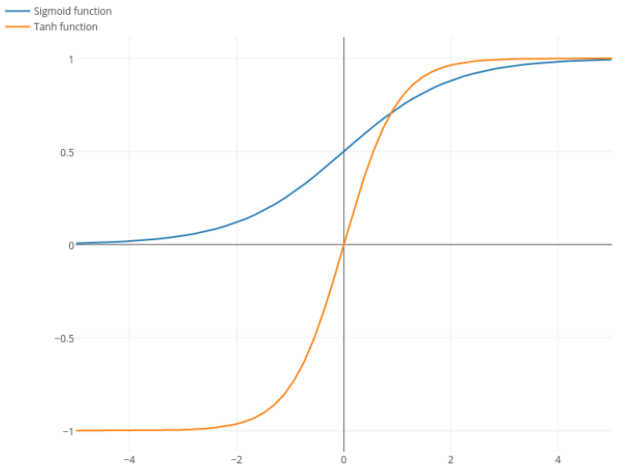


Fig. 5: Sigmoid (blue) and tanh (orange) function responses represented by the Y-axis depending on the input signal (X-axis).

Table 2: Hyper-parameters settings

Hyperparameter	Considered values
Type of Classifier	GRU, LSTM
# Units	30
% Training Set	66%
Batch Size	128
# Epochs	20
Activation Function	sigmoid, tanh
Loss Function	Bin-CE, Bin-CE logits, MSE, Weight_Jaccard, Manhattan

without logits, respectively named hereafter Bin-CE and Bin-CE logits) and the Mean Squared Error (MSE). Logit is defined as the inverse function of the sigmoid. We also implemented two custom loss functions: a variant of the Jaccard distance [69] (named Weight_Jaccard hereafter), and the Manhattan distance between two vectors. The motivation for these two last functions is that because a single trace might be assigned to different process variants, the error should be defined considering a comparison of elements of vectors but not from a single value. This difference between two vectors should define a distance score. The Manhattan distance (sometimes called L1 norm) computes the sum of absolute differences between each element of the two vectors (*i.e.,* in this case, the process variants). The Jaccard distance assesses how many equal elements of two vectors are over their size. We have implemented a variant of the Jaccard distance to cope with floating-point values generated by the networks. The Jaccard distance was employed to evaluate trace dissimilarity in variability-intensive systems (*e.g.,* [35]). Further discussions about the use and characteristics of these loss functions are provided in Section 7.2.

5.4 Model Training

We decided to use only a training set and a test set in our evaluation due to the number of available execution traces. The training and performance evaluation process is done as follows: i) the entire dataset is randomly split into training and test sets. We have used the Keras function *train_test_split* [6] that ensures the data distribution of classes among the two sets are similar. ii) A model is trained using the training set. iii) Its prediction performances are evaluated on the test set. To mitigate biases in our analyses we decided to train and evaluate the performances of each parameterization ten times on each dataset. For each run, the whole training and performance evaluation process is started again (*i.e.,* splitting into training and test sets, training the model, and evaluating its performances). The fact that the splits are done each time mitigates the chances to train and evaluate a model on the best sets solely. Not only that it may change the data used for training the model but it may change the order of appearance too, which may have an impact on the trained model.

5.5 Evaluation Metrics

This work is the first attempt to use RNNs to classify execution traces among variants of a system. One of our goals is to evaluate if such a DL technique is appropriate for this task. We thus computed four different standard metrics that are *Accuracy*, *Precision*, *Recall*, and *F1-score*.

Accuracy. To evaluate the quality of the models that have been learnt, the usual metric is the *Accuracy* measure. Accuracy is defined as

$$Acc = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \tag{1}$$

It is a standard measure in the ML community to assess how well a model performs from a high-level point of view. It has the advantage to be easily computable and it can also be used to refer to the number of wrong predictions (*i.e.,* $1 - Accuracy$).

However, when classes are not well balanced (*i.e.,* the number of traces is way more important for at least one class than for others), *Accuracy* may hide some important information as the number of correct predictions for the classes with more data may take the lead on the number of wrong predictions of the others resulting in a high ratio. To mitigate this aspect from our analysis, we only consider other measures.

Precision. One usual metric to account for the performances of a prediction model is its *precision*. It can be calculated for each class as follows:

$$Precision = \frac{Number\ of\ correct\ predictions}{Number\ of\ predictions\ for\ the\ class} \tag{2}$$

[6] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`

where $Number\ of\ predictions\ for\ the\ class$ is the number of correct predictions and the number of additional data that are wrongly predicted to belong to the class (*i.e.,* false positives).

We gathered all these individual *precision* measures into a global one using a weighted average:

$$Prec = \frac{\sum_{i=1}^{c} Precision_i * supp_i}{Number\ of\ data} \tag{3}$$

where c is the number of classes, $Precision_i$ the *precision* measure for class i, and $supp_i$ the number of data with label i.

Recall. Similarly to the *precision*, the *recall* is also standard to report on the predictions of a model. It can also be calculated for each class and is defined as follows:

$$Recall = \frac{Number\ of\ correct\ predictions}{Number\ of\ labeled\ data\ for\ the\ class} \tag{4}$$

where $Number\ of\ labeled\ data\ for\ the\ class$ is the number of data labelled with the class under consideration.

Similarly to the *precision*, we computed a weighted average to get an overall *recall* measure for the model:

$$Rec = \frac{\sum_{i=1}^{c} Recall_i * supp_i}{Number\ of\ data} \tag{5}$$

where c is the number of classes, $Recall_i$ the *recall* measure for class i, and $supp_i$ the number of data with label i.

F1-score. The F1-score is obtained through the harmonic mean of *precision* and *recall* to get an overview of the global performances of the model in one single measure. The *F1-score* in the case of two classes is defined as:

$$F1 - score = 2\frac{precision * recall}{precision + recall} \tag{6}$$

Again, we can apply this calculation on each class and average with a weight equal to the proportion of data of each class in the (test) set to get an overall value for the model. The three last metrics were computed by the *precision_recall_fscore_support* [7] function in Scikit Learn before being averaged. Also, we did compute confusion matrices [8] for each class. They are available in our replication package.

[7] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html`

[8] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html`

5.6 Running infrastructure

Finally, Table 2 shows: 2 $models \times 1$ $\#units \times 1$ $\%training\ set \times 1$ $batch\ size \times 1$ $\#\ epochs \times 2$ $activation\ funtions \times 5$ $loss\ functions = 20$ different parameterisations of RNNs. We conducted these experiments on three different HPC facilities hosted by the CÉCI.[9] On the first cluster, called Dragon1, we used 1 CPU with 8 cores per task (Intel Sandy Bridge, E5-2650 processors at 2.00GHz) with a Tesla Kepler accelerator (K20m, 1.1 Tflops, float64). For runs on Dragon2, we used 1 CPU with 12 cores (Intel SkyLake, Xeon 6126 processors at 2.60 GHz) associated with an NVidia Tesla Volta V100 accelerator (5120 CUDA Cores, 16GB HBM2, 7.5 TFlops, double precision). On the third cluster, Hercules, we had access to 1 CPU with 8 cores per task (Intel Sandy Bridge, Xeon E5-2660 processors at 2.20 GHz) with an NVidia GeForce accelerator (RTX 2080 Ti, 7.5 TFlops, double precision). Each CPU has been allocated 3 GB of RAM. All our scripts are written in Python 3, with the Keras and Tensorflow frameworks for deep learning. Our replication package is openly available [47].

We conducted a 10-fold validation, where for each fold we randomly defined different train and test sets. For each fold, we evaluate the model by computing all the metrics described in Section 5.5. In total, running our 20 different network parameterisations with 10 repetitions on the six different datasets, resulted in $20 \times 10 \times 6 = 1,200$ runs and more than 151 days of execution. The time needed for a single execution varies between 44 seconds and 13 hours depending on the dataset and the GPU type.

6 Evaluation Results

In this section, we answer our two research questions separately based on:

- box-plots presented in Figures 6 to 9, showing accuracy, precision, recall and F1-score for each parameterisation of each dataset;
- a multi-comparison statistical analysis (see Figure 10), using Friedman's test with Nemenyi's post-hoc analysis;
- Tables presented in Appendix A, with average and standard deviation for the four computed metrics.

All the results (*i.e.,* for each execution of each parameterisation) are also available in our replication package [47], including the code to compute the metrics, box-plots and statistical tests.

6.1 Performance (RQ1)

Table 3 reports the averaged accuracy (over 10 runs) of the 20 considered parameterisations of RNNs, over the 6 datasets (*i.e.,* 120 models). We group

[9] `http://www.ceci-hpc.be/`

Table 3: Number of RNN parameterisations reaching predefined accuracy thresholds. We take into account 120 parameterisations. Accuracies are averaged over 10 runs on each dataset. Each cell indicates the number of times a given RNN model type (column) reaches the threshold (row). The last column gives the total (LSTM+GRU) per accuracy range.

Accuracy	LSTM	GRU	Total
accuracy > 70%	23	21	44
50 < accuracy < 70%	8	7	15
accuracy < 50%	29	32	61

into LSTM and GRU (columns) and the average accuracies into three categories according to a predefined threshold: i)below 50% where we consider models as performing worse than a random assignment to system variants and thus useless; ii) between 50% and 70% where we consider models as being slightly better than random assignments; iii) over 70% where we consider the models as performing well. Out of the 120 models, 44 RNNs parameterisations (first row) yield an accuracy higher than 70%, 15 are between 50% and 70%, and the remaining 61 have an accuracy below 50%. It means that nearly half of the considered models perform better than a random guess, a majority of which (*i.e.,* 44 parameterisations out of 59) performs well in our context.

The highest averaged accuracy for datasets BPIC15 and BPIC20 (top of Figure 6, or Tables 4 and 5 in Appendix) is 88% and 87% respectively with high stability (*i.e.,* low standard deviation). On BPIC20, only five parameterisations out of twenty do not reach 50%. Even better, for BPIC15 only five parameterisations are lower than 70% of accuracy. Top of Figures 7, 8 and 9 confirm these results by giving similar values for precision, recall and F1-score respectively.

Despite the complexity of Claroline datasets, at least one parameterisation obtains an averaged accuracy of 80% for each dataset. For Claroline Dissimilar 10 (middle left of Figure 6 and Table 6 in Appendix), the top parameterisation reaches 99.6% and 4 different parameterisations are above 85%. Claroline Random 10 and Random 50 (middle and bottom right of Figure 6, or Tables 7 and 9 in Appendix) also have several parameterisations above 80%, and their top one gets over 95% of accuracy. Claroline Dissimilar 50 (bottom left of Figure 6, or Table 8 in Appendix) has only one row with an averaged accuracy of 80% and only two other rows above 70%. Among the remaining, 15 rows are below 30%.

