Romero, O.; Wrembel, R. Data engineering for data science: two sides of the same coin. A: International Conference on Big Data Analytics and Knowledge Discovery. "Big Data Analytics and Knowledge Discovery, 22nd International Conference, DaWaK 2020: Bratislava, Slovakia, September 14-17, 2020: proceedings". Springer, 2020, p. 157-166. ISBN 978-3-030-59065-9
The final authenticated version is available online at https://doi.org/10.1007/978-3-030-59065-9_13

Data Engineering for Data Science: Two Sides of the Same Coin

Oscar Romero[1][0000−0001−6350−8328] and Robert Wrembel[2][0000−0001−6037−5718]

[1]Universitat Politècnica de Catalunya, Catalunya, Spain
[2]Poznan University of Technology, Poland
oromero@essi.upc.edu, robert.wrembel@cs.put.poznan.pl

Abstract. A de facto technological standard of data science is based on notebooks (e.g., Jupyter), which provide an integrated environment to execute data workflows in different languages. However, from a data engineering point of view, this approach is typically inefficient and unsafe, as most of the data science languages process data locally, i.e., in workstations with limited memory, and store data in files. Thus, this approach neglects the benefits brought by over 40 years of R&D in the area of data engineering, i.e., advanced database technologies and data management techniques. In this paper, we advocate for a standardized data engineering approach for data science and we present a layered architecture for a data processing pipeline (DPP). This architecture provides a comprehensive conceptual view of DPPs, which next enables the semi-automation of the logical and physical designs of such DPPs.

Keywords: data science · data analytics · data engineering · data management · data processing pipeline

1 Introduction

Within the last years *data science*, whose main goal is to extract business value from massive data, has become the most popular research and technological topic as well as the most wanted IT profession, e.g., [12].

Extracting value from data requires deploying a *data processing pipeline* (DPP), which typically includes the following major tasks: (1) integrating heterogeneous and distributed data, (2) cleaning and standardizing data, (3) eliminating duplicates, and (4) storing cleaned data in a centralized repository. Tasks (1)-(4) represent the common backbone to ingest, model, and standardize relevant data, for the final goal to make them ready for analysis. Note that these tasks are generic and therefore, not tailored to a specific data analysis task. Once data are cleaned and stored in a repository, they can be analyzed by the following tasks: (5) extracting a data view from the centralized repository (and potentially persisting it in another repository), (6) specific pre-processing for a given analytical task at hand, (7) creating test and validation data sets (which also include labeling data for supervised machine learning), (8) and analyzing the data by means of descriptive statistical analysis (e.g., reporting or OLAP)

or by advanced data analysis: predictive approaches (e.g., machine learning) or graph analytics. This step also includes the deployment in run-time environments of the model created. The design of the DPP requires multiple iterations throughout the aforementioned tasks before deploying a final model. These tasks are based on both data engineering and data analysis technologies.

Data engineering primarily refers to data management, i.e., data ingestion, storage, modeling, processing, and querying, cf. [16], but also embraces system building aspects, e.g., service creation and orchestration. As reported by experts in the field [1, 2], up to 80% of time devoted to build a DPP is spent on *preprocessing data* (i.e., tasks (1)-(3) and (6)). This is because such tasks have not been standardized yet and practitioners are approaching them in a case-by-case manner. However, these tasks can benefit from data engineering solutions that could help standardizing them. Indeed, standardization is inevitable in order to achieve the semi-automatization of the most time-consuming and error-prone DPP tasks [14].

A USA job market analysis [17] highlights the relevance of data engineering for data science. Out of the 364,000 estimated data scientist jobs openings in 2020, only 62,000 require analytical skills, and the rest require blended engineering and analytical skills. Similarly, [12] highlights the shortage of 400,000 data workers in Europe blending data engineering and data analysis skills.

