Building Domain-Specific Machine Learning Workflows: A Conceptual Framework for the State-of-the-Practice

BENTLEY JAMES OAKES, Département de génie informatique et génie logiciel, Polytechnique Montréeal, Canada
MICHALIS FAMELIS, Département d'informatique et de recherche opérationnelle, Université de Montréal, Canada
HOUARI SAHRAOUI, Département d'informatique et de recherche opérationnelle, Université de Montréal, Canada

Domain experts are increasingly employing machine learning to solve their domain-specific problems. This article presents to software engineering researchers the six key challenges that a domain expert faces in addressing their problem with a computational workflow, and the underlying executable implementation. These challenges arise out of our conceptual framework which presents the "route" of transformations that a domain expert may choose to take while developing their solution.

To ground our conceptual framework in the state-of-the-practice, this article discusses a selection of available textual and graphical workflow systems and their support for the transformations described in our framework. Example studies from the literature in various domains are also examined to highlight the tools used by the domain experts as well as a classification of the domain-specificity and machine learning usage of their problem, workflow, and implementation.

The state-of-the-practice informs our discussion of the six key challenges, where we identify which challenges and transformations are not sufficiently addressed by available tools. We also suggest possible research directions for software engineering researchers to increase the automation of these tools and disseminate best-practice techniques between software engineering and various scientific domains.

CCS Concepts: • **Computing methodologies → Machine learning**; **Knowledge representation and reasoning**.

Additional Key Words and Phrases: computational workflow, workflow composition, domain experts, machine learning, machine learning pipelines, software engineering framework

ACM Reference Format:
Bentley James Oakes, Michalis Famelis, and Houari Sahraoui. 2023. Building Domain-Specific Machine Learning Workflows: A Conceptual Framework for the State-of-the-Practice. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (January 2023), 49 pages. https://doi.org/10.1145/3638243

1 INTRODUCTION

The past two decades have seen machine learning algorithms, especially deep learning, permeate throughout every scientific, engineering, and business domain to enable new techniques and solve complex challenges. One example of

Authors' addresses: Bentley James Oakes, bentley.oakes@polymtl.ca, Département de génie informatique et génie logiciel, Polytechnique Montréeal, Montréal, Canada; Michalis Famelis, michalis.famelis@umontreal.ca, Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, Canada; Houari Sahraoui, houari.sahraoui@umontreal.ca, Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, Canada.

many is the recent solving of the long-standing protein folding problem, which focuses on how a protein will fold in three-dimensional space given its one-dimensional representation [67].

The enormous power offered by current machine learning techniques is therefore of great interest to stakeholders across all domains. However, utilising these techniques often requires a user who is an expert in their own domain to gain proficiency in not only the concepts of machine learning but also the programming abilities used to call the low-level libraries. This is undesirable as the *domain expert* would like to reason about concepts and terms that are in their own domain. For example, the literature on *domain-specific languages* shows that solving a problem in the *problem domain* is more efficient than solving it in the *solution domain* [68, 127].

Recently, the rise of *low-code* platforms partially addresses this issue by raising the level of abstraction from writing code to interactively defining procedures and GUIs [14, 18]. A domain expert (or "citizen developer") is thus assisted to build applications or computations using an easy-to-use and easy-to-deploy interface. The domain expert may also be able to select *domain-specific* components which directly address concerns in their domain such as IoT [62], or *machine learning* components to simplify machine learning tasks.

Research Problem. In this article, we focus on this intersection of *domain experts*, *low-code solutions*, and *machine learning*. Specifically, we are interested in the *flow-based* [92] nature of *workflows*, which are the typical presentation of computations in low-code platforms and *scientific computing frameworks*. That is, these workflows are composed of computational blocks arranged in a graph structure with the edges denoting data or control dependencies. We note that this representation perfectly matches with the common notion of a machine learning *pipeline* where data is ingested, cleaned, and trained upon to produce a machine learning model [13].

The research question that arises is thus: *Given a domain expert and a domain-specific (DS) problem, what are the techniques required such that a deployable workflow solution involving ML can be easily obtained?*

Contributions. Our main contribution to address this research question is a conceptual framework to map out the tools and techniques for solving a DS *problem* using a *workflow* which involves ML, where that workflow is then deployed into an executable *implementation*. This framework illustrates that there are multiple *choices* or *routes* that a domain expert may choose from to obtain an executable implementation from their problem. For example, a domain expert may wish to first consult an *expert mapping* to determine how the problem should be structured in a ML representation, before they construct a workflow by *manually selecting suitable components from a repository*.

These three layers are divided into *regions*, where *problems*, *workflows*, and *implementations* are organised by the two dimensions of *domain-specificity* and *complexity of machine learning*. The choices of the domain expert can then be seen as *transformations* between different regions of our conceptual framework. These transformations are phrased as interesting challenges which a domain expert who wishes to use machine learning must overcome. These challenges are further explained in Section 3.2.

In other words, we attempt to organise a) the meaning of, and b) tools and techniques such that a domain expert can:

- Map a DS problem to a form suitable for ML
- Obtain a solution workflow for a DS and/or ML problem
- Experiment with ML tools and techniques within a workflow
- Add DS knowledge to improve ML performance (e.g., feature engineering)
- Produce an implementation from a workflow which is well-suited for a domain expert (in terms of scalability, DS tooling, etc.)

- Extract a workflow from an existing implementation (code, Jupyter notebook)

For example, the first challenge is to assist the domain expert in mapping their problem to a ML solution, such that the barrier to understanding the ML concepts is reduced. We have identified that this mapping is likely to be performed through *formal reasoning*, *expert mapping*, or a *data-driven approach*. Huang *et al.* describe such an expert mapping in the domain of genomics [61]. Their work assists genomics researchers by mapping drug and tissue structures from the domain into forms suitable for ML, defining the problem as a ML problem, and recommending suitable ML techniques for solving the ML problem.

To ground the above framework and challenges, we discuss some state-of-the-practice tools which cater to both domain experts and the use of machine learning. This provides insights into the possibilities for providing assistance, and grounds the transformations in our framework. Third, we present six example studies from the literature from different fields, where the study includes the use of machine learning. We discuss the tools the authors utilised and present a rough categorisation of the components in their workflow, allowing us to infer the route through our framework that the domain experts have taken. We also present a discussion concerning the software engineering challenges raised, and future directions for each of these challenges. To encourage the adoption and application of this conceptual framework, we also provide an illustrative use of our framework and potential evaluation metrics.

Overall, this article is intended for software engineering researchers as it provides an analysis of the current issues and tools concerning the use of machine learning by domain experts. We believe this to be a strong contribution which can form the basis for important software engineering research directions and tools in the future. In particular, we intend for this conceptual framework to be used by researchers and tool builders to organize the transformations to be studied and implemented in tools for domain experts.

In other words, our conceptual framework offers an explicit methodology for domain experts to obtain an implemented solution from their problem by selecting these transformations. In this paper, we present how domain experts are partially following this methodology by a) highlighting where transformations are not supported in current tools, and b) describing case studies where users transform their artefacts through regions in our framework. In future work, we hope to provide complete tool support for this conceptual framework, where the domain expert would be able to follow this methodology in one tool ecosystem as described in our illustrative case in Section 7.4.

Article Structure. As our framework relates topics from different disciplines, Section 2 provides background on the topics of *domain-specificity*, *machine learning*, *workflows*, and some *workflow tools*.

Our three-layer conceptual framework relating *problems*, *solution workflows*, and *implementations* is presented in an overview in Section 3. Section 4 discusses each layer of this framework in turn including the transformations within each layer. Section 5 then presents the transformations between layers. These inter-layer transformations involve turning problems into workflows, and from workflows into an implementation.

Example studies are presented in Section 6 to detail how practitioners from various fields are employing workflow tools and executing these transformations on their own problems. Section 7 then uses the framework as a basis for a discussion of the opportunities and challenges for implementing and automating the transformations in the framework. Section 8 presents our concluding thoughts.

2 BACKGROUND

This section provides a brief background in three core topics of this article: the concept of *domain-specificity*, *machine learning*, and *computational workflows*.

2.1 Domain-Specificity

In this article, we discuss the idea of *domain-specificity* as relating to the concepts of a particular domain. This is relevant throughout our framework as a) a domain expert had specific experience and insights into that domain and is less familiar with concepts outside that domain, b) the problem the domain expert has is (partially) expressed using those domain concepts, and c) the technical considerations for a solution such as computing platforms and tools may be specific to that domain. In this article, *domain-specificity* is manifested as a cross-cutting *dimension* of the framework as discussed in Section 3.1. In particular, we specify that *problems*, *workflows*, *components*, and *implementations* can be more or less *domain-specific*. In other words, these elements are *domain-specific* when they are *relevant to experts in that domain, and not relevant to other domains.*

For example, consider the domain of *neuroscience* which is the study of the nervous system. Relevant concepts to a neuroscientist include *neurons*, *signals*, *behaviour*, *activation*, *brain hemispheres*, *neural circuits*, and *brain damage*, with more according to the sub-field of the neuroscientist. These concepts may be formalised in an *ontology*[1], allowing for more precise or (semi-) automated reasoning about the concepts and their connections.

Along with these concepts, the data examined by this expert is highly specific to the domain. In neuroscience, this can include processing of functional Magnetic Resonance Imaging (fMRI) files requiring specialised techniques to handle *motion correction* (correcting the movement of the subject within the scanner) and *smoothing* of the data (to average out the noise present in the measurement). Finally, the datasets, tools, and computational platforms available to a neuroscientist are highly domain-specific such as repositories of mouse brain scans[2].

As a domain expert is most knowledgeable about concepts from their domains, we follow the approach of *domain-specific modelling* in declaring that problems should be solved at a high level of abstraction using domain-specific concepts [68, 127]. That is, the domain expert should use *domain-specific languages* instead of general programming languages, and there should be a layered approach when possible to hide technical and implementation details. This approach has been shown to lower the cognitive workload of learning new concepts and resulting in increased productivity [127]. In the context of this article, we are interested in defining possible tool support such that the domain expert can describe their problem and workflow using domain-specific concepts.

2.2 Machine Learning

Machine learning (ML) can be summarised as "programming computers to optimise a performance criterion using example data or past experience" [4]. That is, based on a collection of data and experiences, ML seeks to automatically create *models* capable of relating output for particular inputs without requiring a programmer to directly implement the steps for computing such outputs. These definitions therefore capture many applications ranging from interpolating based on a simple linear regression up to automated driving by using visual data interpreted using neural networks.

One way of categorising ML approaches is into three approaches: *supervised learning*, *unsupervised learning*, and *reinforcement learning*. In *supervised learning*, the supervisor provides inputs and labelled outputs, and the technique must learn the mapping between inputs and outputs. An example would be to train a linear regression of variables, producing a classifier which can predict whether a tissue sample is a tumor or not [47], or constructing a similarity function such that related objects can be found. In *unsupervised learning*, the technique learns without provided labels. This can offer insights into the structure of the domain such as uncovering related clusters of data or outliers. In

[1]See https://github.com/SciCrunch/NIF-Ontology for an example.
[2]For example: https://neuinfo.org/.

reinforcement learning, the intelligent agent is rewarded with a defined reward function. This allows the agent to explore possible actions and receive automated feedback.

A ML model must be *trained* before it can be used for *reasoning*. For example, consider the scenario of training a model for a “line-of-best-fit” (linear regression). Once the data points have been loaded and any cleaning necessary performed, the data is then commonly divided into *training* and *test* sets. This allows for an unbiased measure of error when testing, as the model has not “seen” the testing data beforehand. The linear regression is then learned by a suitable algorithm, and the linear regression is the produced ML model ready for use. When this model is used it is fed a new piece of data as input, and it will predict a particular output.

Many libraries are available which implement some form of ML. For example, scikit-learn[3] is a popular Python module which offers a high-level API for ML techniques, while Keras[4] provides an API for directly creating neural networks by constructing each layer into a model. ML techniques are also available directly within academic and scientific tools such as the MATLAB Statistics and Machine Learning toolbox[5].

Machine Learning Pipelines. In ML, the term *pipeline* is commonly used to denote the steps involved in the *training* and *reasoning* processes. For example, data can be involved in a linear flow of *loading*, *filtering*, *cleaning*, *splitting* (into a training and testing set), and *trained upon* [13].

A similar linear flow is also present for the neural networks used in *deep learning*. The input data passes through *layers* in this network which recognise patterns in the input data, store relationships in the weights between nodes in inner layers, and then produce output values or categories.

Thus, ML pipelines are a one-to-one match to the workflow formalism we examine in this article. This unification of structure is important when we discuss workflows which involve components from both a particular scientific domain as well as ML components.

2.3 Computational Workflows

General workflows have existed for many years, especially in the manufacturing domain. In this article, we focus on *computational workflows* which define the steps that a computational device follows to produce results.

In particular, we are interested in workflows which represent *flow-based programming* by containing discrete steps representing computational steps [92]. This can be represented by a directed-acyclic graph (DAG) of computational nodes connected by control and data dependencies. Note that in current workflow systems this restriction on acyclic graphs is relaxed, as tools may wish to encode control flow structures such as loops over input files.

This concept of data “flowing” through a workflow is quite a natural structure for many computations we examine in this article. Indeed, some domains may lean into the plumbing metaphor and refer to the workflow as a “pipeline”. Lamprecht *et al.* state that, “Another common, more differentiating view is that pipelines are purely computational and as such a subset of the more general notion of workflows, which can also involve a human element" [78]. To clarify the terminology in this article, we only discuss *computational workflows*. That is, the term ‘workflow’ in this article do not contain human elements and can therefore be equated to *pipelines*.

[3]https://scikit-learn.org
[4]https://keras.io
[5]https://www.mathworks.com/products/statistics.html

2.3.1 Workflow Formalisms and Standards. *Workflows* can be represented in the simplest form as a connected and directed graph, where nodes in the graph are computations and the edges are dependencies of data or control. Extending beyond this representation are well-known formalisms which can also represent workflows.

For example, Petri Nets [110] can allow for formal verification of properties such as liveness for the system, or a Formalism Transformation Graph + Process Model (FTG+PM) can record the formalisms and transformations employed in the workflow [22]. Another well-known workflow standard is the Business Process Model and Notation (BPMN) [23] to formalise both automatic and manual workflows within an organisation.

Bringing together both Petri Nets and BPMN is Yet Another Workflow Language (YAWL)[6]. This workflow language from the 2000's takes Petri Nets as a starting point and adds extensions for commonly-seen workflow patterns [124]. The formal semantics of YAWL allow for verification of workflow properties such as *soundness* (ensuring *an option to complete*, *proper completion*, and *no dead transitions*) [136].

There are an overwhelming number of workflow management systems available for use. Amstutz *et al.* maintain an incomplete listing of more than 300 workflow systems [5]. These systems can be stand-alone or integrated into various platforms[7]. More recently, a number of workflow standards have been developed in various sub-fields but none has yet established dominance over the others [27]. This may soon change with convergence on the Common Workflow Language (CWL)[8].

CWL originated in the bioinformatics community and offers a declarative workflow definition language (a DSL) that can be written in JSON or YAML to be executed by a workflow execution engine [26]. Of particular interest to this report is that DS attributes can be added to workflows and their steps as needed by users, allowing for a great deal of flexibility and discoverability for domain experts. The standard is becoming established throughout multiple domains and has a number of implementing tools, including upcoming support in the graphical Galaxy workflow tool.

2.3.2 Workflow Management Considerations. The large scale of scientific data and the *reproducibility* requirements of scientific processes have encouraged the growth of *workflow management systems* within various scientific domains [115]. For example, ensuring that data sources and computations follow the well-known principles of *Findable*, *Accessible*, *Interoperable*, *Reusable* (FAIR) [131] demands a comprehensive management system. The life sciences in particular have a rich history of workflow systems [78, 84, 103, 106, 133].

As workflows explicitly declare the sequence of computations they perform, they assist in the production of *reproducible* results [121, 133]. That is, they assist to reproduce the results of another experiment [63]. For example, the discrete nature of workflow components means that components can be tagged with provenance information [109] and placed into containers such as Docker containers[9]. This allows for easily accessible, self-contained units which can be accessed through repositories and placed into dependency management systems [34].

Many factors can influence whether a computational process is reproducible, and workflows are only one step towards full reproducibility. For example, Digan *et al.* discuss 40 reproducibility features sourced from recommendations and clinical usages of workflows in the natural language processing (NLP) domain [36]. Mora-Cantallops *et al.* discuss reproducibility in the context of artificial intelligence/ML [91].

[6]https://yawlfoundation.github.io/

[7]These workflows are commonly referred to as "visual scripting". For example, https://unity.com/products/unity-visual-scripting, https://www.blackmagicdesign.com/ca/products/davinciresolve/fusion, and https://lensstudio.snapchat.com/guides/visual-scripting/. Lens Studio is of particular interest with the integration of ML algorithms within the workflow.

[8]https://www.commonwl.org

[9]https://www.docker.com

There are also many other "ilities" relevant for domain experts to use workflow systems. For example, Wratten *et al.* evaluate twelve bioinformatic workflow managers using the categories of *ease of use*, *expressiveness*, *portability*, *scalability*, *learning resources*, and *pipeline initiatives* [133]. Admed *et al* mention *modularity* and *reproducibility* amongst others [2], while Kortelainen adds the important characteristics of *licensing* and *maturity* [71]. Other factors may be connection to specialised tools or computing platforms such as the Hermes middleware platform for increased scalability [70].

2.4 Workflow Tools and Management Systems

This section will touch upon some current workflow systems and report some of the interesting features and considerations implemented for their use by domain experts.

2.4.1 Text-Based. Current text-based workflow systems seem to follow two approaches: either the system is implemented as a module/library for a general programming language such as Python, or the system ingests a standard *markup language/DSL*.

Language Module. A common implementation strategy for workflow tools is to leverage the user's knowledge of a general programming language. Commonly, this is Python due to its widespread usage.

For example, *luigi* is a tool from Spotify[10] which allows a user to build up a dependency graph of *Tasks* which interact with *Target* files. These concepts are defined within Python code and the Luigi API offers access to common database/cloud tools. A web-based scheduler and visualiser is also available for monitoring long-running workflows.

Luigi was extended by Lampa *et al.* into SciLuigi [76] for scientific workflow requirements such as a separation of the workflow and the tasks, audit support, and support for high-performance computing. The authors then developed *SciPipe*[11] in the Go programming language for enhanced type-safety and performance [77].

Workflow systems defined as language modules can also be tailored to particular domains, further reducing the amount of code a domain expert must write to use the workflows.

For example, the *automate* tool[12] for computational materials science [88] offers workflows to copy and customise based on specific analysis of materials, and an API to the analysis tools themselves. atomate uses the FireWorks workflow software which provides provence and reporting support for high-throughput computations [65].

Another example of a domain-specific library for workflows is the *nipype* Python software package[13] to define workflows in neuroimaging [57]. The intention here is to define components commonly used in neuroscience and have them as part of the same workflow. This allows domain experts who know Python to quickly build a workflow of neuroscience-specific tools.

Moving one level of abstraction higher, *fMRIPrep*[14] is an automated workflow built on top of nipype [39, 40]. fMRIPrep adapts to the input data automatically to performing the appropriate preprocessing steps for functional magnetic resonance imaging (fMRI), such as head motion correction and skull stripping. This assists in providing replicable results for neuroimaging studies both in terms of computation and by providing "boilerplate" natural language text for insertion into a research article's method section.

[10]https://github.com/spotify/luigi
[11]https://github.com/scipipe/scipipe
[12]https://atomate.org
[13]https://nipype.readthedocs.io/en/latest/
[14]https://fmriprep.org/en/stable/

Replicable results are also important when considering the development of ML models. The emerging field of Machine Learning Ops (MLOps) tackles the automation, provenance, performance, and other aspects of ML in a workflow-based form [107]. The *ZenML* Python library[15] provides a high-level API to machine learning tasks and tools, while offering workflow management features such as versioning, scheduling, and visualisation.

Markup or Domain-Specific Language. The other workflow specification commonly seen in workflow management systems is to have a definition written in either a markup language (such as XML YAML) or a custom DSL for the workflow itself.