Note that, for Claroline Dissimilar 50, boxplots are either spread out or centred on low values (Bottom left of Figure 6). Moreover, the top three rows also report a high standard deviation for the accuracy (*i.e.,* higher than 0.13 and up to 0.38, in Table 8 in Appendix). It highlights that the results lack stability: at least one execution out of ten does not belong to the same value range. Regarding Claroline Random 10 and Claroline Random 50 (middle and bottom right of Figure 6), the top three parameterisations show very compact boxplots with few outliers. This suggests a more stable accuracy, as confirmed

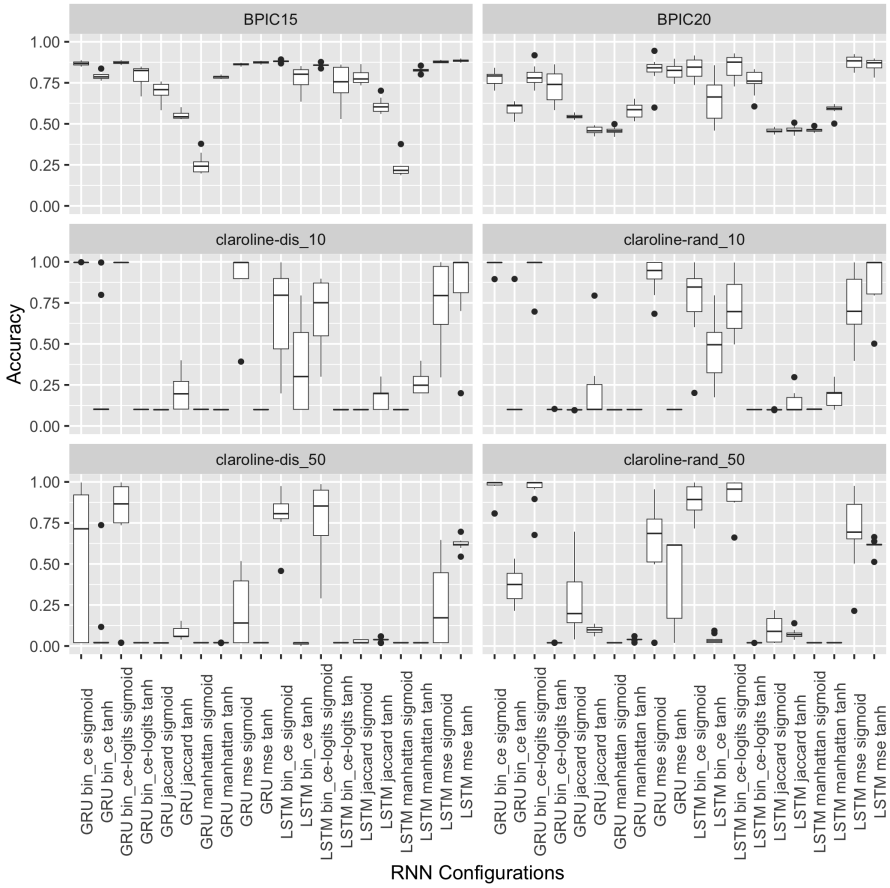


Fig. 6: Boxplots showing the Accuracy over 10 runs for each parametrisation of each dataset.

by a standard deviation between 0.03 and 0.11 for the accuracy (Table 7 and Table 9 in Appendix). The top two parameterisations of Claroline Dissimilar 10 (Table 6) both show an accuracy higher than 0.99 and a standard deviation lower than 0.001, demonstrating very stable results.

Overall, the number of configurations of the Claroline system (10 or 50) neither influences averaged accuracy nor the standard deviation. Similarly, how we sample (random-based or dissimilarity-based) configurations does not impact accuracy. As for BPIC15 and BPIC20, the other metrics (precision, recall and F1-score presented respectively in Figure 7, 8 and 9) only confirm this analysis as they follow the same tendencies.

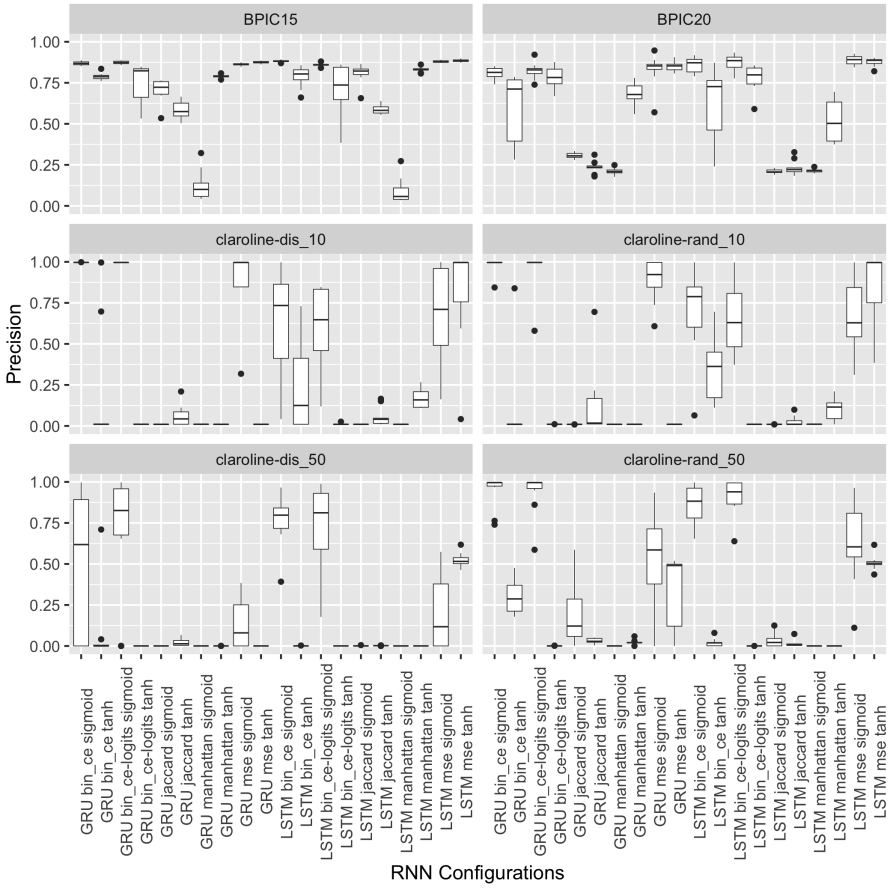


Fig. 7: Boxplots showing the Precision over 10 runs for each parametrisation of each dataset.

Answer to RQ1 (performance): we were able to train RNNs providing an accuracy above 70% (and even above 80%) for each dataset. On Claroline Dissimilar-10 the accuracy can reach 99.6%. The associated standard deviations can be small (*i.e.*, < 0.01) but they are usually higher with the Claroline datasets, regardless of the number of configurations used or the way we select them. Yet, these results suggest there is potential to use RNNs to automatically classify newly generated execution traces among the variants of a system rather than trying to do it manually.

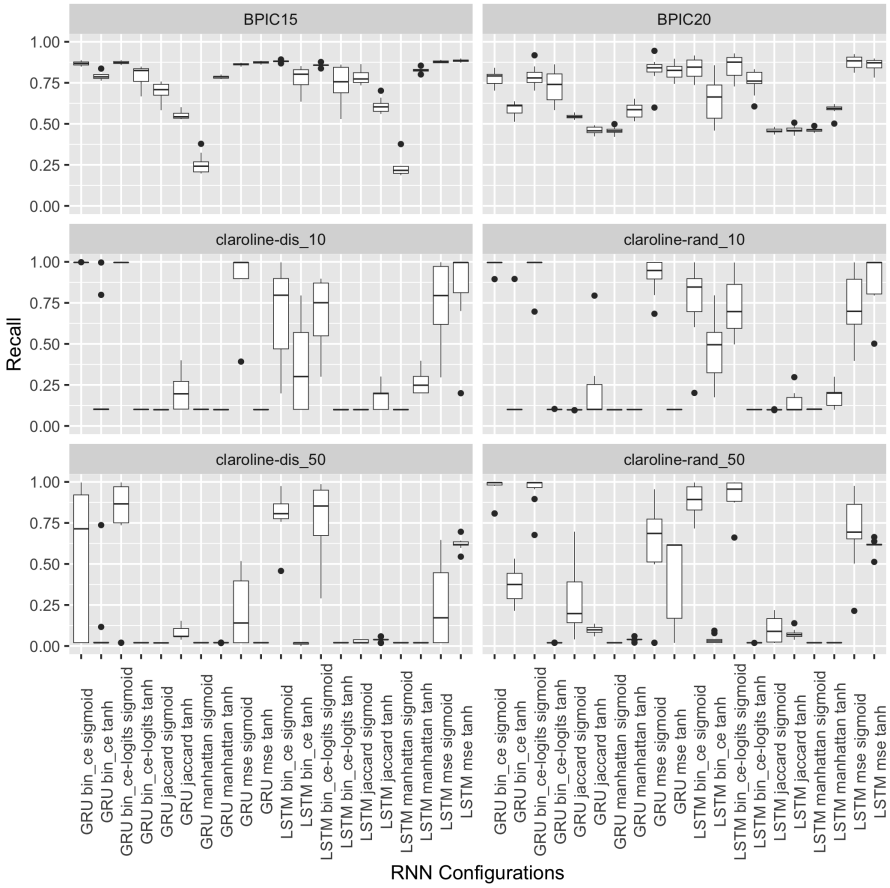


Fig. 8: Boxplots showing the Recall over 10 runs for each parametrisation of each dataset.

6.2 LSTM vs. GRU (RQ2)

Our second RQ is about the prevalence of each type of RNN. Can LSTM or GRU be considered better and should be preferred in this context? To answer this question, we hypothesize that one kind of RNN prevails over the other one and performs a multi-comparison statistical analysis of each 20 RNN parameterisations on all 6 datasets. We used a Friedman's non-parametric test [48] with a significance level $\alpha = 0.05$. This test ranks parameterisations over accuracy and then determines if the differences between parameterisations are significant. We further complete this result with Nemenyi's post-hoc procedure [70, 96] indicating the statistical differences between parameterisations. This procedure can determine equivalence classes, regrouping parameterisations that are statistically similar regarding accuracy.

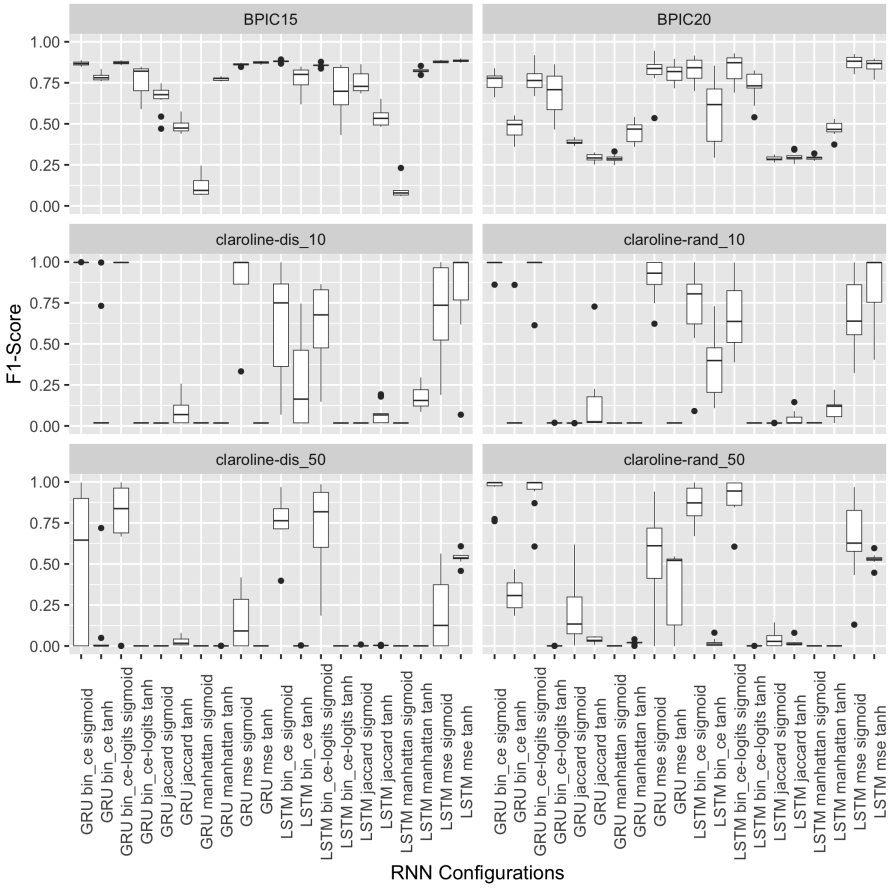


Fig. 9: Boxplots showing the F1-Score over 10 runs for each parametrisation of each dataset.

Figure 10 shows the results of Nemenyi's test. After executing Friedman's test, we obtain a p-value under 0.001, meaning that there is a statistical difference between the accuracy of some of the parameterisations. Nemenyi's post-hoc procedure shows that the minimum distance between two statistically different groups of parameterisations (*i.e.,* the critical distance) is 3.828. The bottom of Figure 10 shows the seven best parameterisations over all the datasets. Statistically, they are equivalent and perform better than the remaining others (belonging to a different group).