A successful application of standardizing DPPs has happened in the domain of Business Intelligence (BI), where for over two decades the industry is applying a reference architecture for data management and On-Line Analytical Processing (OLAP). In this architecture, called the *data warehouse architecture* (DWA), heterogeneous and distributed data sources are integrated by means of a layer called *extract-transform-load* (ETL) or its *extract-load-transform* (ELT) variant. This layer is composed of standardized, i.e., **pre-defined and orchestrated** data processing tasks, which are responsible for a **systematic** preparation of data for further analysis. Then, the data are represented by a multidimensional model that eases the analysis. Note that, for us, standardization does not mean full automation but structuring and orchestrating the DPP in such a way that the system is able to *understand* how data are processed. Indeed, this is the case of ETL DPP pipelines, whose design can hardly be automated, but there are multiple powerful tools that standardize these pipelines as well as facilitate their creation, maintenance (including optimization), and deployment (e.g., IBM Data Stage, Informatica PowerCenter, Ab Initio).

Unfortunately, a standardized approach for data science has not been proposed yet. As a consequence, many companies deploy independent fragments of the whole DPP using multiple technologies and often choose an inadequate technology for a task at hand (e.g., using analytical tools such as R, SAS, or Python scripts for typical data engineering tasks such as integrating heterogeneous and distributed data). This results in non-optimal scenarios, as highlighted in the Beckman Report [3], which advocates for more comprehensive and automatable DPPs. In the same spirit, [6] calls for adapting data management techniques to the new analytical settings. [4] recommends new solutions for developing an

'end-to-end data-to-insights pipeline', which would encompass among others: metadata management, data provenance, declarative way for defining a data pipeline and its optimization with the support of machine learning techniques. Finally, [19] discusses challenges related to DPPs, w.r.t. data organization, data quality, and feature engineering for analytics, based on real-world use cases.

To sum up, in the current data science approaches, data engineering solutions are mostly neglected. Typically, data engineering and analytical tasks, are conducted by independent teams, resulting in multiple independent DPPs. For this reason, a standardized single end-to-end design of a DPP is needed for the system to understand how data are processed through the whole pipeline and to open the door for semi-automatic optimization of the DPP.

In this paper we propose a **complete *rethinking* of DPPs, such that they are orchestrated and unified into one solution**. Accordingly, we discuss how DPPs can benefit from data engineering solutions to standardize, semi-automate, and facilitate data management. Finally, we present an augmented architecture that unifies data engineering and analytical tasks, offering an integrated development, optimization, and execution environment. Section 2 motivates our work with real use cases. Section 3 presents our vision on the augmented architecture.

2 Motivation: Real Cases

In this section we outline two real projects, we currently run, which apply DPPs in a typical way. These particular projects were selected for being representative of practices conducted in several organizations with which the authors collaborate. We especially focus on the bad practices from the data engineering point of view. The practices were identified when these projects started.

- **Development of a data-driven culture in a company**. This project is run at Universitat Politècnica de Catalunya in collaboration with an international company that has several departments developing advanced predictive models for decision making. Each department had a different goal and they focused on different domains. However, most of the data they used were in common, but each department created their own processes to build models. Yet, there was no common data backbone and they independently built their DPPs from data copies stored locally in each department. These copies were in the form of relational database dumps or CSV files, in most of the cases. Importantly, most of the employees were advanced data analysts (with a strong statistical background) but had not been trained in data management techniques. Thus, they did not know where to find relevant data variables for their day-by-day tasks. Simply, they used the datasets at hand within the department and each of them, even inside the same department, created their own DPPs. As the result, even if they were able to share their DPPs in the form of notebooks, each of them executed an independent DPP.
  The company identified this lack of data governance as a main drawback since: DPPs were not optimized, data analysts did not have access to relevant

variables and data were replicated all over the company without control. Going back to the main tasks described in Section 1, this solution neglected the following data engineering solutions:

- Tasks (1)-(4) were not considered at all. They were embedded inside specific analytical DPPs dispersed within the company.
- Task (5) was not implemented. Data were not selected from a common repository but extracted from partial copies. Thus, analysts did not have a clear view on available variables.
- Tasks (6)-(8) were typically conducted in notebooks, on local workstations. Thus, code snippets were executed from scratch each time, without the possibility to share intermediate common results with other DPPs, causing data redundancy.