Compi[16] is a framework to not only build and run workflows, but also deploy the workflows as command-line applications or containerised as Docker containers [85]. That is, once a domain expert has built a workflow, Compi packages the workflow and its dependencies can be easily shared to other domain experts to use as a command-line application. Compi uses the markup language XML to define the workflows as the creators López-Fernández *et al.* argue that a DSL for defining workflows is "less interoperable, being difficult to produce or consume from languages other than the one on which the DSL is based", and that XML is "easy to validate syntactically and semantically through schemas" [85]. A repository of Compi workflows is available through the Compi hub project[17] which aims to provide community exploration of the workflow, including automatic visualisation of the workflow tasks and links to sample input data [99].

Nextflow[18] is a workflow management system "designed specifically for bioinformaticians familiar with programming" [34]. Workflows are designed in a Bash script-like DSL to manage data flow between different workflow components. The Nextflow tool itself has support for obtaining and setting up Docker containers to allow for greater reproducibility of workflows.

Nextflow also has an active ecosystem providing validated open-source pipelines. In particular, the nf-core effort[19] is a community-maintained effort to develop "collaborative, peer-reviewed, best-practice analysis pipelines"[42]. Only one pipeline per data type/analysis is allowed, and all pipelines require quality checks such as a common structure, MIT licensing, continuous integration tests, linting, and appropriate documentation.

2.4.2 Graphical. With the rise of "low-code" platforms, there are an increasing number of *graphical* workflow systems available [14]. A prominent example of this is the domain of business applications, where providers such as outsystems[20] and Mendix[21] provide graphical interfaces to create applications which can involve ML.

A workflow system straddling the domain-specific and business domains is the Konstanz Information Miner (KNIME) [11]. From the University of Konstanz circa 2007, this framework originally focused on pharmaceutical applications[22] but has now scaled up for use within large-scale enterprises. KNIME is based on the Eclipse platform and offers a component library and canvas for drag-and-drop connection of nodes. KNIME also offers a repository for hundreds of components and workflows available for use with a selection of curated components available[23]. Also relevant to this

[15]https://zenml.io/
[16]http://sing-group.org/compi/
[17]https://sing-group.org/compihub/explore
[18]https://www.nextflow.io/
[19]https://nf-co.re/
[20]https://www.outsystems.com/
[21]https://www.mendix.com
[22]https://www.knime.com/knime-open-source-story
[23]https://hub.knime.com/

article is a feature within KNIME called the "Workflow Coach". From KNIME community usage statistics, this panel is able to recommend the next component to use in the workflow[24].

The Workflow Instance Generation and Selection tool (WINGS) [25] focuses on *semantic workflows*, where each input and component has semantic information attached [53]. This information is represented in the form of triples which allows for domain-specific information to be used to select workflow components. WINGS can use this semantic information to select components, input datasets, and parameter values [52].

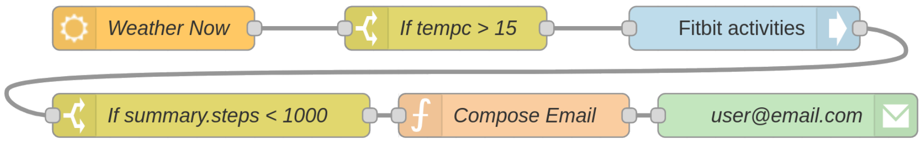


Fig. 1. Example Node-RED flow to send an email when the weather is nice and a step counter is low. Figure modified from [80].

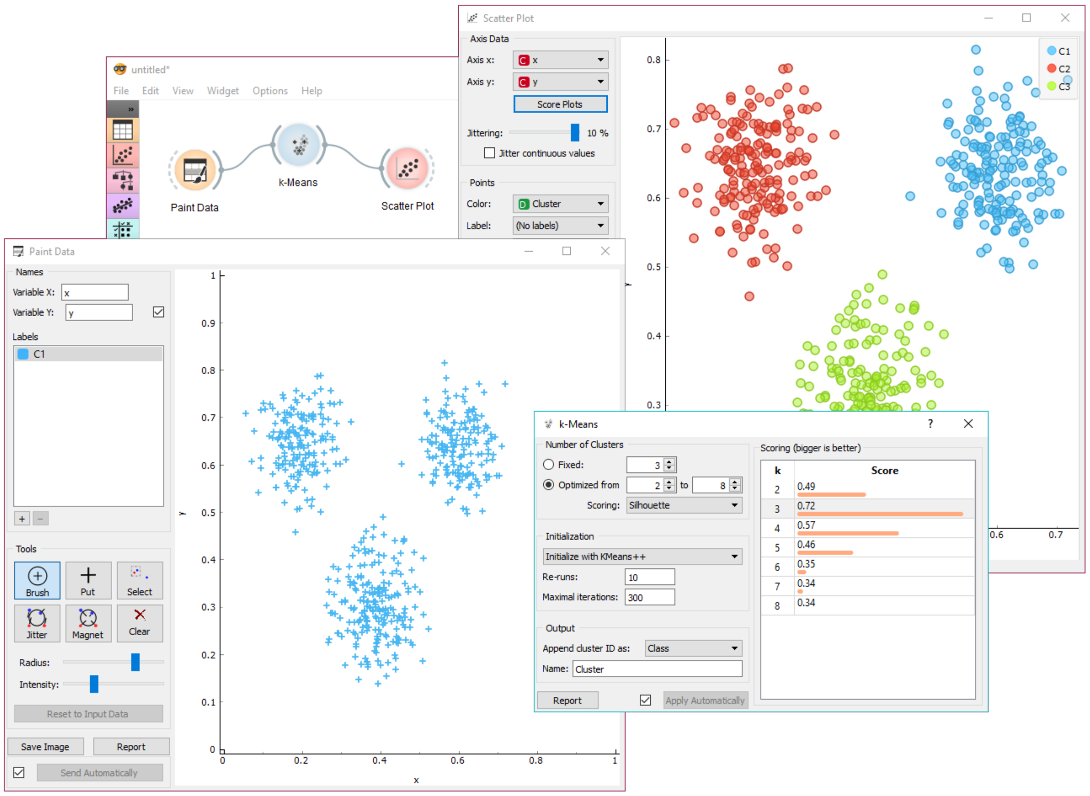
Node-RED. The *Node-RED* tool is a web-based editor for creating Internet of Things (IoT) workflows [25]. Nodes provide built-in functionality or can be customised by adding Javascript code. Figure 1 shows an example flow to remind a user to exercise. A node communicates with a weather service to obtain the temperature. If this value is above 15, then the flow communicates with an activity tracker. If this reports that a user's step count is low, then an email is sent. Flows can be deployed either to the user's local machine or many other embedded devices such as Raspberry PIs[26]. Nodes and workflows can be shared in an online repository[27], allowing users to enhance their workflows with new nodes[28] and sub-workflows.

Orange Tool. The *Orange* tool offers visual scripting of data mining techniques including machine learning operators and visualisation capabilities [31, 32]. Originating out of a bioinformatics research group, Orange focuses on providing an easy-to-learn data science tool. A canvas is provided for users to drag-and-drop nodes from a node library, where typed connections then aid users in assembling a workflow. Figure 2 shows an example workflow where data is visualised before clustering occurs with K-Means [102]. Note the option pane for K-Means allowing a user to tune the parameters. The clustering is then visualised.

Orange has two features which improve the reuse of the tool and its workflows. First is the selection of workflows available on the Orange website[29]. This offers 20 sample workflows for performing common data science tasks such as performing principal component analysis. The second feature is the robust add-on support which provides new nodes for workflows (which are called "widgets" in Orange). For example, currently there are add-ons for bioinformatics, education, and explainable AI available from within Orange itself. Users can also create their own nodes to create DS workflows [54, 123].

Galaxy. The *Galaxy* project is a web-based entire computational workbench for developing biomedical workflows [66]. It has spread to other fields as well with over 5000 publications citing Galaxy[30]. The workbench heavily focuses on

[24]See https://www.knime.com/blog/the-wisdom-of-the-knime-crowd-the-knime-workflow-coach
[25]https://www.wings-workflows.org/
[26]https://projects.raspberrypi.org/en/projects/getting-started-with-node-red/
[27]https://flows.nodered.org/
[28]For example, ML nodes: https://flows.nodered.org/node/node-red-contrib-machine-learning.
[29]https://orangedatamining.com/workflows/
[30]See https://galaxyproject.eu/citations for publications focused on just one online instance of the workbench.



Axis Data

Axis x: x

Axis y: y

Score Plots

Jittering: 10 %

Jitter continuous values

Points

Color: Cluster

Label: (No labels)

Names

Variable X: x

Variable Y: y

Labels

C1

Number of Clusters

Fixed: 3

Optimized from 2 to 8

Scoring: Silhouette

Initialization

Initialize with KMeans++

Re-runs: 10

Maximal iterations: 300

Output

Append cluster ID as: Class

Name: Cluster

Scoring (bigger is better)

k	Score
2	0.49
3	0.72
4	0.57
5	0.46
6	0.35
7	0.34
8	0.34

Fig. 2. Example Orange workflow to visualise data, cluster it with K-Means, and then show the clustering [102].

concerns such as reproducibility, provenance of data and tools, and sharing of workflows and learnings. Figure 3 shows a few components from a workflow for detecting urothelial (bladder) cancer, as defined in the Galaxy workflow management tool [66].

An interesting aspect of the Galaxy project is a focus on community and specialisation. For example, there is a main publicly-available Galaxy instance online[31]. However, to spread out computation costs and offer the possibility of specialising the datasets and tools available, Galaxy can run on a user's local or cloud machine. Furthermore, these other Galaxy instances can be customised into different *workspaces*.

We highlight three recent DS specialisations of Galaxy which are publicly available online. Vandenbrouck *et al.* developed a Galaxy instance for proteomics research [125], Tekman *et al.* provide one for "single cell omics" [120], and Gu *et al.* offer a ML focus [58]. These specialisations each offer targeted DS tools, workflows, and computational resources, allowing domain experts to quickly develop workflows.

3 OVERVIEW OF OUR FRAMEWORK

This section will provide an overview of our conceptual framework by discussing its structure, two dimensions, and relating the framework to our challenge questions introduced in Section 1.

[31] https://usegalaxy.org/

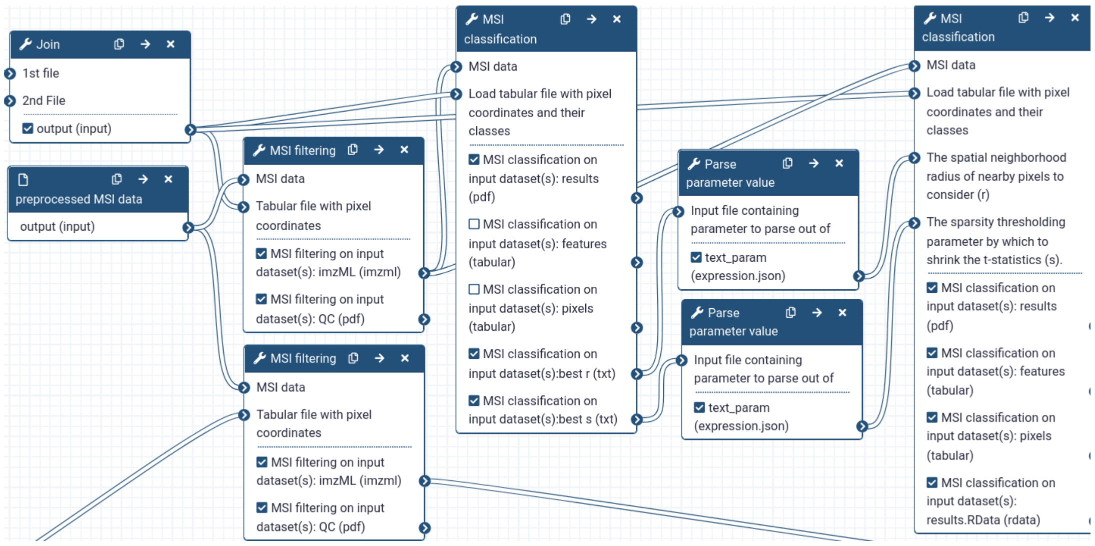

Join
1st file
2nd File
output (input)
preprocessed MSI data
output (input)
MSI filtering
MSI data
Tabular file with pixel coordinates
MSI filtering on input dataset(s): imzML (imzml)
MSI filtering on input dataset(s): QC (pdf)
MSI filtering
MSI data
Tabular file with pixel coordinates
MSI filtering on input dataset(s): imzML (imzml)
MSI filtering on input dataset(s): QC (pdf)
MSI classification
MSI data
Load tabular file with pixel coordinates and their classes
MSI classification on input dataset(s): results (pdf)
MSI classification on input dataset(s): features (tabular)
MSI classification on input dataset(s): pixels (tabular)
MSI classification on input dataset(s):best r (txt)
MSI classification on input dataset(s):best s (txt)
Parse parameter value
Input file containing parameter to parse out of
text_param (expression.json)
Parse parameter value
Input file containing parameter to parse out of
text_param (expression.json)
MSI classification
MSI data
Load tabular file with pixel coordinates and their classes
The spatial neighborhood radius of nearby pixels to consider (r)
The sparsity thresholding parameter by which to shrink the t-statistics (s).
MSI classification on input dataset(s): results (pdf)
MSI classification on input dataset(s): features (tabular)
MSI classification on input dataset(s): pixels (tabular)
MSI classification on input dataset(s): results.RData (rdata)

Fig. 3. A selection of a workflow described in [47], in the workflow management tool Galaxy [66].

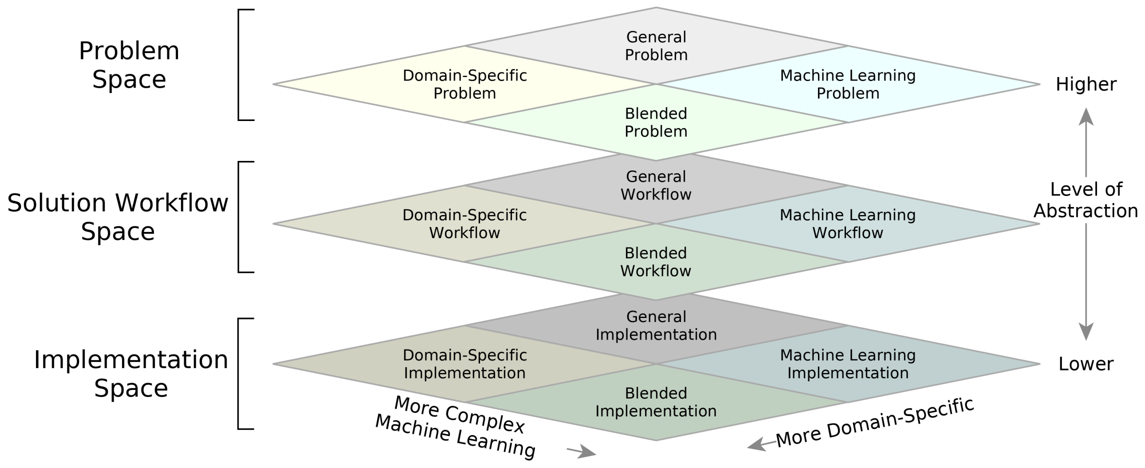


Fig. 4. Our three-layer framework with the *problem*, *solution workflow*, and *implementation* spaces. Each layer is divided into four regions along the *domain-specific* and *complexity of machine learning* dimensions.

As a summary, our conceptual framework maps out the options for a domain expert to choose from such that they can develop a workflow-based solution to their domain-specific problem using machine learning, and ultimately obtain a usable implementation. The framework is seen in Figure 4, separated into twelve regions through the division of three layers and two dimensions. The three horizontal layers define the *problem space*, *workflow solution space*, and the *implementation space*.

The *problem space* layer contains the *problem* of the domain expert. For example, a medical researcher may wish to classify whether urethelial (bladder) tissue is cancerous or not, as in Section 6.3.3. The *solution workflow space* layer contains workflows which solve the problems from the upper layer. For example, Figure 3 shows a selection of components which address this problem in the Galaxy workflow tool. The *implementation space* contains the implementation details for the workflows in the middle layer. In this Galaxy example, this would be the code as executed on the local or cloud machines. Section 4 addresses these layers in detail.

Each layer is decomposed into four regions, defined by the *domain-specific* (DS) and *machine learning* (ML) dimensions. Section 4 also discusses how problems, workflows, and implementations can be mapped or *transformed* between the regions in a layer. Section 5 will then discuss the mappings and transformations *between* layers. That is, how a problem can be mapped or transformed into a workflow, and how a workflow can be mapped or transformed into an implementation.

Note that the term 'transformation' is used in this article in a broad sense, as in model-driven engineering [113]. That is, transformations have as input an artefact from one region and produce an artefact from another region. For example, a transformation on the problem layer could map a problem in the *DS problem space* to the *ML problem space*, representing techniques such as an *expert mapping* [61]. Another transformation would be a mapping between a problem statement and a workflow which can be used to address that problem. These transformations can be either manually-performed or automatic, and use code or pattern approaches. Their important feature is that they affect artefacts through the additional, removal, or modification of elements.

3.1 Dimensions of the Space

This framework defines two dimensions for each layer. First is the notion of *domain specificity* which captures how many DS concepts an artefact contains, and how "familiar" the artefact is to experts in that domain. Our hypothesis is that artefacts with more domain-specific concepts should require lower cognitive complexity for the domain expert to specify and reason about, as reported by Voelter *et al.* [127].

Second, the *complexity of ML dimension* relates to both the amount and cognitive complexity of the ML concepts or algorithms the artefact contains. This ML dimension aims to capture the *specialization* and *prior knowledge* required to specify and correctly configure the ML components. Our hypothesis is that tailoring the ML complexity to the level of understanding of the domain expert will aid their modelling task [95], and avoid performance losses from misconfiguration [137].

For example, consider an artefact from the first layer: a *problem* to be solved. This problem could be very generic such as "*classifying an image into two categories*". Adding *domain specificity* comes from adding domain knowledge to the problem, such as the knowledge that *the image is from an MRI scan of a brain, and the classification is on whether the patient is depressed or not* (see Section 6.3.1). This extra information can also have an impact on the solutions available for the problem, such as requiring particular workflow components to process the MRI images.

The problem can also vary in how *complex with regards to ML* it is. For example, a domain expert unfamiliar with ML may simply wish to classify depressed patients using a black-box ML algorithm. As the expert gains more familiarity with ML techniques, they may desire to search the space of possible solutions. Thus, a more complex ML problem would be the comparison of multiple ML algorithms or architectures, performing parameter optimization of the components, or developing an ML agent using reinforcement learning techniques.

In the *problem space* layer, the problems are categorised based on the DS or ML concepts present. For the *solution workflow* layer, this categorisation depends on the proportion of components and their configuration complexity in the workflow, such as the number of components for processing MRI images. In the *implementation space* layer, this categorisation depends on the use of DS or ML APIs or libraries. Section 4 further discusses how the dimensions divide up each layer in the framework.

Note that by necessity, these categorisations are *fuzzy* and we intentionally do not provide boundaries to these regions or spectrums along the dimensions. Instead, we provide these dimensions to provoke reflection about the *mappings* and *transformations* along these dimensions. That is, what does it mean to make a problem, workflow, or implementation *more domain-specific* or involve *more complex machine learning*, and what are the techniques to do so?

Let us also note that these two dimensions are certainly not orthogonal. Specifically, *machine learning* is itself a domain of interest and these two dimensions may overlap significantly depending on the problem of interest. Therefore, let us state here that when this article refers to *domains* or *domain experts*, we refer to a non-machine learning domain.

3.2 Relation to Challenge Questions

As expressed in our framework, the domain expert wishes to transform their problem from their domain (the *domain-specific problem* region on the top layer in Figure 4) all the way into an implementation using machine learning libraries (the *blended* or *machine learning implementation* regions on the bottom layer). Through these transformations, the domain expert is able to move around through different regions as shown by the example studies in Section 6, though support may be lacking for some operations in various tools as discussed in Section 7.2. In particular, we are interested in determining tool support for transformations enabling the most direct route from the *domain-specific problem* to a *blended workflow* down to a *blended implementation*.

Here we recall the challenge questions from Section 1 and relate them to specific transformations.

Mapping a DS problem to a form suitable for ML. This challenge refers to how domain-specific problems in the *problem space* can be mapped or transformed to include more ML concepts. That is, moving the problems along the *ML dimension* in the *problem space*.