Four pairs of loss and activation functions out of ten seem to stand out from the test. They are:

- MSE and sigmoid
- binary cross-entropy and sigmoid
- binary cross-entropy with logits values and sigmoid

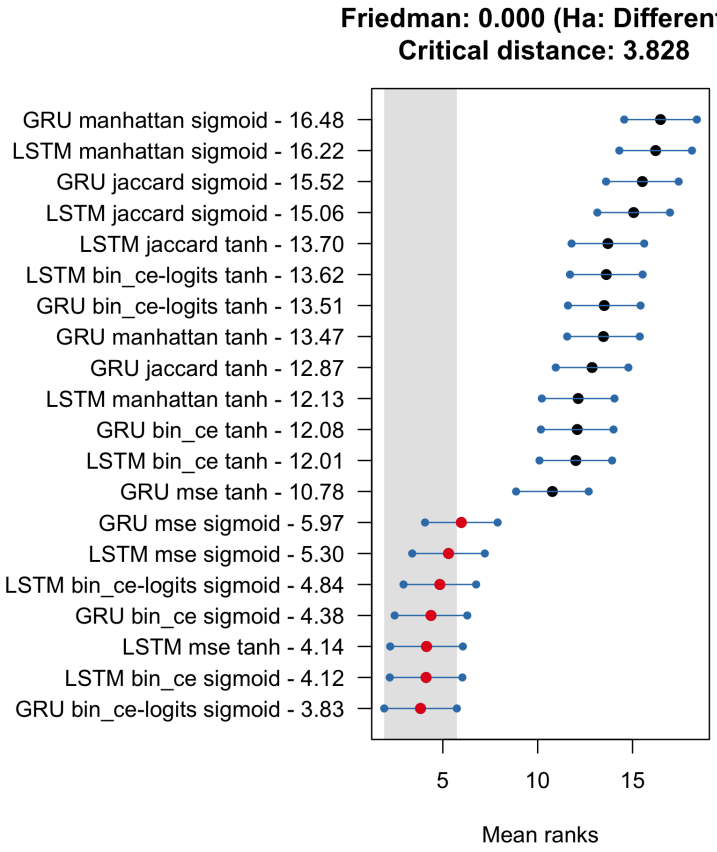


Fig. 10: Result of Friedman's statistical test along with Nemenyi's post-hoc analysis over all datasets and parameterisations

– MSE and tanh (with LSTM only)

For most datasets, these parameterisations can predict the right set of variants with an accuracy greater than (or very close to) 70% (confirmed by Figure 6 and Appendix A). However, sometimes a combination also gives bad results. It is the case with MSE and sigmoid, both with LSTM and GRU, where accuracy does not exceed 0.25 for Claroline Dissimilar 50 (Table 8).

We can observe that the dedicated loss functions (Manhattan and Jaccard distance) give terrible results compared to the other "classical" loss functions. Nemenyi's procedure (Figure 10) assigns them the highest mean ranks. For all Claroline datasets, the accuracy is always under 30%. On BPIC15 and BPIC20, they give better results (respectively up to 82% for BPIC15 and up to 58% for BPIC20) but still lower than the other loss functions.

Regarding the activation functions, our statistical analysis shows that 6 out of 7 best parameterisations use sigmoid instead of tanh.

Nemenyi's procedure shows that LSTMs are present in 4 of the top parameterisations and GRUs in 3 of them. However, these parameterisations are indistinguishable regarding accuracy (*i.e.,* critical distance < 3.823). Appendix A shows that the best parameterisation is an LSTM for BPIC15, BPIC20 and Claroline Dissimilar 50, but it is a GRU for the three other datasets. GRU is also the model giving the best accuracy amongst all datasets with up to $99,6\%$ for Claroline Dissimilar 10 (Table 6). Moreover, the count of LSTMs and GRUs in each category of Table 3 shows similar numbers and indicates that using GRU or LSTM does not influence the results.

Answer to RQ2 (classifiers): In the top combinations of all six datasets, we observed mixed performance of LSTMs and GRUs, with no absolute winner. A statistical comparison showed that 4 out of 7 parameterisations use LSTMs, without any significant difference between the 3 parameterisations using GRUs. Moreover, GRU gives better results on 3 of the datasets (Claroline Dissimilar 10, Claroline Random 10 and Claroline Random 50). Hence, we cannot conclude the prevalence of one over the other for these six datasets. Moreover, our results suggest using the sigmoid activation functions rather than tanh.

7 Discussion and Future Work

This section discusses threats to validity that we identified and other aspects driving our future works.

7.1 Threats to Validity

Internal validity. The datasets we used contain clean and consistent traces (*i.e.,* they omit inconsistent traces when the system crashes or an unexpected event occurs). The BPIC community ensure this property [37, 38] or by the use of an FTS model and the VIBeS framework [31, 34] as a trace generator (for Claroline). For a new VIS, a preprocessing step should take care of trace consistency (*i.e.,* a trace should capture a complete user session). It does not entail that the dataset captures the whole system's behaviour. Indeed logs and models inferred from them represent a partial view of it.

To assess the difficulty of the learning process (*i.e.,* being able to map logs to variants while sharing parts of the traces), we defined our own metrics (see last two columns of Table 1). This definition is inspired by our experience in analysing VISs where commonalities and variabilities between behaviours are key to the analysis. These metrics come from the analysis of the dataset only and give a better understanding of the intrinsic complexity of the learning problem. While they are fairly simple and high-level, they can be computed quickly but

do not provide fine-grained differences (as the Levenshtein distance [79] would do but at the cost of longer computations). Finding the right trade-off between simplicity to compute and precision is left to future work.

The deep learning community is very active, leading to new types (or combinations of types) of models appearing every few months, especially for image processing tasks, where competition is fierce. It is less so regarding models dedicated to time sequences. We selected LTSMs and GRUs for their ability to deal with temporal sequences and to evade the vanishing or exploding gradient issue.

We evaluated 20 distinct parameterisations of RNNs over six datasets. We designed them regarding our goal, based on our previous work [46]. However, since exhaustive coverage of the hyperparameter space is impossible, we may have missed some relevant parameterisations. Dealing with the inherent variability of hyperparameters is a research challenge *per se*.

A way to optimise the parameterisations is to use hyperparameter tuning techniques such as random search or auto-ML [94]. We did not use any in this work but tried to scope the parameterization space with a manual approach similar to a grid search approach [46]. One motivation for this choice is that VaryMinions is the first effort to use RNNs to classify execution traces for variants of systems. Thus, we were not interested in finding the best-performing model (aka the goal of hyperparameter tuning). Rather, we show that, within a reasonable effort, finding a suitable RNN model parameterisation performing well is possible.

External Validity. Compared to our initial results [46], we augmented our experimental setup using Claroline, a VIS. Though our method applies to two different application domains, we cannot ensure that it generalises to all configurable systems. We used six different datasets having different characteristics that mitigate the fact that our method may work only on simple datasets. Among the ones we have used, some were taken from existing competitions (BPIC), and some were generated from scratch (Claroline) allowing us to vary and control the complexity of the learning by modifying the amount of traces available and/or the number of configurations to deal with. Let us note that reverse-engineered models from logs necessarily form an incomplete representation of the behaviour of the system. Indeed, logs cannot capture all execution traces that are often infinite for any real-world system. Besides, we do not guarantee that our cases cover the whole spectrum of VIS, given their diversity and widespread.

A problem when using DL techniques in such a context is imbalanced representations in the training set. The training set may contain fewer occurrences of a configuration of a system or a process (*e.g.,* because of lower popularity or fewer actions need to be performed) with the risk that the trained model may neglect classification errors involving these configurations since they can be considered as rare events. While the Claroline datasets were generated in such a way that imbalance representations were limited, we had no control over the BPIC datasets. They exhibit configuration imbalance but our RNN

models coped with it (*i.e.,* successfully classifying traces belonging to these configurations). Thus, we took no further actions to mitigate this aspect. Of course, class imbalance impact is case-specific.

Replicability. To prevent potential replicability issues, our implementation of VaryMinions and all the results presented in this paper are publicly available on Zenodo for long-term storage [47].

7.2 Hyperparameter Variability

The use of RNNs in this context requires carefully dimensioning the network and considering many parameterisations that can influence classification performances. In what follows, we discuss two elements that may influence them.

Loss functions. We use the mean squared error (MSE) to evaluate prediction errors while training a network, which is traditionally preferred when tackling a regression problem. However, Hui and Belkin [67] showed that this assumption lacks solid theoretical foundations and that MSE is suitable for classification. In particular for NLP applications, where MSE usually outperforms cross-entropy.

The choice of the loss function is tricky since we need to take care of multiple aspects: the formalisation of the problem (*e.g.,* single or multi-label, regression or classification) or the way to compute errors. Even when trying to choose the loss function according to these points (*e.g.,* Jaccard distances have been used to solve SPL problems, as in [35]), our results indicate that the MSE works surprisingly well. Given the importance of a loss function on the observed performance, experimenting with additional loss functions appears promising. For example, the focal loss [84], which penalises more misclassified instances than well-classified ones, is a perspective that we aim to follow.

The interplay of Losses and Activations. We deliberately chose to explore custom loss rather than activation functions. Loss functions are easier to adapt to the problem at hand (by quantifying how far we are from the true label) acting on the network output. Yet, activation functions and loss functions have distinct roles in the network, and they should be considered complementary and not independent. Both are important in the learning process. Activation functions come after every layer inside the network and, together with the weights, set the importance of a specific neuron through the propagation of the network. Loss functions are defined at the end of the network and are used to provide the final class(es). Loss functions are also used to back-propagate the classification errors through the network to optimize the weights in the training phase. From this short description, it is clear that activation and loss functions' interactions affect the model performance. The former may block or lower the importance of discriminative information if incorrectly set while the latter defines the distance from the labels, from which the network optimises itself. Hence, assessing the impact of one type of function alone is not possible. Further investigations on which combinations would be best suited are needed.

Defining new custom activation functions for this specific context is a possible option.

Complexity of the neural networks We argued that learning a trace-to-variant mapping was feasible due to the number of traces w.r.t. the limited number of process variants. Generally, the challenge lies in the fact that having temporal sequences forces dependencies between elements that are usually learned separately. We suppose that deeper RNNs (*i.e.,* increasing the number of hidden layers) may have a positive impact. Adding more layers increases the complexity of the model (as well as requires more resources for training), but allows for a more accurate mapping between traces and variants. Yet, the risk of overfitting must not be neglected. In the future, we will also consider architectures such as auto-encoders to produce a compact intern representation of traces, that could be more efficient in discriminating them according to the process variants. Similarly to other application domains (*e.g.,* image or speech processing), learning more compact representations could rely on new feature descriptors instead of only considering events of a trace.

7.3 Variant-based vs. Option-based Labelling

Our results indicate that applying classification techniques on a variant-based approach (*i.e.,* identify the variants producing a specific trace) using RNNs is promising. However, it has a major drawback: being able to predict that a trace is generated by a variant requires seeing at least one (usually much more) trace(s) generated from this variant. Said differently, enumerating all the variants and executing them all at least once is required for further predictions. If in our evaluation the number of variants was limited, the combinatorial explosion problem inherent to VISs may prevent us to apply these techniques to larger configurable processes like, for instance, continuous integration workflows with hundreds of options, leading to an intractable number of possible variants.

One future possibility to address this limitation is to work on data representation. Indeed, a variant is formed by a combination of (Boolean) options, corresponding to a *configuration* of the system. If we cannot enumerate variants, enumerating options is possible. In this case, we need a new representation which can depict the three states of each option: activated, deactivated or undetermined (*i.e.,* the presence of the option is not relevant for the current context). The neural network will learn a partial configuration allowing for a more fine-grained mapping. This would be useful to locate precisely a combination of options yielding a given anomalous event trace. One can use such learned models in fault localisation and repair techniques [41]. As all labelling approaches, this new option-based approach is a costly task, but unlike a variant-based approach, it is feasible. For example, in Claroline [30, 32] we have more than 5 million variants but only 44 different features. However, this new approach comes with its own challenges. Predicting the wrong features can potentially lead to a violation of the FM's constraints, creating an invalid configuration.

7.4 Data availability

As for any DL technique, the issue of data availability is also present in this work. We managed to train our models with few execution traces (*i.e.,* thousands) compared to the potentially infinite number of traces that the considered systems can produce. However, VaryMinions remains a supervised machine-learning technique and requires a set of execution logs, labelled with the variants of the system that have produced them.

To reduce the labelling effort, the recent field of *semi-supervised learning* [20] techniques seems interesting. Semi-supervised learning takes place when, in the training set, some data have labels but a majority of them are unlabelled (*e.g.,* due to prohibitive cost in labelling that cannot allow labelling more than a few tens). The goal is thus to learn a model while being able to label *automatically* the unlabelled data. In this area, label propagation [68, 78] automatically assigns a new label via propagating the label of already known similar data. We envision using the same technique (or an adapted version) to reduce the labelling effort while being able to take into account more and more execution logs which may improve the prediction performances of VaryMinions models.