– **Predicting consumption of thermal energy**. This project is run at Poznan University of Technology for a Polish company that builds co-generating energy grids. The company collects measurements on thermal energy consumption of their customers (a measurement is taken every hour). Monthly measurement data are stored in separate files. The initial set of data includes over 160 csv files with measurements and weather parameters (for years 2016-2020). The goal of this project is to build separate statistical models and ML models for predicting thermal energy consumption. Within the project only one DPP is used, but it is essential to the whole decision support system. All the steps used in the DPP are implemented in Python. With respect to the main tasks described in Section 1, this solution neglected the following data engineering solutions:

- Tasks (1)-(3) were implemented as Python scripts and run in a Jupyter notebook on a local workstation. As a consequence, a private cleaned dataset was created each time the script was run.
- Task (4) was implemented as csv files.
- Task (5) was not implemented at all, as data were stored locally in OS files. As a consequence, data were not shared and access to data was non-optimal.
- Tasks (6)-(8) were executed in a Jupyter notebook on a local workstation. For this reason, pre-processing, intermediate results, and model building were executed from scratch each time, without the capability to share the results.

To sum up, from the aforementioned projects we can draw the following observations. First, omitting tasks (1)-(4) in a DPP compromises **data governance**, **persistence**, **concurrency**, **safety**, **security**, and **performance** [16]. Second, conducting tasks (6)-(8) independently leads to the same problems. Third, omitting task (5) compromises **data sharing**, since there is no single source of truth. The observations from the aforementioned projects and other similar we participated in, result in the augmented data processing architecture that we propose in this paper.

3 Augmented Data Processing Architecture

In this section **we discuss how to overcome the main disadvantages of a data science approach** to building DPPs and **propose an architecture that promotes the benefits of using data engineering solutions**. In particular, this section highlights four essential challenges to advance in the architecture. The architecture we propose is composed of three layers, as shown in Figure 1. We exemplify it by means of the DPP from the project discussed in Section 2.

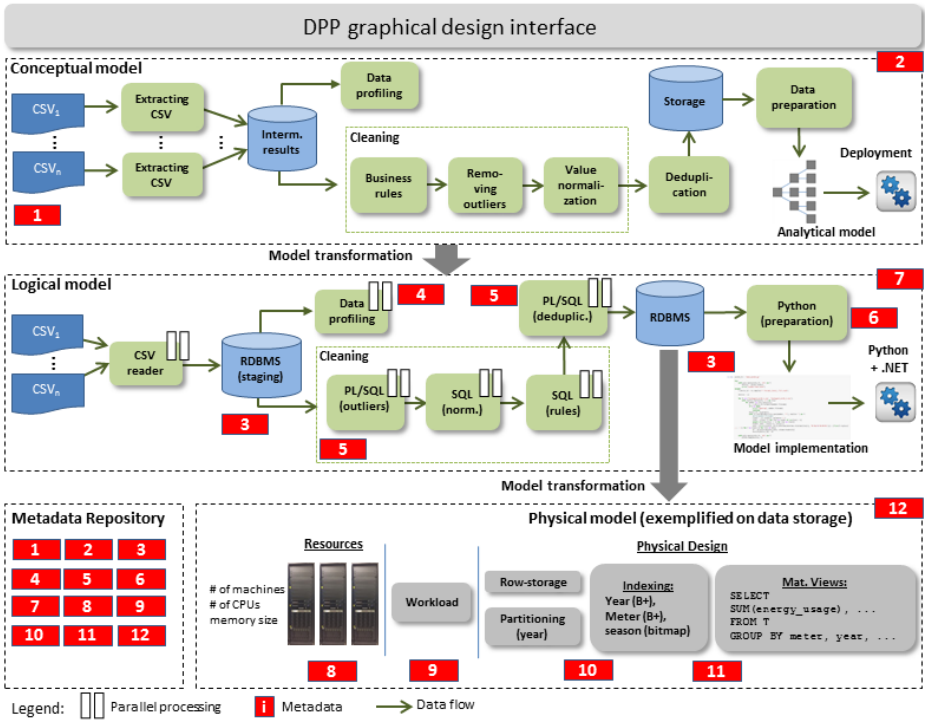

Fig. 1. The augmented data processing architecture

The top layer represents the *conceptual model* of a DPP. The model is designed by means of a unified processing language (cf. Section 3.1). The common data engineering and data analysis tasks (cf. Section 1) are standardized, i.e., available from a pre-defined palette of tasks. Additionally, the DPP designer can implement UDFs. This model abstracts from an implementation, i.e., from particular technologies and optimization techniques of the DPP. At the conceptual layer we envision user-friendly interfaces in the line of Swan - the data mining service [20] and, to a minor extent - JupyterLab.