Providing a solution workflow for a DS and/or ML problem. This challenge refers to the mapping between a problem in the *problem space* and a workflow in the *solution workflow space*. That is, moving from the top to the middle layer in the workflow.

Allowing the domain expert to experiment with appropriate ML components in a DS workflow. This challenge refers to moving solution workflows on the middle layer along the ML dimension through the integration of more ML components.

Adding DS knowledge to improve ML performance. This challenge refers to moving solution workflows on the middle layer along the DS dimension by adding new DS information or components.

Producing an implementation from a workflow which is well-suited for a domain expert. This transformation is between the middle and bottom layers, as the workflow of the domain expert is mapped or transformed to an executable implementation.

Extracting a workflow from an existing implementation (code or notebook). This challenge is an inverse to the previous one, as the implementation on the bottom layer of the framework is instead transformed into a solution workflow on the middle layer.

Challenge Selection. These challenges therefore directly correspond to some of the intra- and inter- layer transformations addressed next in this article. We believe these transformations are most relevant to the tools and example studies discussed in this article, as they enable the domain expert to move from the domain-specific problem to the final implementation.

There are a number of transformations which we do not focus on as challenges. For example, it could be possible to *make a problem, workflow, or implementation more general* through transformations. While this may be appropriate for generalizability, this goes against our focus of a domain expert utilizing machine learning. Other transformations not highlighted as a core challenge include the *modification of the implementation layer in either dimension*, as we believe this to be not relevant from the domain expert's point of view. Finally, we have not highlighted the transformation from the *workflow layer to the problem layer*. Conceptually, this transformation would take the workflow and extract the problem it solves, such as in [35]. This process would thus be performed by workflow repository contributors, and not the domain expert wishing to find a workflow for use.

4 LAYERS AND INTRA-LAYER TRANSFORMATIONS

This section details the three layers of our framework as shown in Figure 3: the *problem space*, the *solution workflow space*, and the *implementation space*. The *regions* and relevant *transformations* within each layer are then presented.

4.1 Problem Space

The *problem space* is where the problem of the domain expert is formulated. For example, consider the tissue-drug problem presented in Section 1, which can be expressed in two ways. In the *domain-specific* space, the problem is to predict whether a drug will work well with a sample of tissue [61]. As a *machine learning* representation, this problem becomes a numerical prediction for efficacy of the graph structure (for the drug) versus a linear string of characters (for the genes present in the tissue). A *blended* version of this problem is visualised in Figure 5 where the structures for the drug and gene are fed into ML encoders and used to make a numerical prediction related to how the drug affects the gene.

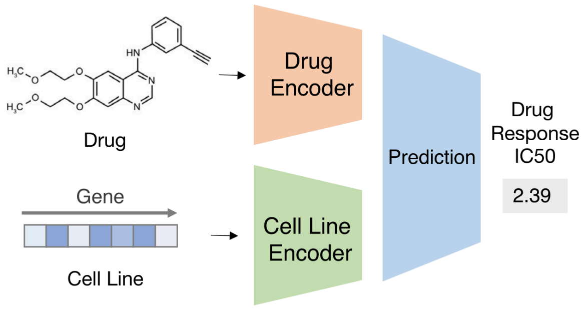


Fig. 5. The *Drug Response Prediction* problem from Huang *et al.* [61].

4.1.1 Artefact Representation.

The artefact for this layer is a *problem* composed of *concepts*. This problem may be specified in multiple ways ranging from a simple informal statement to a complex formal representation. For example, the problem could be specified as natural language research questions, in an ontological manner, in a textual or graphical domain-specific language (DSL), or in a template-based, requirement-like manner.

4.1.2 Layer Regions. In our framework, we define four *regions* for the problem space: *General Problem*, *Domain-Specific Problem*, *Machine Learning Problem*, and *Blended Problem* as shown in Figures 4 and 6. These regions are defined by the *domain-specificity* (DS) and *machine learning* (ML) dimensions as discussed in Section 3.1.

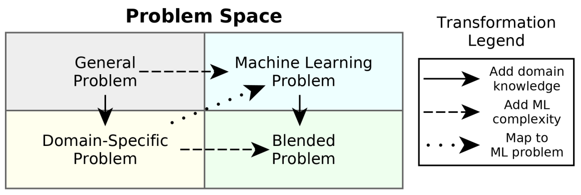


Fig. 6. A representation of the problem space, with intra-layer transformations (Section 4.1.3).

A *general problem* is one which does not refer to any DS concepts, nor does it refer to any ML concepts. This region is provided for completeness as this article focuses on the other three regions.

Domain-specific problems are those that are specified in the domain of interest by experts. They involve concepts which are specialised to that domain. In this article, the DS problem is the starting point from which the expert wishes to solve with a workflow and finally an executable implementation.

A *machine learning problem* is a problem which involves a higher proportion of complex ML concepts and few DS concepts. For example, learning how to classify objects by ingesting data from a table involves more ML concepts than DS concepts. As mentioned in Section 3.1, a complex ML problem could involve the comparison of multiple ML classification algorithms. An even more complex ML problem may also involve the domain expert adding advanced components (such as deep learning, convolutional layers, and transformers), or explicitly configuring the ML components (optimising meta-parameters for training, tuning layer sizes). We would consider these advanced problems are lying "farther" along the ML complexity dimension. This is because these components and configuration would require extra cognitive effort and expertise from the domain expert to correctly utilize.

A *blended problem* is one that contains both DS and ML concepts such as problem represented in Figure 5. This type of problem may be ideal for the domain expert to begin with instead of a DS one, as the ML concepts can be directly conceptualised and/or operationalised in the workflow layer. However, it already implies that the practitioners has access to both domain and machine learning knowledge, and has studied the problem from both domains.

4.1.3 Layer Transformations. As discussed in Section 3.1, it is interesting to reason about how to transform an artefact on each layer to make it more DS or involve more complex ML components. In other words, to make these problems *more DS/ML specific and less general*.

To *increase DS*, a domain expert will have to encode more domain knowledge into the problem. This could be in the form of an ontology representing formal knowledge, or by directly specifying features of interest in the domain and their interaction with other features. For example, consider the example of classifying MRI images for whether they are from a depressed patient or not (Section 6.3.1. These images represent the structure of a patient's brain, and thus the problem may include concepts related to brain structure and function. For example, the problem could shift from general concepts ("*Is this image a depressed brain or not?*") into a more domain-specific one ("*Which are the brain regions active/not active in depressed brains? What is the correlation between activated regions in a depressed brain?*", etc.).

Increasing the *proportion and complexity of ML concepts* is similar. A ML domain expert will identify structures and techniques from the ML domain to cast the problem into. This may be to specify the technique(s) used for classification or learning, or the representation that the problem should take. This is the transformation which represents our challenge question: *mapping a DS problem to a form suitable for ML*.

Direct Mapping. Transformations are also possible directly between the domain-specific and machine learning problem regions. This transformation is how to take a problem which exists in the domain and map it to a problem in the machine learning space, or to create a many-to-many mapping. This opens up further possibilities for insights and implementation.

From a brief investigation of the literature, we have identified three possible techniques for mapping: *expert mapping*, *ontological reasoning*, and *data-driven approaches*. The first technique represents the manual approach, while the second and third techniques represent (semi-) automated approaches based on symbolic and data-driven reasoning.

In *expert mapping*, a group of experts from both the domain of interest and the machine learning domain issue recommendations about how to map the problem. This is shown by Huang *et al.* where problems in genomics are mapped to machine learning representations [61]. This is a high effort technique, but if available, will greatly assist with workflow and implementation creation as experts will understand how their problem directly maps to an ML solution.

This expert mapping is of course possible in other domains. Aneja *et al.* provide a table mapping the problems addressed in the neuro-oncology literature with the ML approach used in those papers [6]. Jablonka *et al.* provide another detailed review with mention of how problems in material science were mapped to ML [64]. As explained in Section 6.2.3, the Kaggle competition website focuses on providing ML solutions for problems. However, this mapping knowledge is not presented explicitly and must be recovered [118].

Another technique we have identified is where *ontological reasoning* is performed to (semi-) automatically assist the domain expert in obtaining an ML or blended problem. In this technique, the expert's domain and the ML domain would be explicitly modelled, and then a reasoner would suggest appropriate ML structures and techniques to apply to the problem.

Existing work suggests particular data cleaning steps or ML workflow requirements based on ontological reasoning about the characteristics of the data set or user input [51, 53, 117]. Such an approach could be used to suggest the most appropriate ML algorithms or architectures based on the expert's problem [78]. For example, if the domain expert wishes to perform classification on MRI images, a reasoner could operate over medical and ML ontologies (and relationships between) to suggest that both Convolutional Neural Networks and/or Support Vector Machines may be appropriate for this solution. This then provides a starting point for the domain expert to acquire more knowledge or begin building their workflow solution.

The third technique is to consider *data-driven approaches*, where previous examples can be generalized to provide an answer for the domain expert. This category includes mining of previous data to provide a mapping. Another emerging approach could involve querying a Large Language Model (LLM) to map a problem statement in natural text or a formal representation to another layer region.

Reversing the Transformation. In principle, these transformations could be reversed. That is, a problem could be made *more general* with these techniques, and a *machine learning* problem could be mapped to a *domain-specific* one. Improving the generalisation and mapping possibilities for problems could help domain or ML experts understand how these regions interact and how ML problems correspond to DS problems.

4.2 Solution Workflow Space

In the second layer of our framework, we define a space of *solution workflows*. These workflows detail the actions required to solve the problem defined on the upper layer.

4.2.1 Artefact Representation. The representation of these workflows is based on those described in Section 2.3 with some available standards presented in Section 2.3.1. The core aspect is that these workflows consist of a directed sequence of components with explicit control and/or data flow. These components represent a step or action in the workflow, and they can be typed to enforce structure on the accepted inputs and outputs and define their semantics.

The idea with this representation is to have a unified, graph-like structure amenable for modularity, re-use, modification, and sharing. As mentioned in Section 2.3, workflows are already common in scientific domains and structured "pipelines" are in place in the data science/machine learning world. Therefore our conceptual framework solely focuses on graph-like workflows as the representation for this middle layer.

4.2.2 Layer Regions. In this space, we again roughly classify workflows into four regions as shown in Figure 7.

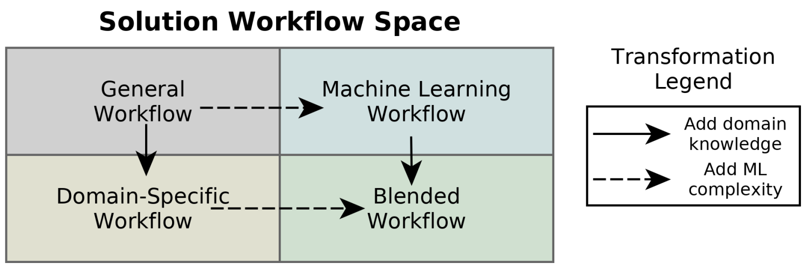


Fig. 7. A representation of the solution workflow space, with intra-layer transformations.

A *general workflow* is one where there are very few domain-specific or machine learning components within the workflow. Thus it is a catch-all category comprising those workflows not discussed below.

In our framework, a *domain-specific workflow* is one that has many components from the domain of interest. These components must address a domain-specific concept which is not of interest to a general user outside of that domain. Examples include loading a file or database with a domain-accepted structure, a computation performing a specific task of interest to the domain, or communication with domain entities such as robotic arms.

Another interesting category of components are those that provide *domain-specific visualizations*. For example, one of the case studies in this article concerns classifying the presence of urothelial (bladder) cancer based on *mass spectrometry imaging* (Section 6.3.3). The set of workflows developed by the authors includes components tailored to generate heat-map plots out of the imaging data to visualize the classification of areas of the bladder tissue, and to understand the spectrum information that leads to a classification result [47]. Thus, these visualization components are *domain-specific* as they are only relevant for this particular imaging technique. Furthermore, the domain expert should be assisted to include these components in their workflow, as this will ease workflow development and comprehension of the results.

As in Section 3.1, we also consider a rough spectrum where some workflows have more domain-specific components than others. For example, one workflow may simply load data from an MRI file and visualise it, while another may perform filtering or spectrum analysis on the data first. This second workflow is thus more domain-specific than the first.

Similar to our category of domain-specific workflows, a *machine learning workflow* is one that contains complex ML components. Again, we state that there is a rough spectrum of these workflows from not including ML components at all to heavily relying on complex components. An example for such an ML workflow is found in one case study (Section 6.3.2) which contains multiple components for classifying and scoring. Here, the workflow treats the input data without considering the domain, and the focus is on configuring and utilizing the ML components. Thus the user is mainly reasoning about ML concepts when examining this workflow, leading to its classification as a *ML workflow*.

The *blended workflow* is one that contains numerous DS and ML components. As well, a component itself may be labelled as *blended*, as it may address a DS concern but heavily utilise ML "within" the execution of the component. We propose that this blended type of workflow is desired for a domain expert to arrive at, since the presence of these components should raise the level of abstraction and increase the modularity and reuse of the workflow. However, it may be preferable for a domain expert not familiar with ML to begin working with a DS workflow, and have the workflow 'adjusted' to become more blended over time as they gain familiarity with ML concepts and components. This offers the domain expert further experimentation, customisation, and optimisation possibilities.

4.2.3 Layer Transformations. We have posed two interesting questions in the introduction of this article of how to *transform* a workflow along these two dimensions. That is, to identify the techniques to: a) increase the *domain-specificity*, and/or b) *complexity of machine learning* in a workflow. These correspond to our challenge questions of: a) *adding DS knowledge to improve ML performance*, and b) *experimenting with ML tools and techniques within a workflow*.

A first approach is to ask a domain or machine learning expert to study the workflow and identify where components should be added or improved. Their knowledge could then be modelled and implemented for automated approaches to detect and suggest components for use, such as that implemented by Kumar *et al.* [74]. These recommender systems could therefore shift the workflow along the *domain-specific* or *machine learning* dimensions. We also note that it may be possible to train and query a Large Language Model (LLM) to offer suitable recommendations [9, 134].

As an example of improving a workflow, consider one that loads genomic data from a table provided in a spreadsheet for further processing. Depending on the requirements of the user, the loading components may be better replaced with a component which is able to download up-to-date genomic data directly from a cloud repository. As mentioned in Section 6.3.3, these domain-specific operations are available in tools like Galaxy.

Domain-specific workflows can also be utilised as a sub-workflow. For example, Sections 2.4.1 and 6.3.1 discuss the *fMRIPrep* workflow which is for performing specific pre-processing for neuroscience data. Thus, a domain expert's workflow becomes more DS when fMRIPrep is used.

The error of ML techniques may also be lowered when DS information is used. One aspect of this is the field of *feature engineering* where new data is extracted from the old to reduce the error of machine learning algorithms such as deep learning [15]. For example, Fan *et al.* study the problem of marking reported software bugs as 'valid' or 'invalid' [45]. From the bug data, they extract new features such as *recent number of bugs by reporter*, *does the bug have a code patch attached*, and *bug text readability scores*.

Reversing the Transformation. Just as with the layer transformations for the problem space (Section 4.1.3), these transformations could indeed be reversed to increase the generality of the workflows. That is, to make a workflow *less* DS and involve *less complex or fewer* ML components. This may help to increase the applicability of the workflow across domains, however we do not consider it further.

4.3 Implementation Space

Similar to the above layers, the *implementation space* is also a rough categorisation of possibilities. In our conceptual framework, the implementation space defines the low-level result which runs on a computational device, such as produced code or the execution of a workflow engine. For example, some workflow management tools are directly implemented in Python, or the Orange editor (Section 2.4.2) can directly execute the workflow inside the tool.

4.3.1 Artefact Representation. In this space, the artefacts are represented as code or another machine executable format. This may be Python code which contains calls to ML APIs, or may be the machine code executed by a workflow engine which is directly interpreting and executing the instructions within a workflow.

These two examples are not at the same level of abstraction. However, the intention with this representation is that: a) this code directly calls upon DS and ML APIs, and b) this code should not be written directly by a domain expert as the level of abstraction is too low.

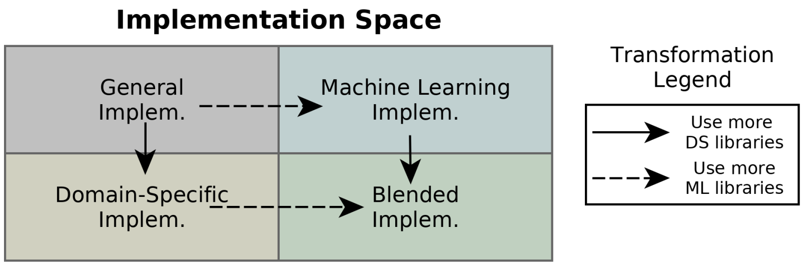


Fig. 8. A representation of the implementation space, with intra-layer transformations.

4.3.2 Layer Regions. Similar to the above layers, the implementation space layer is defined by the same two dimensions. The *domain-specific* dimension defines the proportion of the code which calls upon DS APIs or libraries. The *machine learning* dimension defines the proportion for ML APIs or libraries.

A *general implementation* contains a small proportion of calls to DS or ML APIs or libraries. Thus it is general code.

In a *domain-specific implementation* the code makes calls into an API or library which provides DS computation. For example, the nipype library (Section 2.4) offers a neuroscience-specific Python library. Thus the more calls to libraries like these, the more DS the implementation. Likewise, a *machine learning* implementation has a high proportion of ML API or library calls. An example would be directly calling Tensorflow or other ML library from Python.

A *blended implementation* is one which uses both *domain-specific* and *machine learning* APIs and libraries. Thus the ML libraries are wrapped in a DS interface, and potentially optimised for each DS task. Clearly it would be desirable for the domain expert to obtain this form of implementation to achieve the most specific implementation. However, the domain expert should not write the implementation by hand, and preferably they can instead generate an implementation from their workflow. This is discussed in Section 5.2.

4.3.3 Layer Transformations. Again, the same transformations as the solution workflow layer occur in this layer. Code can be mapped manually or automatically to either a DS library call or a ML one. These transformations could be useful for addressing legacy code and updating it (called "cognification"). However, as argued in Section 7, it is not be efficient to focus on mapping and recommendations at the implementation level. Instead, a more efficient approach would be to extract the workflow from the code and apply analyses at the workflow level.

5 INTER-LAYER TRANSFORMATIONS

This section defines the possible *transformations* used to transform an artefact in one of the layers in our framework to another layer. The transformations discussed here are: a) from the *problem space* to the *solution workflow space*, and b) from the *solution workflow space* to the *implementation space*. Note that transforming a problem from the *problem space* to the *implementation space* is *classical programming*, which we do not elaborate further in this section but appears in example studies in Section 6.

5.1 Problem Space to Solution Workflow Space Transformations

The transformation between the first two layers of our framework transforms *problems* from the *problem space* into *workflows* in the *workflow space*. This transformation corresponds to our challenge question of *obtaining a solution workflow for a DS and/or ML problem*. Practically, this transformation is most likely a combination of a domain expert *building* and/or *finding* workflows and workflow components.

That is, a domain expert could: a) find an existing workflow or components in a repository, b) rely on formal or informal mappings or recommendations to assemble a workflow, or c) build the workflow themselves either from a component library.

Here we will summarise a few of the techniques available in the tools from Section 2.4 to assist a domain expert in obtaining or building a workflow.

Component Libraries. Many of the graphical workflow tools (see Section 2.4.2) use a library of components for the domain expert to select from when building their workflow. Plugins or extensions can extend this component palette with domain-specific components, allowing for easier selection of these components. For example, the Galaxy tool offers workflow components which directly obtain data from biology databases. This aids biologists to efficiently obtain up-to-date data directly into their workflows.

Domain-specific Tutorials and Sample Workflows. The documentation surrounding a workflow tool often provides numerous examples for using the tool and for solving real-world problems. For example, the Nipype tool discussed in Section 2.4.1 offers over 30 neuroscience-specific examples on its website. This allows users to get started quickly to solve their domain-specific problems.

Workflow Repositories. Some of the workflow tools discussed in Section 2.4 have explicit *repositories* for searching and obtaining workflows, or have multiple tutorial workflows for demonstrating the use of their components. These repositories allow for experts to select whole workflows and their component parts for use in solving their problem.