8 Related Work

This paper focuses on using DL techniques to reverse-engineer configurations. However, it is not the only context where DL has been used in conjunction with business processes or SPLs. This section gives an overview of existing approaches where both DL and variable systems meet.

8.1 Machine Learning for Process Monitoring and Mining

Machine learning, in particular deep learning, has been notably used in business process monitoring. For instance, ML models can use past observations to predict the next event in a process [36, 90, 118, 120, 128], the outcome of a process [15, 76, 130], the remaining time [117, 133], vulnerabilities and anomalies [13, 61, 98–100] or even performance [101]. This vast research area called *predictive business process monitoring*, attracted several literature reviews (*e.g.,* [60, 97]). ML can also be used to optimise existing processes [42] or to get a compact representation of traces [16, 17]. Recently, there has been interest in the interpretability of RNNs models, specifically in a process mining context [59].

Han *et al.* [58] use LSTM to discover automatically business processes from textual documentation. However, their work is focused on single processes and does not highlight variability.

8.2 Engineering Configurable Processes

When trying to (reverse-)engineer configurable processes or even perform maintenance and/or evolution, some of the reported techniques rely on grammar-based or evolutionary algorithms, while others are machine learning (ML) oriented. The latter mostly consider tasks like clustering traces (*e.g.,* [115]). However, few techniques allow to retrieving a complete configurable process from event logs. Some approaches use genetic algorithms [18, 77], but they are limited to a small number of variants. Another option is to use (configurable) process fragments to rebuild the configurable model [10]. Sikal *et al.* propose a pattern for variability discovery during process mining, but this approach is only methodological at this stage [114].

In our case, we focus on the classification task. Bobek *et al.* [12] offer recommendations to configure variability-aware business processes at design time with Bayesian Networks. Clustering techniques have also been used [28, 87, 125] to perform classification tasks in an unsupervised way, *i.e.,* without knowing the classes to learn. Song *et al.* use dimensionality reduction techniques to improve trace clustering [115]. In our context, we want to specify the variants (*i.e.,* the classes) to learn. Finally, Hinkka *et al.* [64] aim at categorising traces into classes, thanks to LSTMs and GRUs. However, their approach differs on several points: (i) they define artificial classes, and (ii) they focus on binary classification.

8.3 Machine Learning for Variability-Intensive Systems

While there is a growing interest to employ ML techniques for VIS engineering [43, 102], to the best of our knowledge, classification of variants from behavioural traces using ML techniques has not been studied yet. ML approaches have been used to support performance prediction (*e.g.,* [3, 11, 55, 71, 113, 124, 134]), performance optimisation (*e.g.,* [39, 88, 127, 131, 132]), to improve the search for good and acceptable configurations (*e.g.,* [95, 122, 123]) or to predict unwanted feature interactions [74, 82]. If some of these works also target classification tasks, they consider configurations as the main entry point of their approaches and do not take into account the behaviour of the studied systems. ML also supports usability prediction [129], attacks and vulnerabilities detection [1] and defect prediction [4, 116]. In particular, Strüder *et al.* demonstrated that artificial neural networks were suitable for this last task [116].

While ML can support VIS engineering, the converse, *i.e.,* applying variability-aware techniques to neural networks is also possible. For example, Ghofrani *et al.* [52, 53] proposed a new approach to reuse modules of deep neural networks without additional training. On their side, Ghamizi *et al.* developed a framework to explore variability amongst different neural network architectures and automated search-based techniques to find the optimal one for a given task [50, 51].

8.4 Variability-Intensive Systems Reverse Engineering

Over the years, several approaches were proposed to reverse engineer VISs, and SPLs in particular. These techniques operate at different levels: variability model, mapping between options and VIS artefacts, and learning VIS design models.

8.4.1 Learning Variability models

More than a decade of effort has been devoted to extracting options from VIS artefacts. Due to their popularity, most approaches target feature models [2, 81, 86, 89, 112]. Besides, Ramos-Gutiérrez *et al.* [105] use process mining to retrieve the process of configuring an SPL. VaryMinions does not necessarily need a complete feature model but rather a set of variants. They can be sampled from a feature model as we did for the Claroline system or simply known via product descriptions.

8.4.2 Learning VIS Design Models

There also exist model-based approaches to recover an architectural model of a VIS [8, 73, 83]. This can be useful when the system was not designed with the SPL paradigm in mind (but *e.g.,* by using a clone-and-own approach) and when we want to perform complex maintenance or evolution tasks. Devroey *et al.* [33] designed a technique to retrieve a behavioural model of an SPL. This technique, based on usage models inferred from logs, learns a candidate FTS which should be completed manually with annotations (*feature expressions*). This technique yielded the FTS model we used to generate Claroline datasets. On the other hand, Damasceno *et al.* [26, 27] idea is fully automated, but limited to a few variants. Their proposal consists of an adaptation of a classical learning algorithm ($L*$, by Angluin [5]) which is instrumented to merge individual models of each variant into a model of the complete SPL. Note that merging has a high complexity (*i.e.,* exponential) with respect to the number of variants to merge.

In contrast, VaryMinions does not aim to learn a behavioural model but to build a mapping between behaviour and variants. It may prove useful to automatically annotate SPL models.

8.4.3 Learning VIS Mappings

Feature location is another task in VIS reverse engineering that Cruz *et al.* [25] divided into three categories of techniques: static (based on source code), dynamic (based on execution traces), and textual (based on NLP). Some techniques mix several approaches; for instance, Michelon *et al.* [91] use a hybrid approach based both on static analysis of the source code and dynamic analysis of execution traces. However, the general idea of feature location is slightly different since these are white-box approaches whose purpose is to

map features with source code (*e.g.,* by source code annotations). Their goal is usually to help with maintenance and evolution. Moreover, classical feature location techniques (*e.g.,* [25, 91]) do not use RNNs. In our case, we are more focused on associating *behaviours* with variants or directly with features (see future work in Section 7.3). VaryMinions is thus a black-box and dynamic approach that could be used to make a first classification of variants of interest before delving into the source code or other white-box artifacts.

9 Conclusion

In this work, we evaluated the relevance of using Recurrent Neural Networks (RNNs) to address the problem of how to multi-classify behavioural traces found in logs to the variant(s) they belong. This mapping is highly relevant when debugging variability-intensive systems (VIS) as anomalous behaviour may result from the interaction of a few specific options belonging to some variants amongst a myriad. Based on the promising results we obtained for configurable business processes [46], we extended our experiments to Claroline, a configurable course management system previously re-engineered at the university of Namur. We assessed two popular RNN types – Long Short Term Memory (LSTM) and Gated Recurrent Units (GRU) – under 20 distinct parameterisations on 6 datasets (2 from configurable processes and 4 generated from Claroline models). Our results show that it is always possible to learn a mapping with an accuracy of at least 80% [47]. There is no prevalence of one particular model type (GRU or LSTM) among the best-performing models.

While we demonstrated that VaryMinions easily scales up to at least 50 variants and 5,000+ traces per variant, covering huge configuration spaces, *e.g.,* learning mapping for hundreds or thousands of configurations, may be problematic. It suggests the first item for our future work: offer an option-based encoding for the mapping problem, which would be less prone to variant explosion. We also intend to experiment with other loss functions and design new dedicated ones. Finally, new neural architectures may be considered, such as attention-based ones [126].

Acknowledgements Computational resources have been provided by the Consortium des Équipements de Calcul Intensif (CÉCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2.5020.11 and by the Walloon Region. This research is partly supported by the EOS VeriLearn project, under FRS-FNRS Grant No. O05518F-RG03.

References

1. Abdelrazek, M., Grundy, J., Ibrahim, A.: Towards self-securing software systems: Variability spectrum. In: Software Engineering for Variability Intensive Systems, pp. 119–130. Auerbach Publications (2019)
2. Acher, M., Baudry, B., Heymans, P., Cleve, A., Hainaut, J.L.: Support for reverse engineering and maintaining feature models. In: S. Gnesi, P. Collet, K. Schmid (eds.) VaMoS, p. 20. ACM (2013)

3. Alves Pereira, J., Acher, M., Martin, H., Jézéquel, J.M.: Sampling effect on performance prediction of configurable systems: A case study. In: Proceedings of the ACM/SPEC International Conference on Performance Engineering, pp. 277–288. ACM (2020). DOI https://doi.org/10.1145/3358960.3379137
4. Amand, B., Cordy, M., Heymans, P., Acher, M., Temple, P., Jézéquel, J.M.: Towards learning-aided configuration in 3d printing: Feasibility study and application to defect prediction. In: Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems, pp. 1–9. ACM (2019). DOI https://doi.org/10.1145/3302333.3302338
5. Angluin, D.: Learning regular sets from queries and counterexamples. Information and computation **75**(2), 87–106 (1987)
6. Apel, S., Batory, D.S., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines - Concepts and Implementation. Springer (2013). DOI 10.1007/978-3-642-37521-7. URL `https://doi.org/10.1007/978-3-642-37521-7`
7. Arganese, E., Fantechi, A., Gnesi, S., Semini, L.: Nuts and bolts of extracting variability models from natural language requirements documents. In: Integrating Research and Practice in Software Engineering, pp. 125–143. Springer (2020). DOI https://doi.org/10.1007/978-3-030-26574-8_10
8. Assunção, W.K., Vergilio, S.R., Lopez-Herrejon, R.E.: Automatic extraction of product line architecture and feature models from uml class diagram variants. Information and Software Technology **117**, 106198 (2020)
9. Assunção, W.K.G., Lopez-Herrejon, R.E., Linsbauer, L., Vergilio, S.R., Egyed, A.: Reengineering legacy applications into software product lines: a systematic mapping. Empirical Software Engineering **22**, 2972–3016 (2017)
10. Assy, N., Chan, N.N., Gaaloul, W.: An automated approach for assisting the design of configurable process models. IEEE transactions on services computing **8**(6), 874–888 (2015). DOI https://doi.org/10.1109/TSC.2015.2477815
11. Bacciu, D., Gnesi, S., Semini, L.: Using a machine learning approach to implement and evaluate product line features. In: M.H. ter Beek, A. Lluch-Lafuente (eds.) Proceedings 11th International Workshop on Automated Specification and Verification of Web Systems, WWV 2015, Oslo, Norway, 23rd June 2015, *EPTCS*, vol. 188, pp. 75–83. EPTCS (2015). DOI 10.4204/EPTCS.188.8. URL `https://doi.org/10.4204/EPTCS.188.8`
12. Bobek, S., Baran, M., Kluza, K., Nalepa, G.J.: Application of bayesian networks to recommendations in business process modeling. In: AIBP at AI* IA, pp. 41–50. Springer (2013)
13. Borkowski, M., Fdhila, W., Nardelli, M., Rinderle-Ma, S., Schulte, S.: Event-based failure prediction in distributed business processes. Information Systems **81**, 220–235 (2019). DOI https://doi.org/10.1016/j.is.2017.12.005. URL `https://www.sciencedirect.com/science/article/pii/S0306437917300030`
14. Boussaa, M., Barais, O., Baudry, B., Sunyé, G.: Automatic non-functional testing of code generators families. ACM SIGPLAN Notices **52**(3), 202–212 (2016)
15. Bozorgi, Z.D., Teinemaa, I., Dumas, M., La Rosa, M., Polyvyanyy, A.: Process mining meets causal machine learning: Discovering causal rules from event logs. In: 2020 2nd International Conference on Process Mining (ICPM), pp. 129–136. IEEE (2020)
16. Bui, H.N., Vu, T.S., Nguyen, H.H., Nguyen, T.T., Ha, Q.T.: Exploiting cbow and lstm models to generate trace representation for process mining. In: Asian Conference on Intelligent Information and Database Systems, pp. 35–46. Springer (2020)
17. Bui, H.N., Vu, T.S., Nguyen, T.T., Nguyen, T.C., Ha, Q.T.: A compact trace representation using deep neural networks for process mining. In: 2019 11th International Conference on Knowledge and Systems Engineering (KSE), pp. 1–5. IEEE (2019)
18. Buijs, J.C., van Dongen, B.F., van der Aalst, W.M.: Mining configurable process models from collections of event logs. In: Business process management, pp. 33–48. Springer (2013). DOI https://doi.org/10.1007/978-3-642-40176-3_5
19. Cândido, J., Aniche, M., van Deursen, A.: Log-based software monitoring: a systematic mapping study. PeerJ Computer Science **7**, e489 (2021)
20. Chapelle, O., Schölkopf, B., Zien, A.: Semi-Supervised Learning. MIT press Cambridge (2006)

21. Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder–decoder for statistical machine translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1724–1734. Association for Computational Linguistics (2014). DOI https://doi.org/10.3115/v1/D14-1179
22. Chollet, F., et al.: Keras. `https://keras.io` (2015)
23. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. In: NIPS 2014 Workshop on Deep Learning, December 2014 (2014)
24. Classen, A., Cordy, M., Schobbens, P., Heymans, P., Legay, A., Raskin, J.: Featured transition systems: Foundations for verifying variability-intensive systems and their application to LTL model checking. IEEE Trans. Software Eng. **39**(8), 1069–1089 (2013). DOI 10.1109/TSE.2012.86. URL `https://doi.org/10.1109/TSE.2012.86`
25. Cruz, D., Figueiredo, E., Martinez, J.: A literature review and comparison of three feature location techniques using argouml-spl. In: Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems, pp. 1–10 (2019)
26. Damasceno, C.D.N., Mousavi, M.R., da Silva Simao, A.: Learning to reuse: Adaptive model learning for evolving systems. In: International Conference on Integrated Formal Methods, pp. 138–156. Springer (2019)
27. Damasceno, C.D.N., Mousavi, M.R., da Silva Simao, A.: Learning by sampling: learning behavioral family models from software product lines. Empirical Software Engineering **26**(1), 1–46 (2021)
28. De Weerdt, J., vanden Broucke, S.K., Vanthienen, J., Baesens, B.: Leveraging process discovery with trace clustering and text mining for intelligent analysis of incident management processes. In: IEEE Congress on Evolutionary Computation, pp. 1–8. IEEE (2012). DOI https://doi.org/10.1109/CEC.2012.6256459
29. Developers, T.: Tensorflow (2021). DOI 10.5281/zenodo.4758419
30. Devroey, X.: VIBeS Case Studies: Featured Transition Systems and Feature Models (2020). DOI 10.5281/zenodo.4105900. URL `https://doi.org/10.5281/zenodo.4105900`
31. Devroey, X.: VIBeS: Variability Intensive system Behavioral teSting framework (2022). URL `https://github.com/xdevroey/vibes`
32. Devroey, X., Perrouin, G., Cordy, M., Samih, H., Legay, A., Schobbens, P., Heymans, P.: Statistical prioritization for software product line testing: an experience report. Softw. Syst. Model. **16**(1), 153–171 (2017). DOI 10.1007/s10270-015-0479-8. URL `https://doi.org/10.1007/s10270-015-0479-8`
33. Devroey, X., Perrouin, G., Cordy, M., Schobbens, P., Legay, A., Heymans, P.: Towards statistical prioritization for software product lines testing. In: P. Collet, A. Wasowski, T. Weyer (eds.) The Eighth International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '14, Sophia Antipolis, France, January 22-24, 2014, pp. 10:1–10:7. ACM (2014). DOI 10.1145/2556624.2556635. URL `https://doi.org/10.1145/2556624.2556635`
34. Devroey, X., Perrouin, G., Legay, A., Schobbens, P., Heymans, P.: Covering SPL behaviour with sampled configurations: An initial assessment. In: K. Schmid, Ø. Haugen, J. Müller (eds.) Proceedings of the Ninth International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '15, Hildesheim, Germany, January 21-23, 2015, p. 59. ACM (2015). DOI 10.1145/2701319.2701325. URL `https://doi.org/10.1145/2701319.2701325`
35. Devroey, X., Perrouin, G., Legay, A., Schobbens, P.Y., Heymans, P.: Search-based similarity-driven behavioural spl testing. In: Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems, pp. 89–96 (2016)
36. Di Mauro, N., Appice, A., Basile, T.M.: Activity prediction of business process instances with inception cnn models. In: International conference of the italian association for artificial intelligence, pp. 348–361. Springer (2019)
37. van Dongen, B.: Bpi challenge 2020 (2020). DOI 10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51. URL `https://data.4tu.nl/collections/BPI_Challenge_2020/5065541/1`

38. van Dongen, B.B.: Bpi challenge 2015 (2015). DOI 10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1. URL https://data.4tu.nl/collections/BPI_Challenge_2015/5065424/1
39. Dorn, J., Apel, S., Siegmund, N.: Generating attributed variability models for transfer learning. In: Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems, VAMOS '20. ACM (2020). DOI 10.1145/3377024.3377040. URL https://doi.org/10.1145/3377024.3377040
40. Evermann, J., Rehse, J.R., Fettke, P.: Predicting process behaviour using deep learning. Decision Support Systems **100**, 129–140 (2017). DOI https://doi.org/10.1016/j.dss.2017.04.003
41. Fahland, D., van der Aalst, W.M.: Model repair — aligning process models to reality. Information Systems **47**, 220–243 (2015). DOI https://doi.org/10.1016/j.is.2013.12.007. URL https://www.sciencedirect.com/science/article/pii/S0306437913001725
42. Fernandes, E.C., Fitzgerald, B., Brown, L., Borsato, M.: Machine learning and process mining applied to process optimization: Bibliometric and systemic analysis. Procedia Manufacturing **38**, 84–91 (2019). DOI https://doi.org/10.1016/j.promfg.2020.01.012. URL https://www.sciencedirect.com/science/article/pii/S2351978920300123. 29th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM 2019), June 24-28, 2019, Limerick, Ireland, Beyond Industry 4.0: Industrial Advances, Engineering Education and Intelligent Manufacturing
43. Ferreira, F., Silva, L.L., Valente, M.T.: Software engineering meets deep learning: a mapping study. In: Proceedings of the 36th Annual ACM Symposium on Applied Computing, pp. 1542–1549 (2021)
44. Fortz, S.: Lifts: Learning featured transition systems. In: Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume B, SPLC '21, p. 1–6. Association for Computing Machinery, New York, NY, USA (2021). DOI 10.1145/3461002.3473066. URL https://doi.org/10.1145/3461002.3473066
45. Fortz, S.: Variability-aware Behavioural Learning. In: Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume B, Tokyo, Japan, SPLC '23. Association for Computing Machinery, New York, NY, USA (2023). DOI 10.1145/3579028.3609007. URL https://doi.org/10.1145/3579028.3609007
46. Fortz, S., Temple, P., Devroey, X., Heymans, P., Perrouin, G.: Varyminions: leveraging rnns to identify variants in event logs. In: A. Ampatzoglou, D. Feitosa, G. Catolino, V. Lenarduzzi (eds.) Proceedings of the 5th International Workshop on Machine Learning Techniques for Software Quality Evolution, Athens, Greece, 23 August 2021, pp. 13–18. ACM (2021). DOI 10.1145/3472674.3473980. URL https://doi.org/10.1145/3472674.3473980
47. Fortz, S., Temple, P., Devroey, X., Heymans, P., Perrouin, G.: Varyminions (2022). DOI 10.5281/zenodo.7492126. URL https://zenodo.org/record/7492126. Sophie Fortz is supported by the FNRS via a FRIA grant. Gilles Perrouin is an FNRS Research Associate.
48. García, S., Molina, D., Lozano, M., Herrera, F.: A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the cec'2005 special session on real parameter optimization. Journal of Heuristics **15**(6), 617–644 (2009)
49. Geman, S., Bienenstock, E., Doursat, R.: Neural networks and the bias/variance dilemma. Neural computation **4**(1), 1–58 (1992)
50. Ghamizi, S., Cordy, M., Papadakis, M., Traon, Y.L.: Automated search for configurations of convolutional neural network architectures. In: Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A, pp. 119–130. ACM (2019). DOI https://doi.org/10.1145/3336294.3336306
51. Ghamizi, S., Cordy, M., Papadakis, M., Traon, Y.L.: Featurenet: diversity-driven generation of deep learning models. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings, pp. 41–44. ACM (2020). DOI https://doi.org/10.1145/3377812.3382153
52. Ghofrani, J., Kozegar, E., Bozorgmehr, A., Soorati, M.D.: Reusability in artificial neural networks: an empirical study. In: Proceedings of the 23rd International Systems and Software Product Line Conference-Volume B, pp. 122–129. ACM (2019). DOI https://doi.org/10.1145/3307630.3342419

53. Ghofrani, J., Kozegar, E., Fehlhaber, A.L., Soorati, M.D.: Applying product line engineering concepts to deep neural networks. In: Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A, pp. 72–77. ACM (2019). DOI https://doi.org/10.1145/3336294.3336321
54. Golia, P., Soos, M., Chakraborty, S., Meel, K.S.: Designing samplers is easy: The boon of testers. In: Formal Methods in Computer Aided Design, FMCAD 2021, New Haven, CT, USA, October 19-22, 2021, pp. 222–230. IEEE (2021). DOI 10.34727/2021/isbn.978-3-85448-046-4_31. URL https://doi.org/10.34727/2021/isbn.978-3-85448-046-4_31
55. Ha, H., Zhang, H.: Deepperf: performance prediction for configurable software with deep sparse neural network. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 1095–1106. IEEE (2019)
56. Hafemann Fragal, V., Simao, A., Mousavi, M.R.: Validated test models for software product lines: Featured finite state machines. In: O. Kouchnarenko, R. Khosravi (eds.) Formal Aspects of Component Software, pp. 210–227. Springer International Publishing, Cham (2017)
57. Halin, A., Nuttinck, A., Acher, M., Devroey, X., Perrouin, G., Baudry, B.: Test them all, is it worth it? assessing configuration sampling on the jhipster web development stack. Empir. Softw. Eng. **24**(2), 674–717 (2019). DOI 10.1007/s10664-018-9635-4. URL https://doi.org/10.1007/s10664-018-9635-4
58. Han, X., Hu, L., Mei, L., Dang, Y., Agarwal, S., Zhou, X., Hu, P.: A-bps: Automatic business process discovery service using ordered neurons lstm. In: 2020 IEEE International Conference on Web Services (ICWS), pp. 428–432. IEEE (2020)
59. Hanga, K.M., Kovalchuk, Y., Gaber, M.M.: A graph-based approach to interpreting recurrent neural networks in process mining. IEEE Access **8**, 172923–172938 (2020)
60. Harane, N., Rathi, S.: Comprehensive survey on deep learning approaches in predictive business process monitoring. Modern Approaches in Machine Learning and Cognitive Science: A Walkthrough pp. 115–128 (2020)
61. Hariyanti, E., Djunaidy, A., Siahaan, D.: Information security vulnerability prediction based on business process model using machine learning approach. Computers & Security **110**, 102422 (2021). DOI https://doi.org/10.1016/j.cose.2021.102422. URL https://www.sciencedirect.com/science/article/pii/S0167404821002467
62. Henard, C., Papadakis, M., Perrouin, G., Klein, J., Heymans, P., Traon, Y.L.: Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. IEEE Trans. Software Eng. **40**(7), 650–670 (2014). DOI 10.1109/TSE.2014.2327020. URL https://doi.org/10.1109/TSE.2014.2327020
63. Henard, C., Papadakis, M., Perrouin, G., Klein, J., Traon, Y.L.: PLEDGE: a product line editor and test generation tool. In: 17th International Software Product Line Conference co-located workshops, SPLC 2013 workshops, Tokyo, Japan - August 26 - 30, 2013, pp. 126–129. ACM (2013). DOI 10.1145/2499777.2499778. URL https://doi.org/10.1145/2499777.2499778
64. Hinkka, M., Lehto, T., Heljanko, K., Jung, A.: Classifying process instances using recurrent neural networks. In: International Conference on Business Process Management, pp. 313–324. Springer (2018). DOI https://doi.org/10.1007/978-3-030-11641-5_25
65. Hochreiter, S.: The vanishing gradient problem during learning recurrent neural nets and problem solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **6**(02), 107–116 (1998). DOI https://doi.org/10.1142/S0218488598000094
66. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997). DOI https://doi.org/10.1162/neco.1997.9.8.1735
67. Hui, L., Belkin, M.: Evaluation of neural architectures trained with square loss vs cross-entropy in classification tasks. In: The Ninth International Conference on Learning Representations (ICLR 2021) (2021)
68. Iscen, A., Tolias, G., Avrithis, Y., Chum, O.: Label propagation for deep semi-supervised learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 5070–5079 (2019)
69. Jaccard, P.: Étude comparative de la distribution florale dans une portion des alpes et des jura. Bulletin de la Société Vaudoise des Sciences Naturelles **37**, 547–579 (1901)