The *logical model* instantiates the conceptual one and chooses the appropriate technologies. The logical model leverages workflow optimization techniques (cf. Section 3.3) and applies specific data engineering and data analysis solutions per task (e.g., streaming, batch processing, in-memory), including: (1) the **type** of DBMS, e.g., relational, key-value, graph and (2) the most suitable implementation language for each task.

The *physical model* defines physical components, and their parameters, for the logical model. It includes among others: (1) memory size of workstations, (2) the number of CPUs and threads, (3) physical data layout (e.g., column store or row store), (4) physical data structures in a database, (5) if applicable, the size of a hardware cluster.

The transitions from the conceptual to logical and from logical to physical model resemble that of current DBMSs, and are executed automatically to deliver a final deployment of the DPP that fulfills the user requirements. The transitions are guided by requirements (in the spirit of service level agreement), e.g., execution time, monetary cost, or quality of analytical results (quality of models). The requirements are represented by various types of metadata stored at each of the three levels (cf. Section 3.2).

Note that such **architecture is feasible by means of to virtualization**. Thus, multiple variants of such an architecture can be built and tested, before being deployed. We identified four essential challenges to build the architecture.

3.1 Challenge 1: Unified Data Processing Language

DPPs are typically created at the logical/physical level forcing the user to orchestrate multiple technologies in a DPP. In order to free the user from this burden, we advocate to use a unified data processing language (DPL) to define and maintain DPPs. To this end, the DPL should sit at the conceptual level and be technology agnostic. Further, the language constructs must cover all tasks enumerated in Section 1 and help to automatize the most common data engineering tasks. Ideally, we envision a system equipped with a friendly GUI. The DPL should have a declarative counterpart at the side of the user, in order to alleviate the designer from writing optimal procedural code and to move the burden of optimizing its execution into the system. In this context, some development have already been made, e.g., PySpark - to access a Spark instance from a Jupyter notebook, SparkR - to run R on Spark, KSQL - to execute declarative queries on a Kafka stream, and nipyapi - to access NiFi from Python, but they still does not offer a homogeneous declarative programming environments.

The DPL should include the required information to automate as much as possible the other layers (cf. Fig. 3). The minimum set of features supported by the DPL should include: (1) task orchestration (similar to that of ETL tools), including off-the-shelf operations, (2) notebook-style coding for writing tailored code, i.e., a user-defined function (UDF), and (3) on-line recommendations. The recommendations must include the use of machine learning algorithms to learn the most suitable tasks for a given type of processing. A few approaches have already been proposed for on-line recommendations of DPP tasks, e.g., [10] -

where we proposed ML techniques for data pre-processing for classification, or [21] - that addresses hyper-parameter tuning of ML models.

3.2 Challenge 2: Metadata Management

Metadata are essential to: (1) enable data governance, (2) automate different tasks within a DPP, (3) transform one model into another, (4) optimize execution of a DPP, and (5) self-tune the system. Data governance refers to the capability that enables an organization to ensure that high quality data exist throughout the complete data life-cycle. Data governance encompasses characteristics of data sources, data themselves, and data transformations. In the architecture shown in Fig. 3, the following *metadata categories* are needed, in the spirit of [23] (metadata artifacts are depicted in Fig. 3 as rectangles enumerated from |1| to |12|):

- *Data source description* |1|: describing a content and structure of each source.
- *DPP conceptual design* |2|: describing technology-agnostic tasks and their orchestration, represented by conceptual workflow models, such as BPMN.
- *Capabilities of data storage models* |3|: describing a type of data model and its fundamental features: data representation alternatives and data access plans (i.e., full scan, range scan, random access). For example, the metadata may describe relational databases, their row-oriented data storage, and their wide range of indexing capabilities (B+, hashing, bitmap). These metadata is used to select a type of technology and not a specific DBMS.
- *Data characteristics* |4|: describing the set of data characteristics that are required to automate the transformation from the logical to the physical model, obtained by traditional data profiling techniques [5].
- *Rules for data cleaning and deduplication* |5|: describing domain rules for data cleaning and removing duplicates. Such rules must be described in a computer processable manner.
- *Data preparation rules* |6|: describing the rules preparing data for the analytical task at hand, e.g., discretization of values or value imputation [10].
- *DPP logical design* |7|: describing tasks and their orchestration, including the type of technologies chosen, based on the previously mentioned metadata artifacts; e.g., as a data storage a relational DBMS may be selected with a column store and a particular partitioning scheme.
- *Hardware characteristics* |8|: describing the physical parameters needed for cost-based data access optimization (DAO) [11] of the system (e.g., a size of a computer cluster, the number of available CPUs and memory size, disk block size, disk and network bandwidth).
- *Workload description* |9|: describing the analytical workload. Specifically, the characteristics of queries (e.g., the number of tables accessed, functions used) retrieving the needed data for the analytical tasks and frequencies of the queries. These metadata are required by cost-based DAO.
- *Physical data characteristics* |10|: describing the characteristics of stored data (e.g., data cardinality, average record size, number of blocks occupied by a table or any other structure), which are required for cost-based DAO.