However, a brief glance at these repositories suggests usability issues. For example, one Galaxy workflow repository[32] contains hundreds of workflows, yet the only search options available cover *free text*, *user rating*, and *keyword search*. This may make it quite difficult for a domain expert to find a suitable workflow unless techniques are used to further recover semantic information [35]. Reproducibility issues may also hamper this search as computations may not be consistent between runs or user machines [121].

[32] https://usegalaxy.eu/workflows/list_published

Automated Recommendations. Recent work by Wen *et al.* suggests that it may be possible to automatically determine similarity of workflows in repositories [130]. This could allow for enhanced discoverability of workflows.

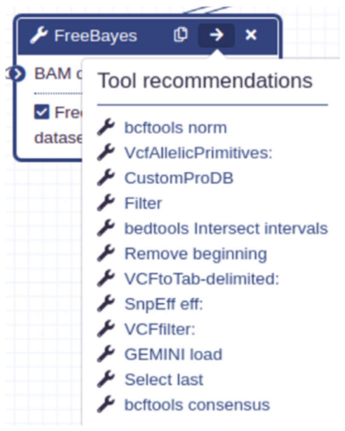


Fig. 9. Automated recommendation of a component. Modified from [74].

Workflow tools can also recommend next components to be placed automatically. This allows for domain experts to be assisted by the tool to build their workflow. For example, the excellent article of Kumar *et al.* presents a recommendation engine for the Galaxy framework [74]. This engine integrates into Galaxy itself to provide component recommendations based on the existing components and the recent usage of that component in the data set. Figure 9 shows the tool providing a list of recommended next components for the user to select from.

Automated Creation. An interesting technique to create the whole workflow at once is to use machine learning techniques themselves to create workflows. This is the field of AutoML [60], which uses accuracy metrics to create a machine learning pipeline for the domain expert's data. A partial or full workflow for the expert can also be provided based on their problem and/or their data [52, 53]. As a recent example of this AutoML approach, we point to Dunn *et al.* who use a benchmarking set in materials science as the basis for creating material science-specific workflows [37]. Kasalica and Lamprecht also discuss the APE automatic workflow composition tool to build workflows based on ontological knowledge [69].

Large Language Models (LLMs) may also be suitable for providing recommendations for workflow components to modify, or the creation of whole workflows at once [134].

5.1.1 Region Transformation. Figure 10 diagrams a selection of the possible transformations between regions on the top *problem space* layer into the middle *solution workflow layer*. For example, a *domain-specific problem* could be solved with a *blended workflow* created through these transformations.

A solid arrow in the figure represents the transformations between the problem layer and the solution workflow layer, as described above. This includes providing the user with access to workflow repositories, component libraries, and any other support such that the user can build their workflow. The dashed arrows indicate where domain-specific component libraries or tutorials can assist in making the solution workflow more domain-specific, alongside the other transformations for building a workflow.

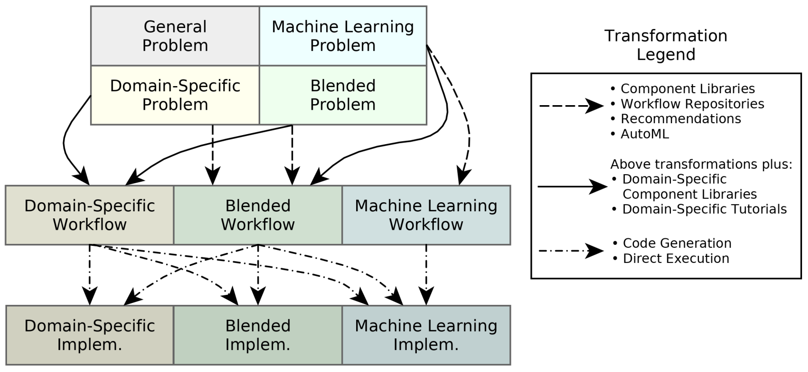


Fig. 10. Transformations between the *problem space*, *solution workflow space*, and the *implementation space*.

Note that transformations from the general problem are not represented as this is not the focus of this article. Also not represented are the unlikely (yet possible) transformations from an ML problem to a DS workflow, or from a DS problem to an ML workflow.

5.1.2 Reversing the Transformation. As mentioned above, it can be challenging for domain experts to search a workflow repository and find suitable workflows for their problem. One direction to address this issue is to automatically 'mine' the *workflow* itself and extract the *problem* that workflow solves, or at least extract some tags and other semantic information [35].

5.2 Solution Workflow Space to Implementation Space Transformations

The transformations between the middle and bottom layers of the framework (see Figure 10) transform a *workflow* in the *solution workflow space* into some form of *code* in the *implementation space* (Section 4.3). This corresponds to our challenge question of *producing an implementation from a workflow which is usable for a domain expert.*

The techniques examined here are: a) re-implement the workflow manually (not discussed below), b) code generation / Model-Driven Engineering (MDE) techniques, or c) workflow execution directly performed by the workflow tool.

Note that in Figure 10 the transformations are between various regions. This reflects that there may be a benefit to produce an implementation which relies on domain-specific libraries, on machine learning libraries, or a blended implementation. For example, Zhou *et al.* describe a PyTorch-based framework for performing molecular dynamics simulation with GPU acceleration [139]. This demonstrates that the implementation for a workflow may need to be tailored based on the workflow domain.

5.2.1 Model-Driven Engineering Techniques. From a workflow defined in a DSL or as components, it can be a straightforward process to perform MDE techniques such as code generation [7]. Due to the modular nature of the workflows, executable code could be generated for each component. This code generation may also take place over a number of intermediate languages, such as using workflow middleware to handle concerns such as scalability (Section 2.4).

5.2.2 Direct Execution. Another way of executing the workflow is to run it inside the workflow tool itself. This relieves the domain expert from running the final code themselves, though it may be more difficult to optimise if needed. Many of the tools in Section 2.4 execute the workflow in this way, either through the host language such as Python or within the tool itself such as Galaxy. This execution may also be local on the domain expert's machine or run on another machine.

5.2.3 Reversing the Transformation. An interesting transformation available is to start with a legacy piece of code and extract the workflow from it as in our challenge question: *extract a workflow from an existing implementation.* This could be a manual inspection or an automatic process. Once completed, the workflow could then become the source of truth and could be moved into a workflow repository for further dissemination and development [20].

6 EXAMPLE STUDIES

This section examines the use of the tools from Section 2.4 in various domains. Example studies selected from the scientific literature provide a sample of how experts in each domain are building domain-specific workflows utilising machine learning. This section thus serves to: a) ground our framework in the state-of-the-practice, and b) highlight research challenges and opportunities where the software engineering community can assist domain experts. In Section 7.4, we will discuss an illustrative study to investigate how a domain expert could be assisted by tools aligned to our framework to easily obtain an implementation for their problem.

As a caveat, these example studies have been selected in an *ad-hoc* and non-systematic manner. Instead, the informal criteria was based on *recent publication*, *available artefacts*, and *variety of domain.* The intention is to provide a flavour of the heterogeneity of the domains and the recent use of tools for reflection about expert assistance, not an extensive literature survey.

We focus on three sets of example studies in this section. The first set is where the study has an *implicit* workflow. That is, there is no explicit workflow graph in one of the standards or tools reported in Section 2.4, and the workflow is expressed in code. The second set of example studies contains *explicit* workflow artefacts, where the workflow is explicitly defined using a workflow standard/tool. The third set is one study where the high-level workflow itself is implicit, but it relies on a sub-workflow defined using a workflow framework.

6.1 Example Study Overview

Table 1 lists the example studies examined in this report and discussed throughout this section. Each study is provided an identifier based on the primary tool used and a short description. The second column in Table 1 denotes whether the study has an *implicit*, *explicit*, or *hybrid* workflow, and the third column lists whether the workflow is created *textually* or *graphically.* The last column reports the regions through our framework that the study has artefacts in. That is, what "path" the authors took through our framework from Section 3. This can be *ML*, *DS*, or *Blended.* For example, the ES4-nipype study is classified as a *blended* problem, and *domain-specific* for both the solution workflow and implementation. The Kaggle study also shows that a blended workflow can have a strong lean towards one dimension. In this case the lean is towards the ML dimension, represented by *BlendedML.*

For each study, the domain-specific problem is described. Then, the regions and transformations from our framework (Section 3) which are relevant are presented.

6.2 Implicit Workflow Studies

The first four example studies presented focus on *implicit* workflows where there is not a workflow explicitly defined in one of the standards or tools from Section 2.4. These studies presented therefore cover cases where the domain

			Workflow	Regions		
Label	**Description**	**Repres.**	**Expression**	**Problem**	**Workflow**	**Implem.**
ES1-PyTorch	Detecting peaks in *metabolomic* data	Textual	Implicit	Blended	Blended	Blended
ES2-MATLAB	Classifying rock origin based on molecular structure (*geochemistry*)	Textual	Implicit	Blended	Blended	Blended
ES3-Kaggle	Crowd-sourcing ML solution to *marine biology* challenge	Textual	Implicit	ML	BlendedML	BlendedML
ES4-nipype	Detecting depression from MRI images (*neuroscience*)	Textual	Hybrid	Blended	DS	DS
ES5-Orange	*Data mining analysis* for smart school Internet traffic.	Graphical	Explicit	ML	ML	ML
ES6-Galaxy	Classification of urothelial cancer (*bioinformatics*)	Graphical	Explicit	DS	DS	DS

Table 1. Summary of the example studies.

expert directly writes an implementation of their problem, skipping the workflow layer of our framework (Section 3). A discussion of these implicit versus explicit workflows is found in Section 7.1.1.

6.2.1 ES1-PyTorch. The first example study we examine represents the situation where a domain expert encodes their problem directly upon a ML library such as PyTorch or sklearn.

As a Suggested Practice. Directly writing code on an ML library is at a low-level of abstraction requiring a great deal of ML knowledge. However, we have found two recent publications where this approach is suggested.

For chemistry students, Lafluente *et al.* present an introductory workshop focusing on utilising Python and visualisation/ML libraries [75]. The example Jupyter notebooks [33] lead students through an introduction to Python, basic statistics, exploratory data analysis, classification, and regression.

In the field of materials science, Wang *et al.* suggest that utilising Python code and the PyTorch library is considered 'best practice' [128]. The example Jupyter notebooks[34] walk a domain expert through an example application to highlight ML techniques and considerations at a very granular level of detail. For example, the reader is taken through constructing the layers of a neural network in PyTorch, along with calling the prediction/back-propagation functions. While this work is very comprehensive and suggests many useful and concrete suggestions for utilising ML in materials science, we (kindly) suggest that this is the wrong level of abstraction for a domain expert to utilise ML at. While the libraries themselves already abstract the low-level details, raising the level of abstraction further may be more appropriate for non-programmers.

Many factors may require utilising ML techniques at this low level of abstraction for functional properties such as performance. For example, obtaining high degrees of parallelism is cited by Zhou *et al.* as one reason for building a material science library upon PyTorch [139]. The PYSEQM library[35] implements functions applicable for semi-empirical quantum mechanics models, offering a high-level and domain-specific interface which is able to compute on the wide variety of GPUS which PyTorch supports, offering a speedup over other tools.

Example Study. As the publications from Lafluente *et al.* and Wang *et al.* are targeted toward chemical science researchers just beginning to utilise ML, we also present here a study of a chemical analysis tool *peakonly* utilising ML.

[33] https://github.com/ML4chemArg/Intro-to-Machine-Learning-in-Chemistry
[34] https://github.com/anthony-wang/BestPractices
[35] https://github.com/lanl/PYSEQM

High-level Step	**Step**	**Description**	**Classification**
Obtain Data	Load file	Load .mzML input data into tool	DS
	ROI detection	Run algorithm to detect ROIs	DS
Calculate	Classify	Run classification with CNN	ML
	Integrate	Run integration with CNN	ML
Analysis	Plot	Plot integrated ROIs	DS
	Write out	Write output CSV file	General

Table 2. Implicit workflow components in *peakonly* tool (Melnikov *et al.*) [90]

Melnikov *et al.* present an application of deep learning to classifying and integrating peaks in raw liquid chromatography-mass spectrometry (LC-MS) data [90]. The problem studied in the work is how to detect regions of interest (peaks) which occur in the noisy LC-MS data. Figure 11 is a visual abstract of the paper[36] showing how the data is first classified by a convolutional neural network (CNN) as a) noise, b) one or more peaks, or c) requiring manual classification. A second CNN then provides the integration boundaries. The results from the CNN are validated against another tool. The authors provide a graphical tool *peakonly* to perform these actions and visualise the output.

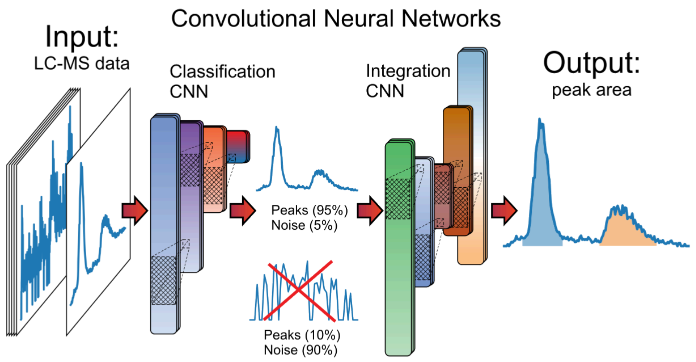


Fig. 11. Visual abstract from Melnikov *et al.* demonstrating classification and integration of peaks [90].

Problem Layer. The DS problem specified in the study is to detect and integrate the peaks in the data. The authors have transformed this into a *blended* problem by utilising the *expert mapping* given by prior work, where deep learning is used to perform peak detection and noise filtering.

Solution Workflow Layer. Figure 11 displays a high-level view of the paper's approach to peak classification and integration. This forms the basis of the workflow components which we extract in Table 2. Note that these steps are performed by a user interacting with the *peakonly* graphical interface, though a runner script is available.

Table 2 is our attempt at extracting the steps in the workflow from the publication and tool of Melnikov *et al.* The first column is the high-level step which summarises the individual steps in the second column. The *description* column details what the step performs, with the quoted text copied from the code to explain the DS steps. The last column in Table 2 is our rough classification of whether the step is *general* (generic programming code), *domain-specific* (DS), or *machine learning* (ML).

[36]Figure reprinted (adapted) with permission from [90]. Copyright 2020 American Chemical Society.

First, the DS file format is loaded into the tool. Then an algorithm runs to detect the regions of interest (ROIs). These two steps are highly domain-specific. The two CNNs are executed to first classify the ROIs and then to integrate the ones with detected peaks. We classify these as ML-intensive steps. Finally, the user is presented with the ROIs in a list where they are able to visually inspect the plot. The data can then be exported to a CSV file.

From our classification of these steps, it is fair to say that this is a *blended* implicit workflow. There is a balance of both DS and ML components within the workflow.

Implementation Layer. The implementation for the *peakonly* tool is in the Python language, with usage of common data science/ML libraries (matplotlib, numpy, pandas, scipy, PyTorch) as well as the DS library pymzML for mass spectrometry data. This code can be classified as *blended* due to the extensive mixed use of these libraries within the tool.

Remarks. The *peakonly* tool is an excellent example of domain experts utilising ML to solve a DS problem and providing an easy-to-use graphical interface to it. This lowers the barriers to entry for other domain experts to utilise ML on their similar problems.

6.2.2 ES2-MATLAB. For the second study which focuses on using the common MATLAB tool which combines data processing and ML techniques, we have selected a publication from Hasterok *et al.* in the field of geochemistry [59]. The studied problem is to use the chemical composition of metamorphic rocks to classify whether the origins of the rocks were sedimentary or igneous. The resulting MATLAB code is available on GitHub[37].

Problem Layer. On the problem layer, there is a clear DS problem of classifying rocks on their chemical structures.

However, the authors venture further into the ML domain to determine which available ML classifiers are best suited for their problem. They describe investigation of principal component analysis (PCA) to filter the data before classification, as well as comparing K-nearest neighbour, decision trees, ensemble trees, and testing various parameters within these classifiers.

A robust knowledge about applying ML to their DS problem is shown. Thus we can classify this paper as addressing a *blended* problem. The method of transforming the DS problem into a blended problem is not detailed, but based on the extensive related work cited we surmise that the authors gained this knowledge by reading past work which is a form of *expert mapping*.

Solution Workflow Layer. As mentioned, the solution provided by the authors does not contain an explicit workflow. Instead, the MATLAB code forms an implicit workflow operating on the input data and resulting in a classification or prediction. The authors provide their code in separate MATLAB scripts with comments, allowing us to reconstruct the predictor workflow and divide the scripts into DS and ML classifications. Note that this table contains workflows for both the training and prediction processes.

From this overview of the workflow, we can conclude that there is a mix of both DS and ML components. Thus this is a *blended* workflow.

[37] https://github.com/dhasterok/global_geochemistry/tree/master/protolith/

High-level Step	Step	Description	Classification
Obtain Data	Load learning data	Data loaded into MATLAB	General
	Treat learning data	`prep_for_cluster.m` selects data for training and testing. Can select to use meta-igneous/meta-sedimentary rocks or not	Blended
Training	Training	Using code generated from MATLAB's Classification Learner app	ML
Analysis	Write out	Write out model files	General
	Plot	Plot training performance	General
Prediction	Convert and read predict data	Converting XLS data to CSV, and reading	General
	FEFIX	"Convert all Fe to FeO and calculate Fe2+/Fe_total ratio"	DS
	CAT2OX	"Convert cations to oxide data when missing"	DS
	OXIDE_NORM	"Computes the oxide norm for a set of given oxides"	DS
	Prediction	Use MATLAB `predict` function with the classifier and input data	ML
Analysis	Write out	Write to CSV file	General
	Analyse	Plot classification performance	General

Table 3. Implicit workflow components in MATLAB scripts for predicting protoliths (Hasterok *et al.*) [59]

Implementation Layer. Following the classification of the implicit workflow as *blended*, it is clear that the implementation is also *blended*. This code was (presumably) hand-written by the authors. The exception is the code in the *training* step[38]. This code seems to have been generated by the MATLAB Classification Learner app[39].

Remarks. From reading the publication of Hasterok *et al*, it is clear that the authors have obtained a great deal of ML knowledge in addition to their domain expertise. They discuss pre-processing the data through principal component analysis (PCA) and perform a comparison between multiple ML classification techniques. This knowledge of both domains is reflected by the classifications given by our framework, where all of the problem, solution workflow, and implementation are blended.

The authors also leverage the ML functions built-into MATLAB for performing the training and classification, including the use of a MATLAB app to generate the appropriate training code. This sets the example study apart from ES1-PyTorch where the low-level ML library was directly used. Instead, MATLAB provides a higher level of abstraction for the domain expert, demonstrating how tool support can assist domain experts in applying ML concepts.

The code provided contains an implicit workflow defined in MATLAB code. However, the authors have taken the time to modularise this code into various functions. This improves the usability and reproducibility of the code amongst other researchers.

6.2.3 ES3-Kaggle. Kaggle[40] is an online data science platform allowing data scientists to share models and code. Kaggle is well-known for its "challenges", where an organisation posts their data and an evaluation metric, and asks the Kaggle community to come up with a solution to that metric. For example, the "NFL Big Data Bowl" challenge asked for a

[38] `train_RUSBoost_Classifier_30l_1000s_20190222.m`.
[39] https://www.mathworks.com/help/stats/classificationlearner-app.html
[40] https://kaggle.com/

prediction of how far one team will advance on the field during one play [56]. The data provided contained position, speed, and rotation information for each player on the field, weather information such as temperature, humidity, and wind velocity, and other data points. Once the competition has ended, the organisation can contact the winners to obtain an explanation of their solution.

Kaggle can be employed as a crowdsourcing tool to provide ML solutions for DS problems [17]. Briefly, the benefits of this approach is that domain experts can directly interface with ML experts on Kaggle-hosted forums to share best practices and domain-specific information. The hope is that this will lead to knowledge transfer and provide high-quality insights for the domain experts, and offer money, prestige, and valuable skill training for the ML experts [119].

There are a number of drawbacks, however. The domain experts have to spend effort to set up the contest by providing prize money and accurate problem data. Also, the solution provided by the ML experts may not be immediately applicable to the DS problem and lessons learned must be transferred back to the domain experts.

For example, in the Killer Shrimp Challenge a trivial solution was found by participants. The data was organised such that all data points with an index value greater than or equal to 2917769 had the presence of the killer shrimp. Thus the solutions of participants could simply test the index of each data point to obtain perfect accuracy on the predictions.