70. Japkowicz, N., Shah, M.: Evaluating learning algorithms: a classification perspective. Cambridge University Press (2011)
71. Kaltenecker, C., Grebhahn, A., Siegmund, N., Apel, S.: The interplay of sampling and machine learning for software performance prediction. IEEE Software **37**(4), 58–66 (2020). DOI https://doi.org/10.1109/MS.2020.2987024
72. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Tech. rep., Carnegie-Mellon University, Software Engineering Institute (1990)
73. Kerdoudi, M.L., Ziadi, T., Tibermacine, C., Sadou, S.: Recovering software architecture product lines. In: 2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 226–235. IEEE (2019)
74. Khoshmanesh, S., Lutz, R.: Does link prediction help find feature interactions in software product lines? In: 2020 IEEE Seventh International Workshop on Artificial Intelligence for Requirements Engineering (AIRE), pp. 87–90. IEEE (2020)
75. Kowsari, K., Brown, D.E., Heidarysafa, M., Meimandi, K.J., Gerber, M.S., Barnes, L.E.: Hdltex: Hierarchical deep learning for text classification. In: 16th IEEE international conference on machine learning and applications (ICMLA), pp. 364–371. IEEE (2017). DOI https://doi.org/10.1109/ICMLA.2017.0-134
76. Kratsch, W., Manderscheid, J., Röglinger, M., Seyfried, J.: Machine learning in business process monitoring: a comparison of deep learning and classical approaches used for outcome prediction. Business & Information Systems Engineering pp. 1–16 (2020)
77. La Rosa, M., Dumas, M.: Configurable process models: how to adopt standard practices in your how way? BPTrends Newsletter (2008)
78. Lee, D.H., et al.: Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In: Workshop on challenges in representation learning, ICML, vol. 3, p. 896 (2013)
79. Levenshtein, V.I., et al.: Binary codes capable of correcting deletions, insertions, and reversals. In: Soviet physics doklady, vol. 10, pp. 707–710. Soviet Union (1966)
80. Li, Y., Schulze, S., Saake, G.: Reverse engineering variability from natural language documents: A systematic literature review. In: Proceedings of the 21st International Systems and Software Product Line Conference-Volume A, pp. 133–142. ACM (2017). DOI https://doi.org/10.1145/3106195.3106207
81. Li, Y., Schulze, S., Saake, G.: Reverse engineering variability from natural language documents: A systematic literature review. In: SPLC'17 - Volume A, SPLC '17, pp. 133–142. ACM, New York, NY, USA (2017). DOI 10.1145/3106195.3106207. URL `http://doi.acm.org/10.1145/3106195.3106207`
82. Li, Y., Schulze, S., Xu, J.: Feature terms prediction: A feasible way to indicate the notion of features in software product line. In: Proceedings of the Evaluation and Assessment in Software Engineering, EASE '20, p. 90–99. Association for Computing Machinery, New York, NY, USA (2020). DOI 10.1145/3383219.3383229. URL `https://doi.org/10.1145/3383219.3383229`
83. Lima, C., Assunção, W.K., Martinez, J., Mendonça, W., Machado, I.C., Chavez, C.F.: Product line architecture recovery with outlier filtering in software families: the apogames case study. Journal of the Brazilian Computer Society **25**(1), 1–17 (2019)
84. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. IEEE Transactions on Pattern Analysis and Machine Intelligence **42**(2), 318–327 (2020). DOI https://doi.org/10.1109/TPAMI.2018.2858826
85. Liu, P., Qiu, X., Huang, X.: Recurrent neural network for text classification with multi-task learning. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI'16, p. 2873–2879. AAAI Press (2016). DOI https://doi.org/10.5555/3060832.3061023
86. Lopez-Herrejon, R.E., Linsbauer, L., Egyed, A.: A systematic mapping study of search-based software engineering for software product lines. Information and Software Technology **61**, 33 – 51 (2015). DOI http://dx.doi.org/10.1016/j.infsof.2015.01.008. URL `//www.sciencedirect.com/science/article/pii/S0950584915000166`
87. Mans, R.S., Schonenberg, M., Song, M., van der Aalst, W.M., Bakker, P.J.: Application of process mining in healthcare–a case study in a dutch hospital. In: International joint conference on biomedical engineering systems and technologies, pp. 425–438. Springer (2008). DOI https://doi.org/10.1007/978-3-540-92219-3_32

88. Martin, H., Acher, M., Pereira, J.A., Jézéquel, J.M.: A comparison of performance specialization learning for configurable systems. In: Proceedings of the 25th ACM International Systems and Software Product Line Conference-Volume A, pp. 46–57 (2021)
89. Martinez, J., Parsai, A.: D3.1: Identification of relevant state of the art. Tech. rep., ITEA 3 ReVAMP2 Project Consortium (2018)
90. Matzner, M., Eskofier, B.: Time matters: Time-aware lstms for predictive business process monitoring. In: Process Mining Workshops: ICPM 2020 International Workshops, Padua, Italy, October 5–8, 2020, Revised Selected Papers, vol. 406, p. 112. Springer Nature (2021)
91. Michelon, G.K., Linsbauer, L., Assunção, W.K., Fischer, S., Egyed, A.: A hybrid feature location technique for re-engineeringsingle systems into software product lines. In: 15th International Working Conference on Variability Modelling of Software-Intensive Systems, pp. 1–9 (2021)
92. Michelon, G.K., Martinez, J., Sotto-Mayor, B., Arrieta, A., Assunção, W.K., Abreu, R., Egyed, A.: Spectrum-based feature localization for families of systems. Journal of Systems and Software **195**, 111532 (2023). DOI https://doi.org/10.1016/j.jss.2022.111532. URL `https://www.sciencedirect.com/science/article/pii/S0164121222002084`
93. Mortara, J., Collet, P.: Capturing the Diversity of Analyses on the Linux Kernel Variability, p. 160–171. Association for Computing Machinery, New York, NY, USA (2021). URL `https://doi.org/10.1145/3461001.3471151`
94. Nagarajah, T., Poravi, G.: A review on automated machine learning (automl) systems. In: 2019 IEEE 5th International Conference for Convergence in Technology (I2CT), pp. 1–6. IEEE (2019)
95. Nair, V., Menzies, T., Siegmund, N., Apel, S.: Using bad learners to find good configurations. In: E. Bodden, W. Schäfer, A. van Deursen, A. Zisman (eds.) Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017, pp. 257–267. ACM (2017). DOI 10.1145/3106237.3106238. URL `https://doi.org/10.1145/3106237.3106238`
96. Nemenyi, P.B.: Distribution-free multiple comparisons. Princeton University (1963)
97. Neu, D.A., Lahann, J., Fettke, P.: A systematic literature review on state-of-the-art deep learning methods for process prediction. Artificial Intelligence Review pp. 1–27 (2021)
98. Nguyen, H.T.C., Lee, S., Kim, J., Ko, J., Comuzzi, M.: Autoencoders for improving quality of process event logs. Expert Systems with Applications **131**, 132–147 (2019). DOI https://doi.org/10.1016/j.eswa.2019.04.052
99. Nolle, T., Seeliger, A., Mühlhäuser, M.: Binet: multivariate business process anomaly detection using deep learning. In: International Conference on Business Process Management, pp. 271–287. Springer (2018). DOI https://doi.org/10.1007/978-3-319-98648-7_16
100. Nolle, T., Seeliger, A., Thoma, N., Mühlhäuser, M.: Deepalign: Alignment-based process anomaly correction using recurrent neural networks. In: International Conference on Advanced Information Systems Engineering, pp. 319–333. Springer (2020). DOI https://doi.org/10.1007/978-3-030-49435-3_20
101. Park, G., Song, M.: Predicting performances in business processes using deep neural networks. Decision Support Systems **129**, 113191 (2020)
102. Pereira, J.A., Martin, H., Temple, P., Acher, M.: Machine learning and configurable systems: A gentle introduction. In: Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A, SPLC '20. ACM (2020). DOI https://doi.org/10.1145/3382025.3414976. URL `https://doi.org/10.1145/3382025.3414976`
103. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag (2005)
104. Pohl, K., Böckle, G., Van Der Linden, F.: Software product line engineering: foundations, principles, and techniques. Springer (2005)
105. Ramos-Gutiérrez, B., Varela-Vaca, Á.J., Galindo, J.A., Gómez-López, M.T., Benavides, D.: Discovering configuration workflows from existing logs using process mining. Empirical Software Engineering **26**(1), 1–41 (2021)
106. Rosa, M.L., Aalst, W.M.V.D., Dumas, M., Milani, F.P.: Business process variability modeling: A survey. ACM Computing Surveys **50**(1), 1–45 (2017). DOI https://doi.org/10.1145/3041957

107. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. nature **323**(6088), 533–536 (1986)
108. Sánchez, A.B., Segura, S., Parejo, J.A., Cortés, A.R.: Variability testing in the wild: the drupal case study. Softw. Syst. Model. **16**(1), 173–194 (2017). DOI 10.1007/s10270-015-0459-z. URL https://doi.org/10.1007/s10270-015-0459-z
109. Schobbens, P., Heymans, P., Trigaux, J., Bontemps, Y.: Generic semantics of feature diagrams. Comput. Networks **51**(2), 456–479 (2007). DOI 10.1016/j.comnet.2006.08.008. URL https://doi.org/10.1016/j.comnet.2006.08.008
110. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. IEEE transactions on Signal Processing **45**(11), 2673–2681 (1997). DOI https://doi.org/10.1109/78.650093
111. She, S., Lotufo, R., Berger, T., Wasowski, A., Czarnecki, K.: The variability model of the linux kernel. VaMoS **10**(10), 45–51 (2010)
112. She, S., Lotufo, R., Berger, T., Wasowski, A., Czarnecki, K.: Reverse engineering feature models. In: Proceedings of the 33rd International Conference on Software Engineering, pp. 461–470. ACM (2011)
113. Shu, Y., Sui, Y., Zhang, H., Xu, G.: Perf-al: Performance prediction for configurable software through adversarial learning. In: Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–11 (2020)
114. Sikal, R., Sbai, H., Kjiri, L.: Configurable process mining: variability discovery approach. In: IEEE 5th International Congress on Information Science and Technology (CiSt), pp. 137–142. IEEE (2018). DOI https://doi.org/10.1109/CIST.2018.8596526
115. Song, M., Yang, H., Siadat, S.H., Pechenizkiy, M.: A comparative study of dimensionality reduction techniques to enhance trace clustering performances. Expert Systems with Applications **40**(9), 3722 – 3737 (2013). DOI https://doi.org/10.1016/j.eswa.2012.12.078. URL http://www.sciencedirect.com/science/article/pii/S095741741201319X
116. Strüder, S., Mukelabai, M., Strüber, D., Berger, T.: Feature-oriented defect prediction. In: Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A, pp. 1–12. ACM (2020). DOI https://doi.org/10.1145/3382025.3414960
117. Sun, X., Hou, W., Ying, Y., Yu, D.: Remaining time prediction of business processes based on multilayer machine learning. In: 2020 IEEE International Conference on Web Services (ICWS), pp. 554–558. IEEE (2020)
118. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with lstm neural networks. In: Advanced Information Systems Engineering, pp. 477–492. Springer (2017). DOI https://doi.org/10.1007/978-3-319-59536-8_30
119. Taymouri, F., Rosa, M.L., Dumas, M., Maggi, F.M.: Business process variant analysis: Survey and classification. Knowledge-Based Systems **211**, 106557 (2021). DOI https://doi.org/10.1016/j.knosys.2020.106557. URL https://www.sciencedirect.com/science/article/pii/S0950705120306869
120. Tello-Leal, E., Roa, J., Rubiolo, M., Ramirez-Alcocer, U.M.: Predicting activities in business processes with lstm recurrent neural networks. In: 2018 ITU Kaleidoscope: Machine Learning for a 5G Future (ITU K), pp. 1–7. IEEE (2018)
121. Temple, P., Acher, M., Jézéquel, J.M.: Empirical assessment of multimorphic testing. IEEE Transactions on Software Engineering **47**(7), 1511–1527 (2021). DOI 10.1109/TSE.2019.2926971
122. Temple, P., Galindo, J.A., Acher, M., Jézéquel, J.: Using machine learning to infer constraints for product lines. In: H. Mei (ed.) Proceedings of the 20th International Systems and Software Product Line Conference, SPLC 2016, Beijing, China, September 16-23, 2016, pp. 209–218. ACM (2016). DOI 10.1145/2934466.2934472. URL https://doi.org/10.1145/2934466.2934472
123. Temple, P., Perrouin, G., Acher, M., Biggio, B., Jézéquel, J.M., Roli, F.: Empirical assessment of generating adversarial configurations for software product lines. Empirical Software Engineering **26**(1), 1–49 (2021)
124. Valov, P., Guo, J., Czarnecki, K.: Empirical comparison of regression methods for variability-aware performance prediction. In: Proceedings of the 19th International Conference on Software Product Line, pp. 186–190 (2015)
125. Varela-Vaca, Á.J., Galindo, J.A., Ramos-Gutiérrez, B., Gómez-López, M.T., Benavides, D.: Process mining to unleash variability management: discovering configuration workflows using logs. In: Proceedings of the 23rd International Systems and