- *Characteristics of physical data structures* |11|: describing physical structures created as part of the self-tuning process, e.g., partitioning schemes or indexes.
- *Run-time monitoring* |12|: describing error logging and execution statistics of the DPP (e.g., query execution time as well as elapsed processing time, CPU time, and I/O of the whole DPP). These statistics are essential for self-tuning capabilities.

All the components of the architecture must be able to generate and consume inter-operable metadata. To this end, we envision a dedicated subsystem, called a *metadata repository* that systematically gathers metadata from all layers and enables metadata exploitation for diverse tasks, i.e., either data governance or automation [18]. Even though multiple metadata representation and metadata management techniques have been have been proposed, a metadata standard for such a complex architecture has not been developed yet.

3.3 Challenge 3: Workflow Optimization

The whole DPP needs to be optimized using solutions from the cost-based query optimization [11]. The problem of DPP optimization is similar to ETL optimization, which a very difficult and still open problem (c.f., [8] for the state of the art). DPP optimization is becoming more difficult in the presence of UDFs, as they are typically treated as black boxes, i.e., their performance characteristics are not known in advance and their semantics cannot be inferred automatically. To attack this problem, [22] propose to infer a UDF behaviour by means of experiments. [13] proposes to include formalized annotations when deploying an UDF, so that the system can learn basic behavioral aspects of the UDF from the annotations. Finally, we proposed an architecture and method for executing UDFs in a parallel environment [9]. The optimization of a DPP in the architecture presented in Figure 3 should draw upon these solutions.

The overall optimization idea that we envisage in the architecture is the following. First, a designer creates a DPP in the design interface, using the combination of predefined tasks and UDFs. Second, based on the cost-based optimization techniques the system builds an optimal (in reality sub-optimal) execution model and executes it. Next, from each execution, statistics are collected for the optimizer to learn and evolve the execution model for future use.

3.4 Challenge 4: Self-tuning Capabilities

The system that we propose must support the following self-tuning features:

- From the available data models (e.g., relational, object, semi-structured, NoSQL, graph), identify the most suitable for a common data pipeline.
- Identify the most suitable database type. To this end, each database can be identified by its capabilities and therefore, it should be possible to select the most adequate to a given problem [15].

- Identify the best physical data layout (i.e., row-store, column-store, and pointer-based), taking into account their features [7].
- Identify the best physical data structures to be deployed in the chosen database, i.e., indexes (bitmap, join, B-tree like, hash, spatial), clusters, partitions, materialized views - all of them offer different performance characteristics.
- Identify relevant intermediate results to be materialized to optimize the DPP execution.
- Decide the physical architecture - single node vs. cluster, and types of machines (e.g., disk-based vs. main memory, number of CPUs, RAM size, disk type) and their number.

All of the aforementioned features need to be described by metadata and a cost model of the whole system must be developed. To the best of our knowledge, neither a metadata standard nor a cost model for such complex systems has been developed yet.

4 Conclusions

In this paper we claim that data processing pipelines used by data scientists and by data engineers are immature and do not fully exploit the advantages of the existing data engineering technologies. Current approaches (i.e., *data analytics* and *data engineering*) have their own advantages and disadvantages and they can be substantially improved to increase: development productivity, processing performance, and self-tuning capabilities. Following our claim, we propose an architecture for a modern data processing pipeline with augmented functionalities: (1) a unified data processing language, (2) metadata management, (3) workflow optimization, and (4) self-tuning capabilities. Building and testing such architectures is feasible by means of the virtualization technologies.