As an example of the effort required to transfer the ML lessons learned back to the domain experts, we point to the excellent article of Sutton *et al.* [118]. This article examines the top three solutions of a materials science challenge to determine the impact of the representation versus the learning method on the final accuracy. They describe how the first-place solution was a novel ML representation for material properties.

Problem Layer. Returning to the Killer Shrimp Challenge, the underlying problem is to detect the presence of the species *Dikerogammarus villosus* (also known as the "Killer Shrimp") which causes environmental damage and is invasive in Europe. The specific Kaggle challenge is to take data points on water salinity, temperature, depth, wave exposure, and the presence of sand, and predict whether the Killer Shrimp will be present or not [41]. We refer readers to the article of Bumann *et al.* [17] for a full description of the challenge set-up and interactions between the domain experts and challenge participants.

It is interesting to note that the DS problem of predicting the presence of Killer Shrimp was effectively turned into a problem of predicting 1 or 0 in a particular column of a spreadsheet. Thus we classify this as a *DS* problem being mapped into a *ML* problem due to the lack of DS concepts in the problem statement.

Solution Workflow Layer. In this report, we extract the (implicit) workflow from the publically-available second-place solution [122]. The solution is available as a Jupyter notebook which aids in the reconstruction of the workflow.

Table 4 displays our extraction of the workflow in the solution. Note that similar to other workflows, we define the loading and saving of CSV files as rather general steps. Most of the remaining steps are solely ML-focused. However, the author of the notebook has also added two DS columns. The first added feature is *water density* which is calculated from the temperature, salinity, and depth values. The second column is a classification of the wave exposure value to record whether the point is extremely or very exposed to waves.

In summary, this workflow is mostly comprised of ML components. However, due to the presence of these added DS features, we classify it as a *blended workflow* with a heavy focus towards the ML side.

[41] https://www.kaggle.com/c/killer-shrimp-invasion

High-level Step	**Step**	**Description**	**Classification**
Obtain Data	Load files	Load .csv files	General
	Clean data	Fill in missing data with `sklearn.impute.IterativeImputer`	ML
Feature Adding	Add Density	Calculate ocean density based on other columns	DS
	Classify Exposure	Classify wave exposure into categories	DS
	Add Temperature Feature	Add column based on trained polynomial of temperature	ML
	Add Outlier Feature	Add column to determine if data point is outlier	ML
Train	Train Classifier	Train a classifier	ML
Predict	Prediction	Predict the presence	ML
Analysis	Write out	Write output CSV file	General
	Plot Feature	Plot the importance of features	ML

Table 4. Workflow components in a Jupyter notebook for the Killer Shrimp Challenge [122].

Implementation Layer. The Jupyter notebook solution contains Python code and imports the expected data science/ML libraries (numpy, pandas, matplotlib, sklearn, xgboost). As the workflow is *blended* (though tilted towards ML), the implementation can be said to be *blended* as well.

Remarks. It was surprising to see features added to the data which were DS. Our expectation was that this sort of DS knowledge would not be as present in Kaggle solutions, based on the expertise focuses of the challenge organisers and the ML experts. For example, we wish to highlight this quote from the analysis of Gordeev and Singer on their winning entry for the football challenge [56]:

> Don't worry about having domain knowledge to attempt a specific problem. The main thing we learned in this competition is that you don't necessarily need domain knowledge or industry [sic] to successfully tackle the data science challenge. Sometimes it even can be an advantage, as you go in blindly without many prior assumptions that might wrongly steer your exploratory analyses.

Thus, providing a transformation from the DS problem to a ML representation may have to be balanced between prior DS knowledge and ML analyses.

6.3 Hybrid and Explicit Workflow Studies

The remaining studies presented here are those which explicitly represent the workflow (or a sub-workflow) in one of the standards or tools from Section 2.4. As discussed in Section 7, an explicit workflow aids with reproducibility, modularisation, collaboration, re-use, etc.

We also note that the explicit workflows tend to have a strong focus on enabling plugins or extensions for domains. This means that users are able to customise the workflows for their domains easily.

6.3.1 ES4-nipype. Practitioners may use workflow management systems such as *nipype* to develop reproducible workflows focusing on particular domain processing tasks. For example, Celestine *et al.* present a Python module[42] for performing pre-processing workflows such as DS file conversion and skull stripping for small mammal MRI brain data [21].

[42] https://github.com/sammba-mri/sammba-mri

High-level Step	**Step**	**Description**	**Classification**
Pre-Process Data	Load data	Load data into Python	General
	Convert format	Convert into BIDS format	DS
	fMRIPrep	fMRIPrep workflow for alignment/timing correction	DS
	FSL FEAT	Smoothing/filtering steps	DS
Defining Features	Data Preparation	"Background removement, normalise, resize, & cube definition"	DS
	Prep. Initial Features	Correlate cubes	DS
	Feature Selection	Rank correlations with stat. tests	ML
Classification	Classification	Run classifiers on data	ML

Table 5. Workflow components for predicting depression from MRI images (Mousavian *et al.*) [93]

These pre-processing tasks are essential for transforming the data such that it can be treated with ML. In this study, we analyse a workflow which uses fMRIPrep as a sub-workflow to process MRI data before it is used to predict whether the person in question has depression [93]. As mentioned in Section 2.4.1, this fMRIPrep automated workflow is built on top of nipype to perform pre-processing of fMRI data [39, 40]. As such, the fMRIPrep tool itself is an *explicitly* defined workflow. However, the authors of Mousavian *et al.* have defined an *implicit* workflow in Python to orchestrate the usage of the fMRIPrep tool. Thus this is an interesting *hybrid* workflow which utilises an explicit tool sub-workflow.

Problem Layer. The specified problem of Mousavian *et al.* is to classify MRI images on whether the subject has Major Depression Disorder (MDD) or not. There are three major challenges addressed in the article. The first is to investigate different correlation measures of the *voxels* (essentially three-dimensional pixels) of the MRI data. These correlation measures relate different areas of the brain together, and are used as features for the ML classification task. The second challenge is to handle imbalanced data sets where many subjects within the set do not have depression, as this can cause issues with classification. The third major challenge is to determine which of 14 ML classifiers performs best on the dataset.

From the problem and these specified challenges, it is clear that this is a *blended* problem which combines DS and ML concepts. Specifically, the correlation challenge is DS, while the imbalanced datasets and choice of classifier challenges are ML-specific.

Solution Workflow Layer. The study implicitly defines a workflow through its use of Python scripts[43]. However, Mousavian *et al.* represent the workflow as explicit blocks in the article [93]. Therefore it is straightforward to classify each task in the workflow as DS or ML as done for the other example studies. Table 5 shows the workflow as defined in the article of Mousavian *et al.*. The steps present in the article clearly identify that the majority of the workflow is *DS*. In particular, only the feature selection and actual classification steps are ML.

Implementation Layer. As the workflow of this study is mostly DS, it follows that the implementation is also very *DS*. In particular, heavy use of DS libraries and tools are used such as *dcm2niix* for format conversion, *PyDeface* for removing facial structure from the images, and the fMRIPrep implementation itself[44].

[43]https://github.com/moosavianmz/DetectingDepression
[44]https://github.com/nipreps/fmriprep

6.3.2 ES5-Orange. The example study for the *Orange* tool focuses on data mining analysis for Internet traffic from a smart school [1].

Problem Layer. The article from Adekitan *et al.* is an exploratory analysis on using machine learning techniques for prediction of Internet traffic at an educational institution. The specific problem is to predict a classification for both download traffic and upload traffic among *low*, *slight*, *moderate*, and *heavy* data traffic. The input information is a numerical day, week, and month, along with the previous day's traffic and an average of the previous two days.

Multiple ML classifiers are used and compared in this analysis from both the Orange and KNIME tools mentioned in Section 2.4.2. As the intention was to compare ML classifiers of two different workflow tools on the data, we classify this problem as a *machine learning* problem.

Solution Workflow Layer. Figure 12 shows the workflow of Adekitan *et al.* in the Orange tool, while the workflow for KNIME is found in their article [1]. The three *general* components at the bottom (*File*, *Data Table*, and *Box Plot*) are used to load the data and visualise it. The *Test & Score* component takes the five ML learners and the loaded data, and performs the ML classification task. The results are then passed to the four evaluation components on the right. We classify the *Test & Score* component, the learners, and the evaluation components as all ML components. There are no DS components in this workflow, however the feature engineering described in the article means that domain knowledge concerning the academic calendar of the institution has been encoded into the source data. Therefore, this workflow can be classified as mostly *ML-based* with a DS feature engineering step.

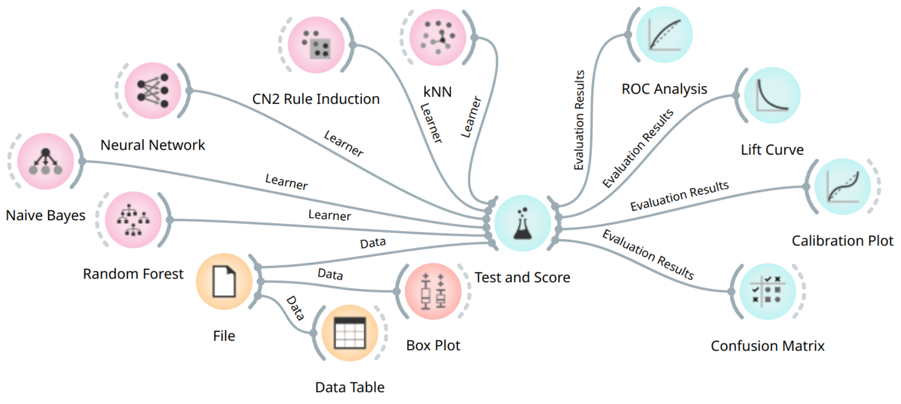


Fig. 12. Solution workflow in the Orange tool. Adapted from [1].

Implementation Layer. The execution for the Orange tool can be performed within the editor itself, or by calling the underlying Python code defined for each component[45]. In Orange, the code for the components used is built upon the scipy and scikit-learn modules. Therefore we classify this implementation as mostly *ML-based*.

Remarks. This example study represents an exploratory usage of ML techniques within a workflow, where the authors performed some DS feature extraction on the data and then applied different classifiers to determine the performance.

[45]Orange is currently unable to generate orchestration Python code from a workflow. See https://github.com/biolab/orange3/issues/1341

Workflow Name	Num. General Comps.	Num. DS Comps. (non-ML)	Num. DS Comps. (ML)	% DS	Classification
WF1: Co-registration and ROI Extraction	22	6	0	21	Mod. DS
WF2: Data Handling and Preprocessing	10	22	0	69	DS
WF3: Classification tumor vs. stroma	17	2	3	23	Mod. DS
WF4: Classification Infiltrating vs. Non-infiltrating	15	3	4	32	Mod. DS
WF5: Visualization	11	13	0	54	DS
WF6: Annotating Potential Identities	11	0	0	0	General

Table 6. Classification of Galaxy workflows for analysing urothelial cancer (Föll *et al.*) [47]

The classification performance found was quite low (55 to 63% accuracy), indicating that further DS feature extraction may be required to improve classification performance.

6.3.3 ES6-Galaxy. The last example study details a workflow for the Galaxy framework. The article of Föll *et al.* tackles supervised classification of urothelial (bladder) cancer using *mass spectrometry imaging* (MSI) [47].

Problem Layer. The input data studied by Föll *et al.* is obtained using MSI. This imaging technique is performed on a slice of tissue. For each region in the sample, the instrument provides a spectrum of the masses of present biomolecules. That is, for each "pixel" of the sample image a one-dimensional spectrum is created where peaks correspond to a particular biocompound. This can be used to visualise and classify regions of the sample where a particular biomolecule is present.

In the problem of Föll *et al.*, this MSI data is manually labelled by an expert as containing either *tumor* tissue or *stroma* (connecting tissue). Tumor tissue is then further classified into *invasive* or *non-invasive*. Therefore the problem statement is to develop a classification workflow which can pre-process and classify this unique spectral data. This problem can be said to be *DS* as it does not refer to the classification techniques used.

Solution Workflow Layer. An extract from the Galaxy workflow of Föll *et al.* is seen in Figure 3 on page 11. In particular, these components are in the workflow which classifies tissue as a tumor or stroma. The *MSI classification* component on the right-hand side of Figure 3 takes the MSI data and parameters and outputs a classification.

Similar to the other example studies, we present a broad analysis of the study workflows[46] in Table 6. For each workflow created by the study authors, we classify each component within as *general*, *domain-specific without ML concepts*, or *domain-specific with ML concepts*[47]. A percentage of the components which are DS is reported in a column on the right-hand side of Table 6 along with a classification of the workflow.

Table 6 indicates that the workflows for this study range from moderately DS to strongly DS. General components are used for dataset loading and manipulation while the DS components perform the non-trivial work. There are no non-domain-specific ML components in these workflows, and the component *MSI Classification* performs the domain-specific classification computations. Thus it is clear that this is a *DS* workflow.

[46] https://github.com/foellmelanie/Bladder_MSI_Manuscript_Galaxy_links
[47] Note that file reading and writing were counted as general components, which somewhat inflates their number. As well, in WF3 and WF4 repeated components for splitting a dataset ten times were counted only once to avoid over-representation.

Implementation Layer. Workflows are run by Galaxy through the web-based tool on either a public or private server. The individual components are defined through XML wrappers which define how to run the underlying tool.

For these particular workflows, the majority of the DS components are specific to MSI as they have been created by the authors in a previous work [46]. These components are wrappers around the Cardinal tool, which is a R language module specifically for analysing mass spectrometry-based imaging [8]. Thus the implementation of this workflow is mostly *DS*.

7 DISCUSSION

This section provides discussion for the main research topic of this article: *what are the ways in which domain experts can use workflow-based tools and techniques to to solve their domain-specific problems using machine learning*. For this discussion, we first present the benefits and drawbacks for structuring this research topic using our three-layer framework organised into two dimensions. Then we examine each of the challenge questions introduced in Section 1 and present what we see to be remaining research and integration challenges for the software engineering community. Finally, we present an illustrative example study where domain experts would employ a tool offering the transformations presented in our framework to easily obtain a machine learning-based solution to their problem.

7.1 Benefits and Drawbacks of the Three-Layer Framework

This section discusses some benefits and drawbacks of organising this research topic as a three-layer framework with inter- and intra-layer transformations.

7.1.1 Benefits.

Separation of Concerns. The main benefit of our framework is the separation of concerns into the three layers: *problem*, *workflow*, and *implementation*. Similar to domain-specific languages, this ensures that the domain expert first addresses the *problem space* which they are familiar with, rather than dealing with the accidental complexity of the workflow and implementation spaces. The framework defines transformations between these layers, offering the domain expert a structured way of progressing their solution.

This separation of concerns is also present in the workflow literature. For example, Lamprecht *et al.* [78] define six stages of workflows over time: *question or hypothesis*, *conceptual workflow*, *abstract workflow* (sequences of tools but not fully configured), *concrete workflow* (ready to run), *production workflow* (ready for reuse), and *scientific results*. The first two stages of *question/hypothesis* and *conceptual workflow* thus map onto our notion of domain-specific problem.

Implicit versus Explicit Workflows. Explicit workflows are both conceptually (and literally) at the centre of our framework. This is because we see numerous benefits with this formalism for domain experts to use in combination with ML.

First, it is obvious that there is compatibility between the use of scientific workflows and ML pipelines. They share the same underlying formalism due to the same concept of control and data flow, as well as concerns about modularity and reuse. A workflow-based approach also seems to be very amenable to visualisation and manipulation in graphical tools, allowing non-experts to quickly build a workflow for their problem.

Second, these standalone components are a useful abstraction over the technical details and complexity of ML approaches. The domain expert does not have to become familiar with the ML libraries or in some cases even a programming language to orchestrate the workflow. Again, this is in concordance with the principles of domain-specific engineering where the domain expert should focus on manipulating concepts within their domain.

Third, providing explicit components to the domain expert allows for enhanced traceability, reuse, scientific replication. As seen with the Galaxy tool, input and output history can be kept for every component in a workflow, along with explicit versioning and supporting information.

Fourth, a standardised workflow system can offer enhanced benefits for deployment on cloud or high-performance systems. For example, Lehmann *et al.* discuss the scalability benefits gained when porting an implicit workflow orchestrated with the Bash shell language to the Nextflow workflow system [83].

In some domains, the use of explicit workflows is a best practice. For example, Poldrack *et al.* discuss the use of nipype for reproducible and scalable workflows in neuroscience [104], while Reiter *et al.* provide a detailed article of techniques for biology experts to get started with workflows[106]. However, other fields may not have such a strong culture of workflows and still recommend coding for problem solving. As an illustrative example, Wang *et al.* recently suggest for material scientists to use PyTorch in Python for ML purposes [128].

Focus on Transformations. Another benefit of our framework is the focus on transformations between regions of layers, as well as between the layers themselves. These transformations can be phrased as challenge questions which are relevant to both software engineers and the domain experts who must use these transformations. This provides a clear intent for each transformation and allows for analysis and identification of current approaches and new techniques. In contrast, the frameworks of Lamprecht *et al.* [78] and Gil *et al.* [53] do not define transformations along the dimensions such as *domain-specificity* and *ML complexity*, which could improve the ability of domain experts to build these workflows. Instead, those works only discuss the transformations between the layers of our framework.

Explicit Domain-Specific and Machine Learning Complexity Dimensions. Section 3.1 describes how our framework relies on the two DS and ML dimensions to both a) categorize the problems, workflows, and implementations of domain experts, and b) offer spectrums whereby transformations can move these artefacts along a dimension to better suit the domain expert.

This explicit nature of the dimensions is in contrast to similar frameworks. For example, Combemale *et al.* describe the *models and data* (MODA) conceptual framework for organizing the processes of integrating model-driven engineering with data-centric systems which can involve ML approaches [24]. Within the MODA framework, domain knowledge is represented as "external knowledge" which is processed into the descriptive, predictive, and/or prescriptive models. For the machine learning aspect, ML techniques and algorithms can be present in these models.

The MODA framework focuses on decomposing model-driven engineering processes into these conceptual models. Thus, the dimensions of domain-specificity and complexity of machine learning are not present, preventing the use of transformations which move artefacts along these transformations and the organization of tools which provide these transformations. The MODA framework also does not have an explicit *problem layer* which the starting point for a domain expert who wishes to employ ML on their problem. Scientific workflows are mentioned by Combemale *et al.* [24], however their creation is through a "composition phase", and implementation through a "deployment phase" where no further information is provided. In contrast, our framework indicates the potential transformations which are available to create these workflows.

7.1.2 Drawbacks. There are a number of drawbacks to our organisation of the research problem onto this three-layer framework. The first is that the two dimensions selected of *domain-specific* and *complexity of machine learning* are not orthogonal as mentioned in Section 3.1. The divisions of these dimensions into regions is also fuzzy as we currently do not provide specific metrics to divide problems, workflows, or implementations due to the qualitative nature of these dimensions. In particular, the classifications of the problem, workflows, and implementations for the example studies in Section 6 are *ad-hoc*. In the future, we are addressing this limitation by developing additional measurable characteristics for each dimension as discussed in Section 7.5. These metrics will focus on task *specificity* [10].

Second, restricting these complex systems onto three layers is a gross simplification. In particular, we acknowledge that the implementation layer is most likely made up of numerous layers of domain-specific or general programming languages. We have classified implementation code in some example studies as *domain-specific* when this is just the top layer of what may be *machine learning* or *general* code at the lowest layer.

Our framework also does not represent many of the qualities which are required for a domain experts to make efficient use of tools which conform to this framework. For example, effective tools should be able to provide intelligence assistance to the domain expert and consider the feedback received [95], or optimize workflows based on static analysis [78]. This would aid the domain expert in applying the transformations discussed here. We have omitted these considerations as we focus on the transformations themselves and how they act on the artefacts of the domain expert. Future work will consider how the domain expert will interact with each transformation in the tool environment.

Lastly, we also acknowledge the incompleteness of this article to cover the research topic. It is impossible to fairly cover all domains or to give an impression of how prevalent the usage of any tool or technique is within a domain. This lack of comprehensiveness may render our framework less applicable when applied to a particular domain.