Software Product Line Conference-Volume A, pp. 265–276. ACM (2019). DOI https://doi.org/10.1145/3336294.3336303
126. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems, pp. 5998–6008 (2017)
127. Velez, M., Jamshidi, P., Siegmund, N., Apel, S., Kästner, C.: White-box analysis over machine learning: Modeling performance of configurable systems. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 1072–1084. IEEE (2021)
128. Venugopal, I., Töllich, J., Fairbank, M., Scherp, A.: A comparison of deep-learning methods for analysing and predicting business processes. arXiv preprint arXiv:2102.07838 (2021)
129. Vyas, G., Vyas, S., Paul, P.K., Sharma, A., Bhardwaj, C.: Prediction algorithms and consecutive estimation of software product line feature model usability. In: 2019 Amity International Conference on Artificial Intelligence (AICAI), pp. 774–777. IEEE (2019)
130. Wang, J., Yu, D., Liu, C., Sun, X.: Outcome-oriented predictive process monitoring with attention-based bidirectional lstm neural networks. In: 2019 IEEE International Conference on Web Services (ICWS), pp. 360–367. IEEE (2019)
131. Weber, M., Apel, S., Siegmund, N.: White-box performance-influence models: A profiling and learning approach. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 1059–1071. IEEE (2021)
132. Weckesser, M., Kluge, R., Pfannemüller, M., Matthé, M., Schürr, A., Becker, C.: Optimal reconfiguration of dynamic software product lines based on performance-influence models. In: Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1, pp. 98–109 (2018)
133. Welsing, M., Maetschke, J., Thomas, K., Gützlaff, A., Schuh, G., Meusert, S.: Combining process mining and machine learning for lead time prediction in high variance processes. In: B.A. Behrens, A. Brosius, W. Hintze, S. Ihlenfeldt, J.P. Wulfsberg (eds.) Production at the leading edge of technology, pp. 528–537. Springer Berlin Heidelberg, Berlin, Heidelberg (2021)
134. Zhang, Y., Guo, J., Blais, E., Czarnecki, K.: Performance prediction of configurable software systems by fourier learning (t). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 365–373. IEEE (2015)

A Appendix 1

This appendix contains 6 tables (one per datasets) representing the average and standard deviation for four metrics computed on 10 iterations. Accuracy, precision, recall and F1-score were computed based on definitions provided in Section 5.5.

The first three columns of the tables show the hyperparameters values for each of the RNNs' parameterisations. For conciseness, we do not report in these tables hyperparameters that were fixed to a single value, such as the batch size or the number of epochs. Indeed, we discussed them in Section 5. The other columns reports the average and standard deviation of accuracy, precision, recall and F1-score.

Table 4: Results for dataset BPIC15: Averaged and standard deviations of different metrics over 10 runs. Each line corresponds to a parameterisation of a RNN.

Dataset	**Model**	**Loss**	**Activation**	**Accuracy**		**Precision**		**Recall**		**F1Score**	
				Avg	**Sd**	**Avg**	**Sd**	**Avg**	**Sd**	**Avg**	**Sd**
BPIC15	LSTM	mse	tanh	0.8848	0.0074	0.8854	0.0074	0.8848	0.0074	0.8845	0.0078
BPIC15	LSTM	bin_ce	sigmoid	0.8806	0.0062	0.8818	0.006	0.8806	0.0062	0.8804	0.0064
BPIC15	LSTM	mse	sigmoid	0.8783	0.0069	0.8796	0.0064	0.8783	0.0069	0.8782	0.0068
BPIC15	GRU	mse	tanh	0.8741	0.007	0.8747	0.0071	0.8741	0.007	0.8733	0.0075
BPIC15	GRU	bin_ce-logits	sigmoid	0.8733	0.0097	0.8738	0.0103	0.8733	0.0097	0.8728	0.0101
BPIC15	GRU	bin_ce	sigmoid	0.8677	0.0126	0.8683	0.0117	0.8677	0.0126	0.8672	0.0125
BPIC15	GRU	mse	sigmoid	0.8623	0.0077	0.863	0.0081	0.8623	0.0077	0.8619	0.0079
BPIC15	LSTM	bin_ce-logits	sigmoid	0.8571	0.0095	0.8597	0.0095	0.8571	0.0095	0.8566	0.0099
BPIC15	LSTM	manhattan	tanh	0.8257	0.0145	0.8311	0.0155	0.8257	0.0145	0.8228	0.0147
BPIC15	GRU	bin_ce-logits	tanh	0.7937	0.0596	0.7502	0.1143	0.7937	0.0596	0.7676	0.0902
BPIC15	GRU	bin_ce	tanh	0.7898	0.021	0.7889	0.0206	0.7898	0.021	0.7841	0.0224
BPIC15	GRU	manhattan	tanh	0.7858	0.0092	0.7896	0.0109	0.7858	0.0092	0.7739	0.0105
BPIC15	LSTM	jaccard	sigmoid	0.7858	0.0447	0.8072	0.0581	0.7858	0.0447	0.7541	0.0666
BPIC15	LSTM	bin_ce	tanh	0.7724	0.0778	0.7864	0.0625	0.7724	0.0778	0.7666	0.0839
BPIC15	LSTM	bin_ce-logits	tanh	0.7436	0.1165	0.7129	0.1585	0.7436	0.1165	0.6944	0.1604
BPIC15	GRU	jaccard	sigmoid	0.6962	0.0583	0.706	0.0696	0.6962	0.0583	0.6566	0.0861
BPIC15	LSTM	jaccard	tanh	0.6098	0.0438	0.588	0.0276	0.6098	0.0438	0.5416	0.0565
BPIC15	GRU	jaccard	tanh	0.5529	0.0244	0.5832	0.0511	0.5529	0.0244	0.4847	0.042
BPIC15	GRU	manhattan	sigmoid	0.2538	0.0585	0.1223	0.0912	0.2538	0.0585	0.1194	0.0627
BPIC15	LSTM	manhattan	sigmoid	0.2313	0.0552	0.0893	0.0776	0.2313	0.0552	0.0926	0.0505

Table 5: Results for dataset BPIC20: Averaged and standard deviations of different metrics over 10 runs. Each line corresponds to a parameterisation of a RNN.

Dataset	**Model**	**Loss**	**Activation**	**Accuracy**		**Precision**		**Recall**		**F1Score**	
				Avg	**Sd**	**Avg**	**Sd**	**Avg**	**Sd**	**Avg**	**Sd**
BPIC20	LSTM	mse	sigmoid	0.8744	0.0397	0.8876	0.0279	0.8744	0.0397	0.8716	0.0424
BPIC20	LSTM	mse	tanh	0.8585	0.0375	0.8764	0.0261	0.8585	0.0375	0.8539	0.0416
BPIC20	LSTM	bin_ce-logits	sigmoid	0.8541	0.0714	0.8736	0.0499	0.8541	0.0714	0.8425	0.0852
BPIC20	LSTM	bin_ce	sigmoid	0.8397	0.0642	0.8598	0.0485	0.8397	0.0642	0.8308	0.0744
BPIC20	GRU	mse	sigmoid	0.8258	0.0897	0.829	0.0997	0.8258	0.0897	0.8148	0.1086
BPIC20	GRU	mse	tanh	0.8198	0.0489	0.8508	0.0287	0.8198	0.0489	0.8094	0.0574
BPIC20	GRU	bin_ce-logits	sigmoid	0.7895	0.0617	0.8229	0.0499	0.7895	0.0617	0.7722	0.0728
BPIC20	GRU	bin_ce	sigmoid	0.7758	0.0428	0.81	0.0338	0.7758	0.0428	0.758	0.054
BPIC20	LSTM	bin_ce-logits	tanh	0.7569	0.072	0.7797	0.0806	0.7569	0.072	0.7284	0.0924
BPIC20	GRU	bin_ce-logits	tanh	0.7303	0.1003	0.7828	0.0714	0.7303	0.1003	0.6852	0.1422
BPIC20	LSTM	bin_ce	tanh	0.6469	0.126	0.6276	0.2199	0.6469	0.126	0.5695	0.1876
BPIC20	GRU	bin_ce	tanh	0.5918	0.041	0.5966	0.205	0.5918	0.041	0.478	0.0642
BPIC20	LSTM	manhattan	tanh	0.5878	0.0332	0.5173	0.1275	0.5878	0.0332	0.4707	0.0459
BPIC20	GRU	manhattan	tanh	0.5826	0.0458	0.6858	0.0653	0.5826	0.0458	0.4506	0.0629
BPIC20	GRU	jaccard	sigmoid	0.5449	0.0139	0.3075	0.0176	0.5449	0.0139	0.3898	0.0165
BPIC20	LSTM	jaccard	tanh	0.4661	0.0236	0.2321	0.0435	0.4661	0.0236	0.2995	0.0278
BPIC20	LSTM	manhattan	sigmoid	0.4629	0.0115	0.2144	0.0107	0.4629	0.0115	0.293	0.0123
BPIC20	GRU	jaccard	tanh	0.4602	0.0229	0.2362	0.0367	0.4602	0.0229	0.2933	0.0242
BPIC20	GRU	manhattan	sigmoid	0.4584	0.0217	0.2105	0.02	0.4584	0.0217	0.2884	0.023
BPIC20	LSTM	jaccard	sigmoid	0.4576	0.0144	0.2096	0.0132	0.4576	0.0144	0.2875	0.0152

Table 6: Results for dataset Claroline Dissimilar 10: Averaged and standard deviations of different metrics over 10 runs. Each line corresponds to a parameterisation of a RNN.