The approach that we propose not only aims at improving a DPP design, performance, and maintenance, but also opens numerous new research and technological directions, including:

- the development of a unified data processing language for designing DPPs at the conceptual level;
- a metadata standard for sharing and exchanging metadata by all the DPP components at the conceptual, logical, and physical level;
- cost-based and dynamic optimization of DPPs, including multi-flow optimization;
- optimization of DPPs with user-defined functions;
- the development of a cost model for a complex, multi-layered system;
- self-tuning technologies, e.g., based on machine learning, to optimize DPPs at the logical and physical levels.

Such extensions should open the door to reach DPP-as-a-service.

References

1. Data Warehouse Trends Report. Technical report, Panoply, 2018.
2. Data Engineering, Preparation, and Labeling for AI 2019. Technical report, Cognilytica Research, 2019.
3. D. Abadi, R. Agrawal, A. Ailamaki, et al. The beckman report on database research. *Communications of the ACM*, 59(2):92–99, 2016.
4. D. Abadi, A. Ailamaki, D. Andersen, et al. The seattle report on database research. *SIGMOD Rec.*, 48(4), 2020.
5. Z. Abedjan, L. Golab, F. Naumann, and T. Papenbrock. *Data Profiling*. Synthesis Lectures on Data Management. Morgan & Claypool, 2018.
6. S. Abiteboul, I. Manolescu, P. Rigaux, M. Rousset, and P. Senellart. *Web Data Management*. Cambridge University Press, 2011.
7. I. Alagiannis, S. Idreos, and A. Ailamaki. H2O: a hands-free adaptive store. In *Proc. of SIGMOD*, pages 1103–1114, 2014.
8. S. M. F. Ali and R. Wrembel. From conceptual design to performance optimization of ETL workflows: current state of research and open problems. *The VLDB Journal*, pages 1–25, 2017.
9. S. M. F. Ali and R. Wrembel. Towards a cost model to optimize user-defined functions in an ETL workflow based on user-defined performance metrics. In *Proc. of ADBIS*, pages 441–456, 2019.
10. B. Bilalli, A. Abelló, T. Aluja-Banet, and R. Wrembel. Intelligent assistance for data pre-processing. *Computer Standards & Interfaces*, 57:101–109, 2018.
11. S. Chaudhuri. An overview of query optimization in relational systems. In *Proc. of PODS*, pages 34–43, 1998.
12. European Commission. Towards a Thriving Data-driven Economy, 2018.
13. S. Ewen, S. Schelter, K. Tzoumas, D. Warneke, and V. Markl. Iterative parallel data processing with stratosphere: an inside look. In *Procs. of SIGMOD*, pages 1053–1056, 2013.
14. Forrester Consulting. Digital Businesses Demand Agile Integration, 2019.
15. V. Gadepally, P. Chen, J. Duggan, A. J. Elmore, B. Haynes, J. Kepner, S. Madden, T. Mattson, and M. Stonebraker. The BigDAWG polystore system and architecture. In *Proc. of IEEE HPEC*, pages 1–6, 2016.
16. H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database systems - the complete book*. Pearson Education, 2009.
17. IBM. The Quant Crunch Report, 2017.
18. S. Nadal, V. Herrero, O. Romero, A. Abelló, X. Franch, S. Vansummeren, and D. Valerio. A software reference architecture for semantic-aware big data systems. *Information & Software Technology*, 90:75–92, 2017.
19. A. Nazábal, C. K. I. Williams, G. Colavizza, C. R. Smith, and A. Williams. Data engineering for data analytics: A classification of the issues, and case studies. *CoRR*, abs/2004.12929, 2020.
20. D. Piparo, E. Tejedor, P. Mato, L. Mascetti, J. T. Moscicki, and M. Lamanna. SWAN: A service for interactive analysis in the cloud. *Future Generation Comp. Systems*, 78:1071–1078, 2018.
21. A. Quemy. Data pipeline selection and optimization. In *Proc. of DOLAP*, 2019.
22. F. Vaandrager. Model learning. *Communication of the ACM*, 60(2):86–95, 2017.
23. J. Varga, O. Romero, T. B. Pedersen, and C. Thomsen. Analytical metadata modeling for next generation BI systems. *J. of Systems and Software*, 144:240–254, 2018.