7.2 Considering Tool Support and Future Research for Challenges

In this section, we again present the challenge questions from the introduction in Section 1. For each question we then present our thoughts on how the challenge has been addressed by the tools and example studies seen in this article. We then present the potential research directions for each question.

Table 7 offers a general analysis on whether the tools discussed in this section addressed the challenge questions. For each question, a summary of the techniques from Sections 4 and 5 is presented. Symbols then provide an indication whether the challenge is *not*, *partially*, or *more fully* addressed by the tools. The last column then highlights the best examples which address each challenge.

From Table 7 it is clear that there are a number of challenge questions which are not addressed in current workflow tools or their ecosystems. We also identify that while most tools offer support for constructing workflows from problems, this is not yet an automated process. Thus there are ample opportunities for improving the experience of domain experts to create solution workflows as discussed in the next sections.

7.2.1 Mapping a DS problem to a form suitable for ML.

The first challenge we have selected focuses on the *problem layer*. That is, how to assist the domain expert to choose the machine learning techniques which may assist them. Our analysis in Table 7 indicates that none of the workflow frameworks discussed in this article tackle this challenge. This may be expected as the workflow management systems focus on the workflow and implementation layers of our framework. However, further integration of problem specification and mapping into these tools may assist domain experts.

For example, most of the example studies examined in Section 6 directly define a ML or blended problem. This could indicate that domain experts are having to gain sufficient machine learning knowledge to begin their study, instead of

Challenge	Techniques		Tool Support	Best Supporting Tool(s)
Mapping DS $\longrightarrow$ ML	Expert mapping	[6, 17, 61, 64, 118]	○	None
	Ontological reasoning	[78, 117]	○	None
	Data-driven approaches		○	None
Problem $\longrightarrow$ Workflow	Domain-specific examples	[58, 120]	●	Galaxy, nipype
	Workflow repositories	[34, 42, 49, 99]	●	Galaxy, KNIME, Nextflow, Orange
	Automatic reasoning	[51, 52, 69, 78]		APE, WINGS
Incr. workflow ML	Suggestions	[60, 74]	◐	Galaxy, KNIME
	Experimentation		◐	KNIME, Orange
Incr. workflow DS	DS component plugins	[123]	●	Node-RED, Orange, WINGS
	Suggestions	[74, 89]	◐	Galaxy, KNIME
	DS sub-workflows	[39, 57, 88]	◐	automate, Galaxy, nipype
Workflow $\longrightarrow$ Implementation	Deployment to DS platform	[65, 70, 104]	●	automate, Compi, Galaxy, Node-RED
	Run in tool	[85]	●	KNIME, Orange
Implementation $\longrightarrow$ Workflow	Language integration	[77]	◐	Nipype
	Code mining	[20]	○	None

Table 7. Broad analysis of techniques and tool support for answering challenge questions.
○ means that these tools do not support the transformation, ◐ is partial support, and ● means the transformation is well-supported.

leaving the problem as a DS problem. In particular, the study of Kaggle (Section 6.2.3) shows that domain experts will go to great lengths to obtain ML expertise on their problem.

One potential research direction to address this challenge involves bringing together semantic information from the DS and ML domains. In particular, ontological information could be used to match problems in a particular domain with a ML specification [78]. An alternative approach is to rely on data-driven approaches including deep learning. For example, if a suitable Large Language Model (LLM) is trained and integrated into a tool, a domain expert could directly ask for a suitable ML representation of their problem and obtain the mapping.

7.2.2 *Providing a solution workflow for a DS and/or ML problem.*

This challenge is the core focus of these workflow management systems as they provide the domain expert with the formalisms and assistance to build up the workflow. However, it is clear that tools have different ways of assisting the user, as described in Section 5.1. This includes assisted workflow composition, domain-specific examples, component libraries, and workflow repositories.

Many frameworks use a repository approach to improve the discoverability of workflows. That is, they provide a website for domain experts to search for a workflow which suits their needs [48]. We also point to the impressive Collective Knowledge framework [48] for a repository focusing on AI, ML, and system research[49].

However there still remains a challenge to connect the workflow solutions present in the repository with the problem faced by the domain expert [49]. The literature discusses manual and automatic semantic extraction techniques which can assist domain experts in finding workflows [35, 103]. However, enabling this at-scale across multiple domains and tools will continue to be a challenge.

[48]Examples include https://workflowhub.eu/, httpsL//hub.knime.com, and https://nf-co.re.
[49]https://cknowledge.io

As an example of the rich information to extract from workflows, we point to the work of Lamprecht *et al.* who discuss automatic discovery and the static analysis of workflows [78]. This includes technical parameters (versioning, FAIRness metrics, usage, etc.), domain-specific considerations (relevance of components to a domain, similarity to existing workflows, type and format of results, etc.), and community influence (citations, comments, ratings, etc.).

Automated *workflow composition* can also assist domain experts in building their workflows. A promising approach is to combine the techniques of AutoML [60] with domain knowledge about required domain-specific components [51, 69]. Large Language Models (LLMs) may also be a fruitful area of research to provide recommendations for workflow construction, as they could be tuned towards a particular domain [19, 134].

Whether a workflow is found or not, the domain expert is likely to want to add their own components. Thus another sub-challenge is to improve the suggestion possibilities for domain experts. Recommendations are found in the KNIME, Galaxy, and low-code tools [3, 74], but we see further potential in this research area. For example, suggesting larger pieces of workflows, improved semantic reasoning such that components relate directly to the domain [89], and employing machine learning techniques themselves to suggest components [100].

7.2.3 Allowing the domain expert to experiment with appropriate ML components. This challenge relates to the ease of which a domain expert can modify their workflow to include ML components. We believe that these domain experts should be encouraged and assisted to experiment with ML techniques within a workflow by the tooling. For example, the Orange tool (Section 6.3.2) makes it simple to add ML components to a workflow and visualise the results. This sort of visual experimentation fits perfectly with the component-based nature of workflows and ties in well with the challenge of improving the automatic recommendation systems. This experimentation step can also be partially automated by integrating AutoML techniques [60] or recommender systems [74] into the workflow tool to dynamically react to workflow changes.

7.2.4 Adding DS knowledge. The domain expert should be assisted in integrating their domain knowledge into the workflow, to lower the cognitive burden and to employ domain-specific libraries and algorithms. Here, the plug-in approach of providing domain-specific components is common among the tools examined here. Sub-workflows such as nipype (Section 6.3.1) can also assist the domain expert in reusing previously-built workflows.

Another research challenge is how to utilise the DS knowledge of a domain expert to directly improve the performance of ML techniques. This is seen in some example studies (such as in Section 6.2.3 and Section 6.3.1) where the features themselves were modified to take into domain knowledge. As with other challenges described here, one approach may be to combine domain knowledge represented in an ontology with suggestions for features to extract, such as provided by unsupervised feature extraction [116] or ontology embeddings [73]. Only the WINGS tool was seen to improve the performance of ML techniques by utilising DS knowledge. This is possible due to semantic reasoning of domain knowledge which is used to select appropriate components and parameters. More ML-specific techniques such as feature extraction are possible but there does not seem to be integrated support for this challenge in the tools examined here.

7.2.5 Producing an implementation from a workflow which is well-suited for a domain expert. Once a domain expert has created their workflow they must be able to run it in a scalable manner. This is addressed in multiple tools from Section 2.4. In particular, Galaxy offers powerful computational resources in a web-based platform. This allows bioinformatics experts to run their workflows on specialised platforms. However, there will always be additional challenges and opportunities to ensure that a domain expert can deploy their solution on the correct infrastructure. For example, code could be generated or parameterised based on the domain-specific tasks or data operated on, such as image- or voxel-based datasets [30]. The rise of containerisation also opens up many challenges to ensure that these containers are distributed for optimal scalability and security [104].

7.2.6 Extracting a workflow from an existing implementation (code or notebook). The final challenge we highlight in our article is to convert the existing implementations a domain expert may have into workflows. Amongst other benefits, this would improve the modularisation and dissemination of these solutions [20]. For example, Jupyter notebooks [50] are a well-known paradigm for storage, dissemination, and reproduction of experimental results [101]. Each *cell* in a notebook contains text or executable code, where the results of code are shown directly underneath. This format thus provides a narrative to provide context for the code, which is useful for disseminating results or tutorials on a topic.

Rule *et al.* suggest that scientists spend time to make Jupyter notebooks themselves form part of a workflow [108]. An interesting line of research is therefore to develop tooling and techniques to automate this process, such that the legacy notebooks of domain experts or machine learning experts[51] can be automatically promoted to explicit workflows [20].

None of the tools offer support for extracting workflows from existing code. However, the light-weight nature of the Python module-based tools (such as *luigi* or *nipype* from Section 2.4) could be seen as an easy way to “lift” existing code into an explicit workflow.

7.3 External Challenges

Beyond the challenges related to the framework itself, we also identify two other challenges which are important to increase the impact of addressing the problems specified in this article. These challenges are: a) strengthening the workflow community as a whole, and b) proposing tools and techniques to guide a domain expert in solving their problems.

7.3.1 Strengthening the Workflow Community. For domain experts to be able to effectively use workflows to solve their problems using ML, there must be a strong cross-domain workflow community. This community will then be able to pool knowledge and resources to best solve domain problems.

The excellent article of da Silva *et al.* suggests current challenges and proposed activities in a workflow community context [27]. The challenges they see are: *FAIR computational workflows*, *AI workflows*, *exascale challenges and beyond*, *APIs, reuse, interoperability and standards*, *training and education*, and *building a workflows community*.

We also see other avenues to strengthen the workflow community. In particular, we note the recent research and commercial interest of *low-code* platforms which are in some cases workflow management tools [14, 62]. It may be possible to leverage this interest into further developing workflow management systems by providing support for commercial domains. For example, the KNIME tool (Section 2.4.2) started development for solving pharmaceutical applications, but has now evolved to offer a commercial solution.

Crowdsourcing knowledge is also another possibility to build up the workflow community. For example, Paul-Gilloteaux *et al.* suggest the organisation of regular “taggathons” to annotate tools, workflows, components, databases, and training materials with terms from an ontology [103].

We also point towards Kaggle (Section 6.2.3) as an interesting community of domain and ML experts. Despite the issues with crowdsourcing [17, 119], it may be possible to further utilise this pool of knowledge. In particular, we

[50] https://jupyter.org/

[51] See Quaranta *et al.* for a dataset of Kaggle notebooks [105].

suggest that offering incentives for competition participants to include explicit workflows and reusable components in their solution may assist with reuse of their efforts.

Bringing in further expertise from software engineering sub-fields could also bring benefits to the workflow community. In particular, we draw from our own expertise in model-driven engineering to suggest that there are many research avenues to explore.

For instance, the multi- view/formalism/level of abstraction approach of multi-paradigm modelling may assist in reducing the cognitive complexity of domain experts [50]. A concrete example is providing *views* on a workflow such that the domain expert can focus on different aspects of the workflow as needed.

Another research avenue would be integration of model management approaches such as modelling variability and uncertainty techniques into workflow management tools [44]. The last research avenue we propose would be the integration of verification and validity techniques such as recording performance metrics [29], enhancing type safety [41], and checking for formal properties such as reachability [43].

7.3.2 Guiding the Domain Expert. The last challenge we mention in this article is how to guide the domain expert in both finding the best practices and tools for their domain, as well as their path in the framework.

Recently there have been articles in multiple domains walking a domain expert through the best tools and techniques available to employ ML [61, 75, 87, 107, 128]. For example, Nakhle and Harfouche provide four detailed Jupyter notebooks[52] walking domain experts in phenomics (plant sciences) through four steps of a ML task [97].

These four steps (*[image] dataset selection*, *data preprocessing*, *data analysis*, and *performance analysis and explanation*) are representative of most ML workflows. Therefore we suggest that similar dissemination efforts in different domains may assist domain experts. In particular, collaborative knowledge bases for a domain expert to navigate the tools and resources available in their domain may be useful.

Another tooling effort could be to dynamically assist the domain expert in producing template workflows based on their domain-specific problem [81, 98]. However, this raises the question of how to assist the domain expert through the regions of our framework in Section 3.

For example, consider three approaches to take a *DS* problem and arrive at a *blended* workflow. The first approach is on the problem level, where the domain expert is provided with basic ML knowledge to assist them in refining the DS problem to include ML concepts. The second approach is to immediately build a workflow, and then use AutoML [60] techniques, assist the user in experimentation (as in the Orange tool), or use ontological recommendations [89] to complete the workflow. The last approach is to follow principles from *human-guided machine learning* to iteratively build out the workflow [38, 51, 111]. In these three approaches, more or less automation may be appropriate depending on the task and user [129].

A further consideration is whether to hide or expose the ML concepts and components based on the ML knowledge of the domain expert. This could allow a user to work with a mostly *DS* workflow, and over time adjust the workflow towards a *blended* workflow as they gain insight and familiarity with the ML components.

7.4 Illustrative Use of the Framework

This section discusses how the conceptual framework presented in this article can be made actionable to guide domain experts from their domain problem to obtain a workflow utilizing ML. That is, if the transformations presented in this article were implemented in an available tool, we illustrate how a domain expert could use that tool. Thus this section

[52]Available here: https://github.com/HarfoucheLab/Ready-Steady-Go-AI

provides an illustrative 'user story' for researchers and tool builders to develop useful tools and techniques for domain experts which are aligned with our conceptual framework.

Domain Problem. The domain problem selected for this illustrative use is *melanoma classification*. That is, given an image of a *skin lesion*, can these images be classified as *malignant* (cancerous) or benign [135]. As skin cancer is a dangerous and prevalent disease, it is important to address any technical issues that these domain experts may have in understanding the problem and providing solutions.

In the literature, automatic melanoma classification has been examined for years using vision techniques [112]. Recently, ML approaches have also been applied [135], bolstered by the availability of high-quality datasets from the International Skin Imaging Collaboration (ISIC)[53].

The domain expert in this case is thus a dermatology researcher, who wishes to explore how ML techniques can be used to classify skin lesion images. In particular, we imagine they are concerned with examining different ML techniques and parameters to determine which solution provides the highest accuracy.

Problem Transformations. On the problem layer, a tool aligned with our framework would be able to accept this problem statement as text or using a modelled language. The mapping from the classification to the ML techniques would then be provided through techniques such as stored expert knowledge, ontological mappings, or querying a Large Language Model. For example, as the problem concerns *image classification*, then a convolutional neural network (CNN) approach is a good suggestion, while a decision tree is a poor suggestion, based on the form of the input data. Subtler suggestions may also be presented, such as suggesting different CNN architectures such as ResNet or DenseNet, based on what performs better for medical images [138].

As the domain expert may not be experienced enough to appreciate this difference between different architectures, this information may need to be hidden to avoid confusion. As discussed in Section 7.3.2, the tool should react to the ML expertise of the user to only reveal a digestible amount of complexity.

From these transformations, the domain expert may arrive at different formulations of the melanoma classification problem, such as: a) *how do I classify melanoma images using ML?*, b) *how do I classify melanoma images using CNNs?*, or c) *what is the best architecture for CNNs for classifying melanoma images?*. These formulations progressively escalate the complexity within the ML dimension.

Domain-specific knowledge may also factor into the problem. For example, dermatologists may use the ABCD(E) rule for assessing lesions, focusing on the lesion's *asymmetry*, *border*, *colour*, *diameter*, and *evolution* (over time) [96]. The domain question could then become: *how useful is the notion of the lesion diameter to CNN approaches?*.

Therefore the future tool would walk the user through specifying their domain problem, and indicating which ML techniques are appropriate to address that problem.

Obtaining the Workflow. From the problem, the tool should then provide the domain expert with the workflow components to address that problem. As mentioned in Table 7, this could be the tool providing a workflow repository to search for other workflows which address melanoma classification, offering a component library for image analysis, or by integrating ontological techniques to automatically suggest relevant components.

For example, as skin images often include body hair, it is likely that the domain expert will require a component to remove these artefacts from the images [82]. This component can then be suggested whenever the problem includes

[53]https://www.isic-archive.com/

medical images. Lee and Chin also discuss the performance of various types of image augmentation, which can be suggested as other components to include in the workflow whenever a CNN technique is used [82].

Thus the tool should be able to provide workflows or workflow components to the user, based on the specified problem. Another transformation is for the tool to load a user's Jupyter notebook as an initial workflow. This would allow for the below transformations to take place.

Suggestions for the Workflow. Once an initial or partial workflow has been produced for the user, the tool should use the transformations listed in Table 7 to tailor the workflow to the domain expert and the problem. For example, the domain expert should be provided with a guided environment to experiment with the ML components, where instant feedback and level-appropriate documentation is provided, as in the Orange tool (Section 2.4.2).

Suggestions can also be provided, such as recommending components or sub-workflows which appear in other workflows which address melanoma classification. For example, domain-specific visualization components could be suggested to indicate how the diameter and colour of a skin lesion corresponds to the classification of lesions. In this way, the workflow can be transformed to be more domain-specific or have more complex ML as the user desires.

Executing the Workflow. Once the workflow for classifying melanoma has been built, the domain expert will wish to execute it. Depending on the technical architecture, the size of the dataset, and the user's wishes, there are options including: a) running the components within the tool, b) deployment to a computational platform, and/or c) exporting to the user's preferred scripting/workflow language, such as Python. For example, the melanoma classification workflow could be deployed on a mobile device for user studies [126].

Any results should then be presented to the user within the tool, such that they may continue to experiment with and refine the workflow. For the dermatologist expert, these results should be presented in a domain-specific visualization if possible, explaining how the classification was performed. For example, this could be a heat-map to demonstrate whether the size markers around the lesion have been used (incorrectly) as features for classification [132].

7.5 Evaluation Metrics

Following the illustrative framework use in the last section, we present here potential quantitative and qualitative evaluation metrics for evaluating an application of the framework on a domain expert's problem. These metrics are intended to capture how the domain expert spends time and cognitive effort during the construction and modification of their workflow, such that the underlying tools and transformations can be measured, compared, and improved over time. The metrics we have selected are a mix of objective and subjective, as we also wish to take into account some human aspects of how the domain expert interacts with the framework and underlying tool [10].

Here, we provide a sample of metrics divided into four different categories reflecting aspects of our conceptual framework. There are metrics related to: a) how the *built artefacts* (problem, workflow, and implementation) fit into the framework regions, b) the *usage of the tool* implementing this framework, c) the *recommendations or guidance* provided by the tool, and d) the *workflow* itself produced by the tool.

7.5.1 Framework Metrics. As demonstrated in Section 6, we also define some metrics and guidelines for how the case study artefacts fit within our framework. We perform a quantitative *count of labelled components* as either *domain-specific* or *involving machine learning*. These counts are then used for a coarse division of each artefact into the regions of our framework.

We are developing further quantitative and qualitative metrics to better define the *domain-specificity* and *ML complexity* of each artefact. For the domain-specific dimension, it may be possible to provide a spectrum based on whether the involved components are *data-format* specific, *domain*-specific, or *field*-specific. For the ML complexity dimension, we are considering measures for the level of expertise or pre-requisite topics needed to understand the components, and measures for the configuration space implied by the components. For example, these measures could involve the number of ML algorithms used, the presence of deep learning, and a measure of the decomposition (black-box nature, layers, parameters) of each algorithm.

7.5.2 Tool Usage Metrics. Our conceptual framework will need to be realized by a tool for a domain expert to utilize it to create domain-specific ML workflows. Thus, these human-computer interaction metrics can be used to evaluate the implementation(s) of this framework.

Quantitative Metrics. The effort of the domain expert using the tool can be estimated using human-computer interaction metrics such as *time taken to complete tasks*, *number of mouse clicks*, or the *distance of mouse movement* (Fitt's Law) [86]. There are also *bio-feedback approaches* to measure the cognitive load of the user, such as using skin sensors (electrodermal activity) [55], eye tracking (areas of interest, fixations, scanpaths), or measuring brain activity (neuronal blood flow) [114]. Biehl *et al.* quantified the decrease in the number of decisions required by users when utilizing a tool [12], which could be considered a measure of reduced cognitive effort.

Qualitative Metrics. Domain expert interviews and surveys can be utilized to measure the subjective experience of their use of the tool. For example, the System Usability Scale (SUS) provides Likert scales to record the user's response to statements such as "I think that I could use the product without the support of the technical person", and "I could use the product without having to learn anything new" [16, 72]. The User Experience Questionnaire (UEQ)[54] provides 26 7-point Likert scales for recording products along the dimensions of *Attractiveness*, *Perspicuity* (easy to become familiar), *Efficiency* (lack of unnecessary effort), *Dependability* (predictability), *Stimulation* (exciting and motivating), and *Novelty* (creative and interesting) [79].