Dataset	Model	Loss	Activation	Accuracy		Precision		Recall		F1Score	
				Avg	Sd	Avg	Sd	Avg	Sd	Avg	Sd
claroline-dis_10	GRU	bin_ce	sigmoid	0.9968	0.0006	0.9968	0.0006	0.9968	0.0006	0.9968	0.0006
claroline-dis_10	GRU	bin_ce-logits	sigmoid	0.9964	0.0003	0.9965	0.0003	0.9964	0.0003	0.9964	0.0003
claroline-dis_10	GRU	mse	sigmoid	0.9064	0.1868	0.8842	0.2111	0.9064	0.1868	0.8905	0.2059
claroline-dis_10	LSTM	mse	tanh	0.8536	0.2532	0.8181	0.3077	0.8536	0.2532	0.8245	0.2977
claroline-dis_10	LSTM	mse	sigmoid	0.7458	0.2554	0.6726	0.3102	0.7458	0.2554	0.6913	0.2986
claroline-dis_10	LSTM	bin_ce-logits	sigmoid	0.6885	0.217	0.6007	0.2605	0.6885	0.217	0.6147	0.2531
claroline-dis_10	LSTM	bin_ce	sigmoid	0.6871	0.3138	0.6213	0.3627	0.6871	0.3138	0.6258	0.362
claroline-dis_10	LSTM	bin_ce	tanh	0.3592	0.2734	0.2388	0.27	0.3592	0.2734	0.2636	0.2772
claroline-dis_10	LSTM	manhattan	tanh	0.2687	0.0785	0.1686	0.0583	0.2687	0.0785	0.1767	0.0732
claroline-dis_10	GRU	bin_ce	tanh	0.2599	0.3392	0.1774	0.3597	0.2599	0.3392	0.1875	0.3619
claroline-dis_10	GRU	jaccard	tanh	0.1998	0.1043	0.0595	0.0644	0.1998	0.1043	0.0843	0.079
claroline-dis_10	LSTM	jaccard	tanh	0.1796	0.0786	0.0552	0.0564	0.1796	0.0786	0.0728	0.0643
claroline-dis_10	GRU	manhattan	sigmoid	0.1012	0.0015	0.0103	0.0003	0.1012	0.0015	0.0186	0.0005
claroline-dis_10	GRU	bin_ce-logits	tanh	0.1006	0.0012	0.0101	0.0002	0.1006	0.0012	0.0184	0.0004
claroline-dis_10	GRU	mse	tanh	0.0998	0.0017	0.01	0.0003	0.0998	0.0017	0.0181	0.0006
claroline-dis_10	LSTM	bin_ce-logits	tanh	0.0997	0.0019	0.0116	0.005	0.0997	0.0019	0.0181	0.0007
claroline-dis_10	LSTM	manhattan	sigmoid	0.0996	0.0016	0.0099	0.0003	0.0996	0.0016	0.018	0.0006
claroline-dis_10	LSTM	jaccard	sigmoid	0.0995	0.0016	0.0099	0.0003	0.0995	0.0016	0.018	0.0006
claroline-dis_10	GRU	manhattan	tanh	0.0991	0.0022	0.0098	0.0004	0.0991	0.0022	0.0179	0.0007
claroline-dis_10	GRU	jaccard	sigmoid	0.0989	0.0022	0.0098	0.0004	0.0989	0.0022	0.0178	0.0008

Table 7: Results for dataset Claroline Random 10: Averaged and standard deviations of different metrics over 10 runs. Each line corresponds to a parameterisation of a RNN.

Dataset	**Model**	**Loss**	**Activation**	**Accuracy**		**Precision**		**Recall**		**F1Score**	
				Avg	**Sd**	**Avg**	**Sd**	**Avg**	**Sd**	**Avg**	**Sd**
claroline-rand_10	GRU	bin_ce	sigmoid	0.9861	0.0321	0.9811	0.0482	0.9861	0.0321	0.9828	0.0428
claroline-rand_10	GRU	bin_ce-logits	sigmoid	0.9664	0.0947	0.9548	0.1317	0.9664	0.0947	0.9581	0.1211
claroline-rand_10	GRU	mse	sigmoid	0.9154	0.1063	0.8869	0.1355	0.9154	0.1063	0.8945	0.1292
claroline-rand_10	LSTM	mse	tanh	0.8886	0.1639	0.859	0.2065	0.8886	0.1639	0.8619	0.1999
claroline-rand_10	LSTM	bin_ce	sigmoid	0.7678	0.2389	0.7068	0.2775	0.7678	0.2389	0.7211	0.2713
claroline-rand_10	LSTM	mse	sigmoid	0.7352	0.2056	0.6751	0.2397	0.7352	0.2056	0.6863	0.2358
claroline-rand_10	LSTM	bin_ce-logits	sigmoid	0.7183	0.1685	0.6454	0.2028	0.7183	0.1685	0.6573	0.2015
claroline-rand_10	LSTM	bin_ce	tanh	0.4748	0.1905	0.3559	0.2044	0.4748	0.1905	0.382	0.2052
claroline-rand_10	GRU	jaccard	tanh	0.2099	0.2221	0.1228	0.2179	0.2099	0.2221	0.1342	0.2249
claroline-rand_10	LSTM	manhattan	tanh	0.1902	0.0734	0.1045	0.0698	0.1902	0.0734	0.1076	0.0661
claroline-rand_10	GRU	bin_ce	tanh	0.1793	0.2517	0.0928	0.2621	0.1793	0.2517	0.1023	0.2661
claroline-rand_10	LSTM	jaccard	tanh	0.1385	0.0696	0.0271	0.0311	0.1385	0.0696	0.0426	0.0441
claroline-rand_10	LSTM	manhattan	sigmoid	0.1012	0.0017	0.0102	0.0003	0.1012	0.0017	0.0186	0.0006
claroline-rand_10	GRU	manhattan	tanh	0.1007	0.0015	0.0101	0.0003	0.1007	0.0015	0.0184	0.0005
claroline-rand_10	GRU	bin_ce-logits	tanh	0.1007	0.0013	0.0101	0.0003	0.1007	0.0013	0.0184	0.0005
claroline-rand_10	LSTM	bin_ce-logits	tanh	0.1004	0.0019	0.0101	0.0004	0.1004	0.0019	0.0183	0.0007
claroline-rand_10	GRU	mse	tanh	0.1001	0.0018	0.01	0.0004	0.1001	0.0018	0.0182	0.0006
claroline-rand_10	GRU	manhattan	sigmoid	0.0991	0.0018	0.0098	0.0004	0.0991	0.0018	0.0179	0.0006
claroline-rand_10	LSTM	jaccard	sigmoid	0.0989	0.0016	0.0098	0.0003	0.0989	0.0016	0.0178	0.0005
claroline-rand_10	GRU	jaccard	sigmoid	0.0983	0.0014	0.0097	0.0003	0.0983	0.0014	0.0176	0.0005

Table 8: Results for dataset Claroline Dissimilar 50: Averaged and standard deviations of different metrics over 10 runs. Each line corresponds to a parameterisation of a RNN.

Dataset	Model	Loss	Activation	Accuracy		Precision		Recall		F1Score	
				Avg	Sd	Avg	Sd	Avg	Sd	Avg	Sd
claroline-dis_50	LSTM	bin_ce	sigmoid	0.8001	0.1387	0.7631	0.1562	0.8001	0.1387	0.7603	0.1532
claroline-dis_50	LSTM	bin_ce-logits	sigmoid	0.7833	0.2161	0.7367	0.2528	0.7833	0.2161	0.7403	0.25
claroline-dis_50	GRU	bin_ce-logits	sigmoid	0.7225	0.381	0.6943	0.3834	0.7225	0.381	0.7002	0.3848
claroline-dis_50	LSTM	mse	tanh	0.6211	0.0379	0.5221	0.045	0.6211	0.0379	0.5373	0.0374
claroline-dis_50	GRU	bin_ce	sigmoid	0.5256	0.4459	0.4901	0.4408	0.5256	0.4459	0.4975	0.4442
claroline-dis_50	LSTM	mse	sigmoid	0.2437	0.25	0.197	0.223	0.2437	0.25	0.1983	0.2226
claroline-dis_50	GRU	mse	sigmoid	0.2089	0.2153	0.1375	0.1617	0.2089	0.2153	0.1507	0.1747
claroline-dis_50	GRU	bin_ce	tanh	0.1015	0.2253	0.0761	0.2231	0.1015	0.2253	0.0781	0.2257
claroline-dis_50	GRU	jaccard	tanh	0.0778	0.0386	0.0209	0.0217	0.0778	0.0386	0.026	0.0251
claroline-dis_50	LSTM	jaccard	tanh	0.0389	0.0131	0.0019	0.0011	0.0389	0.0131	0.0036	0.002
claroline-dis_50	LSTM	jaccard	sigmoid	0.0273	0.01	0.0013	0.0016	0.0273	0.01	0.0024	0.0026
claroline-dis_50	GRU	manhattan	sigmoid	0.0201	0.0003	0.0004	0.0	0.0201	0.0003	0.0008	0.0
claroline-dis_50	GRU	manhattan	tanh	0.0201	0.0008	0.0004	0.0001	0.0201	0.0008	0.0009	0.0002
claroline-dis_50	LSTM	bin_ce-logits	tanh	0.0201	0.0004	0.0004	0.0	0.0201	0.0004	0.0008	0.0
claroline-dis_50	LSTM	manhattan	sigmoid	0.0201	0.0006	0.0004	0.0	0.0201	0.0006	0.0008	0.0
claroline-dis_50	GRU	mse	tanh	0.02	0.0004	0.0004	0.0	0.02	0.0004	0.0008	0.0
claroline-dis_50	GRU	bin_ce-logits	tanh	0.02	0.0004	0.0004	0.0	0.02	0.0004	0.0008	0.0
claroline-dis_50	LSTM	manhattan	tanh	0.0199	0.0004	0.0004	0.0	0.0199	0.0004	0.0008	0.0
claroline-dis_50	GRU	jaccard	sigmoid	0.0192	0.0002	0.0004	0.0	0.0192	0.0002	0.0007	0.0
claroline-dis_50	LSTM	bin_ce	tanh	0.0158	0.0078	0.001	0.0009	0.0158	0.0078	0.0014	0.0012

Table 9: Results for dataset Claroline Random 50: Averaged and standard deviations of different metrics over 10 runs. Each line corresponds to a parameterisation of a RNN.

Dataset	**Model**	**Loss**	**Activation**	**Accuracy**		**Precision**		**Recall**		**F1Score**	
				Avg	**Sd**	**Avg**	**Sd**	**Avg**	**Sd**	**Avg**	**Sd**
claroline-rand_50	GRU	bin_ce	sigmoid	0.9562	0.0785	0.9442	0.1023	0.9562	0.0785	0.9475	0.0953
claroline-rand_50	GRU	bin_ce-logits	sigmoid	0.9498	0.1013	0.9368	0.1304	0.9498	0.1013	0.939	0.1238
claroline-rand_50	LSTM	bin_ce-logits	sigmoid	0.9197	0.1032	0.9085	0.1109	0.9197	0.1032	0.9041	0.1209
claroline-rand_50	LSTM	bin_ce	sigmoid	0.8855	0.0936	0.8607	0.1175	0.8855	0.0936	0.8631	0.1106
claroline-rand_50	LSTM	mse	sigmoid	0.698	0.221	0.6235	0.248	0.698	0.221	0.6429	0.2421
claroline-rand_50	LSTM	mse	tanh	0.6136	0.0387	0.5077	0.046	0.6136	0.0387	0.5289	0.0372
claroline-rand_50	GRU	mse	sigmoid	0.5852	0.3281	0.5186	0.322	0.5852	0.3281	0.5331	0.324
claroline-rand_50	GRU	mse	tanh	0.4372	0.288	0.3482	0.2402	0.4372	0.288	0.3691	0.2544
claroline-rand_50	GRU	bin_ce	tanh	0.3667	0.1106	0.3056	0.1063	0.3667	0.1106	0.3179	0.1041
claroline-rand_50	GRU	jaccard	sigmoid	0.2623	0.2071	0.1773	0.186	0.2623	0.2071	0.1923	0.1943
claroline-rand_50	LSTM	jaccard	sigmoid	0.1028	0.0811	0.0373	0.0467	0.1028	0.0811	0.0464	0.0546
claroline-rand_50	GRU	jaccard	tanh	0.0981	0.0224	0.0305	0.0166	0.0981	0.0224	0.0375	0.0185
claroline-rand_50	LSTM	jaccard	tanh	0.0745	0.0282	0.0135	0.0215	0.0745	0.0282	0.0186	0.0227
claroline-rand_50	LSTM	bin_ce	tanh	0.0395	0.0262	0.0205	0.0247	0.0395	0.0262	0.0195	0.0258
claroline-rand_50	GRU	manhattan	tanh	0.0394	0.0094	0.0232	0.0146	0.0394	0.0094	0.0205	0.0095
claroline-rand_50	LSTM	manhattan	tanh	0.0202	0.0004	0.0004	0.0	0.0202	0.0004	0.0008	0.0
claroline-rand_50	LSTM	manhattan	sigmoid	0.0199	0.0003	0.0004	0.0	0.0199	0.0003	0.0008	0.0
claroline-rand_50	GRU	bin_ce-logits	tanh	0.0198	0.0004	0.0007	0.0007	0.0198	0.0004	0.0008	0.0
claroline-rand_50	LSTM	bin_ce-logits	tanh	0.0198	0.0005	0.0004	0.0	0.0198	0.0005	0.0008	0.0
claroline-rand_50	GRU	manhattan	sigmoid	0.0197	0.0006	0.0004	0.0	0.0197	0.0006	0.0008	0.0