7.5.3 Recommendations and Guidance Metrics. With the rise of intelligent modelling guidance approaches, it is expected that tools which align with our conceptual framework will be able to assist the domain expert in specifying their problem and building their workflow. For example, as mentioned in Section 7.4, the tool should be able to suggest relevant components based on the problem domain and/or the components already existing in the workflow. Here, we suggest relevant metrics to measure how useful these suggestions are.

Quantitative Metrics. When the tool presents suggestions to the user, the user may accept them. Thus, the *acceptance rate* can be measured to determine the suggestion usefulness, or how *diverse* the resulting artefacts are [9]. If a ground truth is provided, metrics such as *precision* and *recall* can provide an indication of the success of recommendations [33]. The efficiency (*time and resources taken*) for this guidance process is also important to ensure that suggestions are provided in a *timely* manner [94].

Qualitative Metrics. Mussbacher *et al.* have defined various metrics for evaluating intelligent modelling assistants [94]. These metrics include qualitative aspects such as: the *quality* of the results, *autonomy* for the assistant to gather information, *relevance* of the results (defined as usefulness for each user), *confidence* of how sure the assistant is about

[54] https://www.ueq-online.org/

the suggestion, how much *trust* the user has in the assistance, and *explainability* for whether the user understands the suggestions.

7.5.4 Workflow Metrics. As the domain expert is building a workflow using our conceptual framework and the tool, it is possible to define metrics about the workflow itself. These metrics can be used to compare workflows built using different tools, or after the workflows undergo the transformations as we have defined them in our framework.

Quantitative Metrics. The *component count* of the workflow may be an estimate of how much effort is used to create and understand it. D'Aloisio *et al.* define other measurements of ML workflows such as *computational complexity* (space and time requirements), the *prediction correctness* (precision, recall, *etc.*), and *fairness* (avoiding prejudice or favouritism) [28].

Qualitative Metrics. D'Aloisio *et al.* also define inter-related *privacy* and *interpretability* qualitative metrics for ML workflows [28]. Privacy refers to the hiding of sensitive information in the dataset, while interpretability focuses on how the user can understand the results produced by the workflow.

8 CONCLUSION

This article presents a conceptual framework to structure the decisions, transformations, and tools whereby domain experts can utilise machine learning to solve their problems. In particular, we focus on the computational workflow representation of solutions where executable components are connected by control and data flow edges. Examining the state-of-the-practice, we identify six key challenges that a domain expert may face in developing an executable workflow:

- Map a DS problem to a form suitable for ML
- Obtain a solution workflow for a DS and/or ML problem
- Experiment with ML tools and techniques within a workflow
- Add DS knowledge to improve ML performance (e.g., feature engineering)
- Produce an implementation from a workflow which is well-suited for a domain expert (in terms of scalability, DS tooling, etc.)
- Extract a workflow from an existing implementation (code, Jupyter notebook)

These challenges are represented by transformation within regions of our conceptual framework. This framework has three layers, consisting of the *problem layer*, *workflow solution layer*, and *implementation layer*. Each layer is further structured with two dimensions representing the *domain specificity* and *complexity of machine learning* of the artefacts on that layer.

This conceptual framework structures our investigation of the state-of-the-practice for how domain experts are employing machine learning. In particular, a selection of textual and graphical workflow tools are briefly presented to illustrate tool support for the challenges and transformations we have identified. Example studies selected from recent works in various domains further explore how the problems, workflows, and implementations created by domain experts are heterogeneous in terms of the amount of domain specificity and machine learning complexity. We also provide a short discussion on each challenge to indicate possible research directions, and a thought experiment of efficient workflow development in a tool which conforms to our framework.

This article thus forms a basis for further discussion and software engineering research into assisting domain experts with developing workflow solutions which employ machine learning. Integrating best practices from software engineering and across tools will reduce the friction for domain experts to utilise these powerful techniques and unlock new possibilities in their application to pressing scientific issues. In particular, we present this framework as a way of structuring the transformations to be integrated into future tool architectures, such that users can use these transformations to easily obtain and customize their workflow.

Our current and future research focuses on the implementation of a tool conforming to this framework which is able to guide a domain expert from a problem to a solution. Following the illustrative case described in Section 7.4, we are currently collaborating with domain experts to develop tooling suitable to create a workflow for melanoma classification. We will then evaluate our tool using quantitative and qualitative methods in user-facing experiments to estimate the time and cognitive effort saved by the tool support (Section 7.5). In the future, this tooling framework will then serve as a research platform to investigate how to better assist domain experts in creating ML workflows.

ACKNOWLEDGMENTS

This work was performed when the first author was a post-doctoral researcher in DIRO at the Université de Montréal. The authors would like to sincerely thank their colleagues Dr. Istvan David (McMaster University) and Dr. Jessie Galasso (McGill University) for their insightful discussions on this article.

CONFLICT OF INTEREST

The authors declare that they have no conflict of interest.

REFERENCES

[1] Aderibigbe Israel Adekitan, Jeremiah Abolade, and Olamilekan Shobayo. 2019. Data mining approach for predicting the daily Internet data traffic of a smart university. *Journal of Big Data* 6, 1 (2019), 1–23. Figure adapted under the Creative Commons Attribution 4.0 International License http://creativecommons.org/licenses/by/4.0/.

[2] Azza E Ahmed, Joshua M Allen, Tajesvi Bhat, Prakruthi Burra, Christina E Fliege, Steven N Hart, Jacob R Heldenbrand, Matthew E Hudson, Dave Deandre Istanto, Michael T Kalmbach, et al. 2021. Design considerations for workflow management systems use in production genomics research and the clinic. *Scientific reports* 11, 1 (2021), 1–18.

[3] Lissette Almonte, Iván Cantador, Esther Guerra, and Juan de Lara. 2020. Towards automating the construction of recommender systems for low-code development platforms. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 1–10.

[4] Ethem Alpaydin. 2020. *Introduction to machine learning*. MIT press.

[5] Peter Amstutz, Maxim Mikheev, Michael R. Crusoe, Nebojša Tijanić, Samuel Lampa, et al. 2021. Existing Workflow systems. Common Workflow Language wiki, GitHub. https://s.apache.org/existing-workflow-systems Updated 2021-12-14, accessed 2022-01-06.

[6] Sanjay Aneja, Enoch Chang, and Antonio Omuro. 2019. Applications of artificial intelligence in neuro-oncology. *Current opinion in neurology* 32, 6 (2019), 850–856.

[7] Syeeda Nilofer Banoo. 2020. *Flow-based Programming for Machine Learning*. Master's thesis. Technical University of Munich.

[8] Kyle D Bemis, April Harry, Livia S Eberlin, Christina Ferreira, Stephanie M van de Ven, Parag Mallick, Mark Stolowitz, and Olga Vitek. 2015. Cardinal: an R package for statistical analysis of mass spectrometry-based imaging experiments. *Bioinformatics* 31, 14 (2015), 2418–2420.

[9] Meriem Ben Chaaben. 2023. *Few-Shot Prompt Learning for Automating Model Completion*. Master's thesis. Université de Montréal.

[10] Oussama Ben Sghaier, Jean-Sebastien Boudrias, and Houari Sahraoui. 2023. Toward Optimal Psychological Functioning in AI-driven Software Engineering Tasks: The SEWELL-CARE Assessment Framework. arXiv:arXiv:2311.07410

[11] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Christoph Sieb, Kilian Thiel, and Bernd Wiswedel. 2008. KNIME: The Konstanz Information Miner. In *Data Analysis, Machine Learning and Applications*, Christine Preisach, Hans Burkhardt, Lars Schmidt-Thieme, and Reinhold Decker (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 319–326.

[12] Matthias Biehl, Jad El-Khoury, Frédéric Loiret, and Martin Törngren. 2014. On the modeling and generation of service-oriented tool chains. *Software & Systems Modeling* 13 (2014), 461–480.

[13] Sumon Biswas, Mohammad Wardat, and Hridesh Rajan. 2022. The art and practice of data science pipelines: A comprehensive study of data science pipelines in theory, in-the-small, and in-the-large. In *Proceedings of the 44th International Conference on Software Engineering*. 2091–2103.

[14] Alexander C Bock and Ulrich Frank. 2021. In search of the essence of low-code: an exploratory study of seven development platforms. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 57–66.

[15] Andrea Borghesi, Federico Baldo, and Michela Milano. 2020. Improving deep learning models via constraint-based domain knowledge: a brief survey. *arXiv preprint arXiv:2005.10691* (2020).

[16] John Brooke. 1996. SUS: a "quick and dirty" usability scale. *Usability evaluation in industry* 189, 3 (1996), 189–194.

[17] Adrian Bumann and Robin Teigland. 2021. The Challenges of Knowledge Combination in ML-based Crowdsourcing–The ODF Killer Shrimp Challenge using ML and Kaggle. In *Proceedings of the 54th Hawaii International Conference on System Sciences*. 4930.

[18] Jordi Cabot. 2020. Positioning of the low-code movement within the field of model-driven engineering. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 1–3.

[19] Yuzhe Cai, Shaoguang Mao, Wenshan Wu, Zehua Wang, Yaobo Liang, Tao Ge, Chenfei Wu, Wang You, Ting Song, Yan Xia, et al. 2023. Low-code LLM: Visual Programming over LLMs. *arXiv preprint arXiv:2304.08103* (2023).

[20] Lucas AMC Carvalho, Regina Wang, Yolanda Gil, and Daniel Garijo. 2017. NiW: Converting Notebooks into Workflows to Capture Dataflow and Provenance.. In *K-CAP Workshops*.

[21] Marina Celestine, Nachiket A Nadkarni, Clément M Garin, Salma Bougacha, and Marc Dhenain. 2020. Sammba-MRI: A library for processing SmAll-MaMmal BrAin MRI data in Python. *Frontiers in neuroinformatics* 14 (2020), 24.

[22] Moharram Challenger, Ken Vanherpen, Joachim Denil, and Hans Vangheluwe. 2020. FTG+PM: Describing Engineering Processes in Multi-Paradigm Modelling. In *Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems*. Springer, Cham, 259–271.

[23] Michele Chinosi and Alberto Trombetta. 2012. BPMN: An introduction to the standard. *Computer Standards & Interfaces* 34, 1 (2012), 124–134.

[24] Benoit Combemale, Jorg Kienzle, Gunter Mussbacher, Hyacinth Ali, Daniel Amyot, Mojtaba Bagherzadeh, Edouard Batot, Nelly Bencomo, Benjamin Benni, Jean-Michel Bruel, et al. 2020. A Hitchhiker's Guide to Model-Driven Engineering for Data-Centric Systems. *IEEE Software* 38, 4 (2020), 71–84.

[25] OpenJS Foundation & Contributors. [n. d.]. Node-RED. https://nodered.org.

[26] Michael R Crusoe, Sanne Abeln, Alexandru Iosup, Peter Amstutz, John Chilton, Nebojša Tijanić, Hervé Ménager, Stian Soiland-Reyes, Bogdan Gavrilovic, and Carole Goble. 2021. Methods included: Standardizing computational reuse and portability with the common workflow language. *arXiv preprint arXiv:2105.07028* (2021).

[27] Rafael Ferreira da Silva, Henri Casanova, Kyle Chard, Ilkay Altintas, Rosa M Badia, Bartosz Balis, Tainã Coleman, Frederik Coppens, Frank Di Natale, Bjoern Enders, et al. 2021. A community roadmap for scientific workflows research and development. In *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*. IEEE, 81–90.

[28] Giordano d'Aloisio, Antinisca Di Marco, and Giovanni Stilo. 2022. Modeling Quality and Machine Learning Pipelines through Extended Feature Models. *arXiv preprint arXiv:2207.07528* (2022).

[29] István Dávid, Hans Vangheluwe, and Yentl Van Tendeloo. 2018. Translating engineering workflow models to DEVS for performance evaluation. In *2018 Winter Simulation Conference (WSC)*. IEEE, 616–627.

[30] Ewa Deelman, Anirban Mandal, Ming Jiang, and Rizos Sakellariou. 2019. The role of machine learning in scientific workflows. *The International Journal of High Performance Computing Applications* 33, 6 (2019), 1128–1139.

[31] Janez Demšar, Tomaž Curk, Aleš Erjavec, Črt Gorup, Tomaž Hočevar, Mitar Milutinovič, Martin Možina, Matija Polajnar, Marko Toplak, Anže Starič, et al. 2013. Orange: data mining toolbox in Python. *the Journal of machine Learning research* 14, 1 (2013), 2349–2353.

[32] Janez Demšar and Blaz Zupan. 2005. From experimental machine learning to interactive data mining. *White Paper (www. ailab. si/orange), Faculty of Computer and Information science, University of Ljubljana* (2005).

[33] Claudio Di Sipio, Juri Di Rocco, Davide Di Ruscio, and Phuong T Nguyen. 2023. MORGAN: a modeling recommender system based on graph kernel. *Software and Systems Modeling* (2023), 1–23.

[34] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. 2017. Nextflow enables reproducible computational workflows. *Nature biotechnology* 35, 4 (2017), 316–319.

[35] Juan Sebastian Beleno Diaz and Claudia Bauzer Medeiros. 2017. WorkflowHunt: combining keyword and semantic search in scientific workflow repositories. In *2017 IEEE 13th International Conference on e-Science (e-Science)*. IEEE, 138–147.

[36] William Digan, Aurélie Névéol, Antoine Neuraz, Maxime Wack, David Baudoin, Anita Burgun, and Bastien Rance. 2021. Can reproducibility be improved in clinical natural language processing? A study of 7 clinical NLP suites. *Journal of the American Medical Informatics Association* 28, 3 (2021), 504–515.

[37] Alexander Dunn, Qi Wang, Alex Ganose, Daniel Dopp, and Anubhav Jain. 2020. Benchmarking materials property prediction methods: the Matbench test set and Automatminer reference algorithm. *npj Computational Materials* 6, 1 (2020), 1–10.

[38] Vito D'Orazio, James Honaker, Raman Prasady, and Michael Shoemate. 2019. Modeling and forecasting armed conflict: AutoML with human-guided machine learning. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 4714–4723.

[39] Oscar Esteban, Rastko Ciric, Karolina Finc, Ross W Blair, Christopher J Markiewicz, Craig A Moodie, James D Kent, Mathias Goncalves, Elizabeth DuPre, Daniel EP Gomez, et al. 2020. Analysis of task-based functional MRI data preprocessed with fMRIPrep. *Nature protocols* 15, 7 (2020), 2186–2202.

[40] Oscar Esteban, Christopher J Markiewicz, Ross W Blair, Craig A Moodie, A Ilkay Isik, Asier Erramuzpe, James D Kent, Mathias Goncalves, Elizabeth DuPre, Madeleine Snyder, et al. 2019. fMRIPrep: a robust preprocessing pipeline for functional MRI. *Nature methods* 16, 1 (2019), 111–116.

[41] Riley Evans, Samantha Frohlich, and Meng Wang. 2022. CircuitFlow: A Domain Specific Language for Dataflow Programming. In *International Symposium on Practical Aspects of Declarative Languages*. Springer, 79–98.

[42] Philip A Ewels, Alexander Peltzer, Sven Fillinger, Harshil Patel, Johannes Alneberg, Andreas Wilm, Maxime Ulysse Garcia, Paolo Di Tommaso, and Sven Nahnsen. 2020. The nf-core framework for community-curated bioinformatics pipelines. *Nature biotechnology* 38, 3 (2020), 276–278.

[43] Javier Fabra, María José Ibáñez, Joaquín Ezpeleta, et al. 2018. Behavioral analysis of scientific workflows with semantic information. *IEEE Access* 6 (2018), 66030–66046.

[44] Michalis Famelis and Marsha Chechik. 2019. Managing design-time uncertainty. *Software & Systems Modeling* 18, 2 (2019), 1249–1284.

[45] Yuanrui Fan, Xin Xia, David Lo, and Ahmed E Hassan. 2018. Chaff from the wheat: Characterizing and determining valid bug reports. *IEEE transactions on software engineering* 46, 5 (2018), 495–525.

[46] Melanie Christine Föll, Lennart Moritz, Thomas Wollmann, Maren Nicole Stillger, Niklas Vockert, Martin Werner, Peter Bronsert, Karl Rohr, Björn Andreas Grüning, and Oliver Schilling. 2019. Accessible and reproducible mass spectrometry imaging data analysis in Galaxy. *GigaScience* 8, 12 (2019), giz143.

[47] Melanie Christine Föll, Veronika Volkmann, Kathrin Enderle-Ammour, Konrad Wilhelm, Dan Guo, Olga Vitek, Peter Götz Christian Bronsert, and Oliver Schilling. 2021. Moving translational mass spectrometry imaging towards transparent and reproducible data analyses: A case study of an urothelial cancer cohort analyzed in the Galaxy framework. *bioRxiv* (2021).

[48] Grigori Fursin. 2021. Collective knowledge: organizing research projects as a database of reusable components and portable workflows with common interfaces. *Philosophical Transactions of the Royal Society A* 379, 2197 (2021), 20200211.

[49] Daniel Garijo, Yolanda Gil, and Oscar Corcho. 2017. Abstract, link, publish, exploit: An end to end framework for workflow sharing. *Future Generation Computer Systems* 75 (2017), 271–283.

[50] Holger Giese, Tihamér Levendovszky, and Hans Vangheluwe. 2006. Summary of the workshop on multi-paradigm modeling: Concepts and tools. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 252–262.

[51] Yolanda Gil, James Honaker, Shikhar Gupta, Yibo Ma, Vito D'Orazio, Daniel Garijo, Shruti Gadewar, Qifan Yang, and Neda Jahanshad. 2019. Towards human-guided machine learning. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*. 614–624.

[52] Yolanda Gil, Varun Ratnakar, and Christian Fritz. 2010. Assisting scientists with complex data analysis tasks through semantic workflows. In *2010 AAAI Fall Symposium Series*.

[53] Yolanda Gil, Varun Ratnakar, Jihie Kim, Pedro Gonzalez-Calero, Paul Groth, Joshua Moody, and Ewa Deelman. 2010. Wings: Intelligent workflow-based design of computational experiments. *IEEE Intelligent Systems* 26, 1 (2010), 62–72.

[54] Primož Godec, Matjaž Pančur, Nejc Ilenič, Andrej Čopar, Martin Stražar, Aleš Erjavec, Ajda Pretnar, Janez Demšar, Anže Starič, Marko Toplak, et al. 2019. Democratized image analytics by visual programming through integration of deep models and small-scale machine learning. *Nature communications* 10, 1 (2019), 1–7.

[55] Lucian José Gonçales, Kleinner Farias, and Bruno C da Silva. 2021. Measuring the cognitive load of software developers: An extended Systematic Mapping Study. *Information and Software Technology* 136 (2021), 106563.

[56] Dmitry Gordeev and Philipp Singer. 2020. From Football Newbies to NFL (data) Champions: A Winner's Interview with The Zoo. https://medium.com/kaggle-blog/from-football-newbies-to-nfl-data-champions-a-winners-interview-with-the-zoo-391793168714

[57] Krzysztof Gorgolewski, Christopher D Burns, Cindee Madison, Dav Clark, Yaroslav O Halchenko, Michael L Waskom, and Satrajit S Ghosh. 2011. Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python. *Frontiers in neuroinformatics* 5 (2011), 13.

[58] Qiang Gu, Anup Kumar, Simon Bray, Allison Creason, Alireza Khanteymoori, Vahid Jalili, Björn Grüning, and Jeremy Goecks. 2021. Galaxy-ML: An accessible, reproducible, and scalable machine learning toolkit for biomedicine. *PLOS Computational Biology* 17, 6 (2021), e1009014.

[59] D Hasterok, Matthew Gard, CMB Bishop, and David Kelsey. 2019. Chemical identification of metamorphic protoliths using machine learning methods. *Computers & Geosciences* 132 (2019), 56–68.

[60] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A Survey of the State-of-the-Art. *Knowledge-Based Systems* 212 (2021), 106622.

[61] Kexin Huang, Cao Xiao, Lucas M Glass, Cathy W Critchlow, Greg Gibson, and Jimeng Sun. 2021. Machine learning applications for therapeutic tasks with genomics data. *Patterns* 2, 10 (2021), 100328. https://doi.org/10.1016/j.patter.2021.100328

[62] Felicien Ihirwe, Davide Di Ruscio, Silvia Mazzini, Pierluigi Pierini, and Alfonso Pierantonio. 2020. Low-code Engineering for Internet of Things: A state of research. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 1–8.

[63] Peter Ivie and Douglas Thain. 2018. Reproducibility in scientific computing. *ACM Computing Surveys (CSUR)* 51, 3 (2018), 1–36.

[64] Kevin Maik Jablonka, Daniele Ongari, Seyed Mohamad Moosavi, and Berend Smit. 2020. Big-data science in porous materials: materials genomics and machine learning. *Chemical reviews* 120, 16 (2020), 8066–8129.

[65] Anubhav Jain, Shyue Ping Ong, Wei Chen, Bharat Medasani, Xiaohui Qu, Michael Kocher, Miriam Brafman, Guido Petretto, Gian-Marco Rignanese, Geoffroy Hautier, et al. 2015. FireWorks: A dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience* 27, 17 (2015), 5037–5059.

[66] Vahid Jalili, Enis Afgan, Qiang Gu, Dave Clements, Daniel Blankenberg, Jeremy Goecks, James Taylor, and Anton Nekrutenko. 2020. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2020 update. *Nucleic acids research* 48, W1 (2020), W395–W402.

[67] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature* 596, 7873 (2021), 583–589.

[68] Juha Kärnä, Juha-Pekka Tolvanen, and Steven Kelly. 2009. Evaluating the use of domain-specific modeling in practice. In *Proceedings of the Object-Oriented Programming, Systems, Languages and Applications workshop on Domain-Specific Modeling*.
[69] Vedran Kasalica and Anna-Lena Lamprecht. 2020. APE: A command-line tool and API for automated workflow composition. In *International Conference on Computational Science*. Springer, 464–476.
[70] Athanassios M Kintsakis, Fotis E Psomopoulos, Andreas L Symeonidis, and Pericles A Mitkas. 2017. Hermes: Seamless delivery of containerized bioinformatics workflows in hybrid cloud (HTC) environments. *SoftwareX* 6 (2017), 217–224.
[71] Panu Kortelainen. 2021. *Manage Your Workflows: A Classification Framework and Technology Review of Workflow Management Systems*. Ph. D. Dissertation. Tampere University.
[72] Philip Kortum, Claudia Ziegler Acemyan, and Frederick L Oswald. 2021. Is it time to go positive? Assessing the positively worded system usability scale (SUS). *Human factors* 63, 6 (2021), 987–998.
[73] Maxat Kulmanov, Fatima Zohra Smaili, Xin Gao, and Robert Hoehndorf. 2021. Semantic similarity and machine learning with ontologies. *Briefings in bioinformatics* 22, 4 (2021), 1–18.
[74] Anup Kumar, Helena Rasche, Björn Grüning, and Rolf Backofen. 2021. Tool recommender system in Galaxy using deep learning. *GigaScience* 10, 1 (2021), giaa152. Figure adapted under the Creative Commons Attribution 4.0 International License http://creativecommons.org/licenses/by/4.0/.
[75] Deborah Lafuente, Brenda Cohen, Guillermo Fiorini, Agustín Alejo García, Mauro Bringas, Ezequiel Morzan, and Diego Onna. 2021. A Gentle Introduction to Machine Learning for Chemists: An Undergraduate Workshop Using Python Notebooks for Visualization, Data Processing, Analysis, and Modeling. *Journal of Chemical Education* 98, 9 (2021), 2892–2898.
[76] Samuel Lampa, Jonathan Alvarsson, and Ola Spjuth. 2016. Towards agile large-scale predictive modelling in drug discovery with flow-based programming design principles. *Journal of cheminformatics* 8, 1 (2016), 1–12.
[77] Samuel Lampa, Martin Dahlö, Jonathan Alvarsson, and Ola Spjuth. 2019. SciPipe: A workflow library for agile development of complex and dynamic bioinformatics pipelines. *GigaScience* 8, 5 (2019), giz044.
[78] Anna-Lena Lamprecht, Magnus Palmblad, Jon Ison, Veit Schwämmle, Mohammad Sadnan Al Manir, Ilkay Altintas, Christopher JO Baker, Ammar Ben Hadj Amor, Salvador Capella-Gutierrez, Paulos Charonyktakis, et al. 2021. Perspectives on automated composition of workflows in the life sciences. *F1000Research* 10 (2021).
[79] Bettina Laugwitz, Theo Held, and Martin Schrepp. 2008. Construction and evaluation of a user experience questionnaire. In *HCI and Usability for Education and Work: 4th Symposium of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society, USAB 2008, Graz, Austria, November 20-21, 2008. Proceedings 4*. Springer, 63–76.
[80] Rodger Lea. [n. d.]. Node-RED Programming Guide. http://noderedguide.com/. Accessed January 2022..
[81] Doris Jung-Lin Lee and Stephen Macke. 2020. A Human-in-the-loop Perspective on AutoML: Milestones and the Road Ahead. *IEEE Data Engineering Bulletin* (2020).
[82] Kin Wai Lee and Renee Ka Yin Chin. 2020. The effectiveness of data augmentation for melanoma skin cancer prediction using convolutional neural networks. In *2020 IEEE 2nd International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*. IEEE, 1–6.
[83] Fabian Lehmann, David Frantz, Sören Becker, Ulf Leser, and Patrick Hostert. 2021. FORCE on Nextflow: Scalable analysis of earth observation data on commodity clusters. In *Int. Workshop on Complex Data Challenges in Earth Observation*.
[84] Jeremy Leipzig. 2017. A review of bioinformatic pipeline frameworks. *Briefings in bioinformatics* 18, 3 (2017), 530–536.
[85] Hugo López-Fernández, Osvaldo Graña-Castro, Alba Nogueira-Rodríguez, Miguel Reboiro-Jato, and Daniel Glez-Peña. 2021. Compi: a framework for portable and reproducible pipelines. *PeerJ Computer Science* 7 (2021), e593.
[86] I Scott MacKenzie. 1992. Fitts' law as a research and design tool in human-computer interaction. *Human-computer interaction* 7, 1 (1992), 91–139.
[87] Vivien Marx. 2020. When computational pipelines go 'clank'. *Nature Methods* 17, 7 (2020), 659–662.
[88] Kiran Mathew, Joseph H Montoya, Alireza Faghaninia, Shyam Dwarakanath, Muratahan Aykol, Hanmei Tang, Iek-heng Chu, Tess Smidt, Brandon Bocklund, Matthew Horton, et al. 2017. Atomate: A high-level interface to generate, execute, and analyze computational materials science workflows. *Computational Materials Science* 139 (2017), 140–152.
[89] Russell P McIver. 2015. *A knowledge-based approach to scientific workflow composition*. Ph. D. Dissertation. Cardiff University.
[90] Arsenty D Melnikov, Yuri P Tsentalovich, and Vadim V Yanshole. 2019. Deep learning for the precise peak detection in high-resolution LC–MS data. *Analytical chemistry* 92, 1 (2019), 588–592.
[91] Marçal Mora-Cantallops, Salvador Sánchez-Alonso, Elena García-Barriocanal, and Miguel-Angel Sicilia. 2021. Traceability for Trustworthy AI: A Review of Models and Tools. *Big Data and Cognitive Computing* 5, 2 (2021), 20.
[92] J Paul Morrison. 1994. Flow-based programming. In *Proc. 1st International Workshop on Software Engineering for Parallel and Distributed Systems*. 25–29.
[93] Marzieh Mousavian, Jianhua Chen, Zachary Traylor, and Steven Greening. 2021. Depression detection from sMRI and rs-fMRI images using machine learning. *Journal of Intelligent Information Systems* 57, 2 (2021), 395–418.
[94] Gunter Mussbacher, Benoit Combemale, Silvia Abrahão, Nelly Bencomo, Loli Burgueño, Gregor Engels, Jörg Kienzle, Thomas Kühn, Sébastien Mosser, Houari Sahraoui, et al. 2020. Towards an assessment grid for intelligent modeling assistance. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 1–10.
[95] Gunter Mussbacher, Benoit Combemale, Jörg Kienzle, Silvia Abrahão, Hyacinth Ali, Nelly Bencomo, Márton Búr, Loli Burgueño, Gregor Engels, Pierre Jeanjean, et al. 2020. Opportunities in intelligent modeling assistance. *Software and Systems Modeling* 19, 5 (2020), 1045–1053.

[96] Franz Nachbar, Wilhelm Stolz, Tanja Merkle, Armand B Cognetta, Thomas Vogt, Michael Landthaler, Peter Bilek, Otto Braun-Falco, and Gerd Plewig. 1994. TheABCD rule of dermatoscopy: high prospective value in the diagnosis of doubtful melanocytic skin lesions. *Journal of the American Academy of Dermatology* 30, 4 (1994), 551–559.

[97] Farid Nakhle and Antoine L. Harfouche. 2021. Ready, Steady, Go AI: A practical tutorial on fundamentals of artificial intelligence and its applications in phenomics image analysis. *Patterns* 2, 9 (2021), 100323. https://doi.org/10.1016/j.patter.2021.100323

[98] Soroosh Nalchigar. 2020. *From business goals to analytics and machine learning solutions: a conceptual modeling framework.* Ph. D. Dissertation. University of Toronto (Canada).

[99] Alba Nogueira-Rodríguez, Hugo López-Fernández, Osvaldo Graña-Castro, Miguel Reboiro-Jato, and Daniel Glez-Peña. 2020. Compi hub: a public repository for sharing and discovering compi pipelines. In *International Conference on Practical Applications of Computational Biology & Bioinformatics*. Springer, 51–59.

[100] Azita Nouri, Philip E Davis, Pradeep Subedi, and Manish Parashar. 2021. Exploring the Role of Machine Learning in Scientific Workflows: Opportunities and Challenges. *arXiv preprint arXiv:2110.13999* (2021).

[101] Bentley James Oakes, Romain Franceschini, Simon Van Mierlo, and Hans Vangheluwe. 2019. The Computational Notebook Paradigm for Multi-paradigm Modeling. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. 449–454. https://doi.org/10.1109/MODELS-C.2019.00072

[102] Data Mining Fruitful & Fun Orange. [n. d.]. Orange Website. https://orange.biolab.si/.

[103] Perrine Paul-Gilloteaux, Sébastien Tosi, Jean-Karim Hériché, Alban Gaignard, Hervé Ménager, Raphaël Marée, Volker Baecker, Anna Klemm, Matúš Kalaš, Chong Zhang, et al. 2021. Bioimage analysis workflows: community resources to navigate through a complex ecosystem. *F1000Research* 10 (2021).

[104] Russell A Poldrack, Krzysztof J Gorgolewski, and Gael Varoquaux. 2018. Computational and informatics advances for reproducible data analysis in neuroimaging. *arXiv preprint arXiv:1809.10024* (2018).

[105] Luigi Quaranta, Fabio Calefato, and Filippo Lanubile. 2021. KGTorrent: A dataset of Python Jupyter Notebooks from Kaggle. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 550–554.

[106] Taylor Reiter, Phillip T Brooks, Luiz Irber, Shannon EK Joslin, Charles M Reid, Camille Scott, C Titus Brown, and N Tessa Pierce-Ward. 2021. Streamlining data-intensive biology with workflow systems. *GigaScience* 10, 1 (2021), giaa140.

[107] Philipp Ruf, Manav Madan, Christoph Reich, and Djaffar Ould-Abdeslam. 2021. Demystifying MLOps and Presenting a Recipe for the Selection of Open-Source Tools. *Applied Sciences* 11, 19 (2021), 8861.

[108] Adam Rule, Amanda Birmingham, Cristal Zuniga, Ilkay Altintas, Shih-Cheng Huang, Rob Knight, Niema Moshiri, Mai H Nguyen, Sara Brin Rosenthal, Fernando Pérez, et al. 2019. Ten simple rules for writing and sharing computational analyses in Jupyter Notebooks. , e1007007 pages.

[109] Vinícius W Salazar, João Vitor Ferreira Cavalcante, Daniel de Oliveira, Fabiano Thompson, and Marta Mattoso. 2021. BioProv-A provenance library for bioinformatics workflows. *Journal of Open Source Software* 6, 67 (2021), 3622.

[110] Khodakaram Salimifard and Mike Wright. 2001. Petri Net-based modelling of workflow systems: An overview. *European journal of operational research* 134, 3 (2001), 664–676.

[111] Aécio Santos, Sonia Castelo, Cristian Felix, Jorge Piazentin Ono, Bowen Yu, Sungsoo Ray Hong, Cláudio T Silva, Enrico Bertini, and Juliana Freire. 2019. Visus: An interactive system for automatic machine learning model building and curation. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. 1–7.

[112] Jacob Scharcanski and M Emre Celebi. 2013. *Computer vision techniques for the diagnosis of skin cancer.* Springer.

[113] S. Sendall and W. Kozaczynski. 2003. Model Transformation: The Heart And Soul Of Model-driven Software Development. *IEEE Software* 20, 5 (2003), 42–45. https://doi.org/10.1109/MS.2003.1231150

[114] Zohreh Sharafi, Yu Huang, Kevin Leach, and Westley Weimer. 2021. Toward an objective measure of developers' cognitive activities. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 3 (2021), 1–40.

[115] Rina Singh, Jeffrey A Graves, Valentine Anantharaj, and Sreenivas R Sukumar. 2019. Evaluating Scientific Workflow Engines for Data and Compute Intensive Discoveries. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 4553–4560.

[116] Saúl Solorio-Fernández, J Ariel Carrasco-Ochoa, and José Fco Martínez-Trinidad. 2020. A review of unsupervised feature selection methods. *Artificial Intelligence Review* 53, 2 (2020), 907–948.

[117] Patricia Centeno Soto, Nour Ramzy, Felix Ocker, and Birgit Vogel-Heuser. 2021. An ontology-based approach for preprocessing in machine learning. In *2021 IEEE 25th International Conference on Intelligent Engineering Systems (INES)*. IEEE, 000133–000138.

[118] Christopher Sutton, Luca M Ghiringhelli, Takenori Yamamoto, Yury Lysogorskiy, Lars Blumenthal, Thomas Hammerschmidt, Jacek R Golebiowski, Xiangyue Liu, Angelo Ziletti, and Matthias Scheffler. 2019. Crowd-sourcing materials-science challenges with the NOMAD 2018 Kaggle competition. *npj Computational Materials* 5, 1 (2019), 1–11.

[119] Christoph Tauchert, Peter Buxmann, and Jannis Lambinus. 2020. Crowdsourcing Data Science: A Qualitative Analysis of Organizations' Usage of Kaggle Competitions. In *Proceedings of the 53rd Hawaii international conference on system sciences*.

[120] Mehmet Tekman, Bérénice Batut, Alexander Ostrovsky, Christophe Antoniewski, Dave Clements, Fidel Ramirez, Graham J Etherington, Hans-Rudolf Hotz, Jelle Scholtalbers, Jonathan R Manning, et al. 2020. A single-cell RNA-seq Training and Analysis Suite using the Galaxy Framework. *bioRxiv* (2020).

[121] Guillaume Theaud, Jean-Christophe Houde, Arnaud Boré, François Rheault, Felix Morency, and Maxime Descoteaux. 2020. TractoFlow: A robust, efficient and reproducible diffusion MRI pipeline leveraging Nextflow & Singularity. *NeuroImage* 218 (2020), 116889.

[122] Curtis Thompson. 2020. Killer Shrimp 2nd Place Solution. https://www.kaggle.com/cwthompson/killer-shrimp-2nd-place-solution.

[123] Marko Toplak, Stuart T Read, Christophe Sandt, and Ferenc Borondics. 2021. Quasar: Easy Machine Learning for Biospectroscopy. *Cells* 10, 9 (2021), 2300.

[124] Wil MP Van Der Aalst and Arthur HM Ter Hofstede. 2005. YAWL: yet another workflow language. *Information systems* 30, 4 (2005), 245–275.

[125] Yves Vandenbrouck, David Christiany, Florence Combes, Valentin Loux, and Virginie Brun. 2019. Bioinformatics tools and workflow to select blood biomarkers for early cancer diagnosis: an application to pancreatic cancer. *Proteomics* 19, 21-22 (2019), 1800489.

[126] Jessica Velasco, Cherry Pascion, Jean Wilmar Alberio, Jonathan Apuang, John Stephen Cruz, Mark Angelo Gomez, Benjamin Molina Jr, Lyndon Tuala, August Thio-ac, and Romeo Jorda Jr. 2019. A smartphone-based skin disease classification using mobilenet cnn. *arXiv preprint arXiv:1911.07929* (2019).

[127] Markus Voelter, Bernd Kolb, Klaus Birken, Federico Tomassetti, Patrick Alff, Laurent Wiart, Andreas Wortmann, and Arne Nordmann. 2019. Using language workbenches and domain-specific languages for safety-critical software development. *Software & Systems Modeling* 18, 4 (2019), 2507–2530.

[128] Anthony Yu-Tung Wang, Ryan J. Murdock, Steven K. Kauwe, Anton O. Oliynyk, Aleksander Gurlo, Jakoah Brgoch, Kristin A. Persson, and Taylor D. Sparks. 2020. Machine Learning for Materials Scientists: An Introductory Guide toward Best Practices. *Chemistry of Materials* 32, 12 (2020), 4954–4965. https://doi.org/10.1021/acs.chemmater.0c01907

[129] Dakuo Wang, Q Vera Liao, Yunfeng Zhang, Udayan Khurana, Horst Samulowitz, Soya Park, Michael Muller, and Lisa Amini. 2021. How much automation does a data scientist want? *arXiv preprint arXiv:2101.03970* (2021).

[130] Yiping Wen, Junjie Hou, Zhen Yuan, and Dong Zhou. 2020. Heterogeneous information network-based scientific workflow recommendation for complex applications. *Complexity* 2020 (2020).

[131] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data* 3, 1 (2016), 1–9.

[132] Julia K Winkler, Christine Fink, Ferdinand Toberer, Alexander Enk, Teresa Deinlein, Rainer Hofmann-Wellenhof, Luc Thomas, Aimilios Lallas, Andreas Blum, Wilhelm Stolz, et al. 2019. Association between surgical skin markings in dermoscopic images and diagnostic performance of a deep learning convolutional neural network for melanoma recognition. *JAMA dermatology* 155, 10 (2019), 1135–1141.

[133] Laura Wratten, Andreas Wilm, and Jonathan Göke. 2021. Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers. *Nature methods* 18, 10 (2021), 1161–1168.

[134] Likang Wu, Zhi Zheng, Zhaopeng Qiu, Hao Wang, Hongchao Gu, Tingjia Shen, Chuan Qin, Chen Zhu, Hengshu Zhu, Qi Liu, et al. 2023. A Survey on Large Language Models for Recommendation. *arXiv preprint arXiv:2305.19860* (2023).

[135] Yinhao Wu, Bin Chen, An Zeng, Dan Pan, Ruixuan Wang, and Shen Zhao. 2022. Skin Cancer Classification With Deep Learning: A Systematic Review. *Frontiers in Oncology* 12 (2022).

[136] Moe Thandar Wynn, HMW Verbeek, Wil MP van der Aalst, Arthur HM ter Hofstede, and David Edmond. 2009. Business process verification–finally a reality! *Business Process Management Journal* (2009).

[137] Yuanshun Yao, Zhujun Xiao, Bolun Wang, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. 2017. Complexity vs. performance: empirical analysis of machine learning as a service. In *Proceedings of the 2017 Internet Measurement Conference*. 384–397.

[138] Chaoning Zhang, Philipp Benz, Dawit Mureja Argaw, Seokju Lee, Junsik Kim, Francois Rameau, Jean-Charles Bazin, and In So Kweon. 2021. ResNet or DenseNet? introducing dense shortcuts to ResNet. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 3550–3559.

[139] Guoqing Zhou, Ben Nebgen, Nicholas Lubbers, Walter Malone, Anders MN Niklasson, and Sergei Tretiak. 2020. Graphics processing unit-accelerated semiempirical Born Oppenheimer molecular dynamics using PyTorch. *Journal of Chemical Theory and Computation* 16, 8 (2020), 4951–4962.