Marcelle: Composing Interactive Machine Learning Workflows and Interfaces

Jules Françoise, Baptiste Caramiaux, Téo Sanchez

Marcelle: Composing Interactive Machine Learning Workflows and Interfaces

JULES FRANÇOISE, Université Paris-Saclay, CNRS, Laboratoire Interdisciplinaire des Sciences du Numérique, 91400, Orsay, France

BAPTISTE CARAMIAUX, Sorbonne Université, CNRS, ISIR, 75005, Paris, France

TÉO SANCHEZ, Université Paris-Saclay, CNRS, Inria, Laboratoire Interdisciplinaire des Sciences du Numérique, 91400, Orsay, France

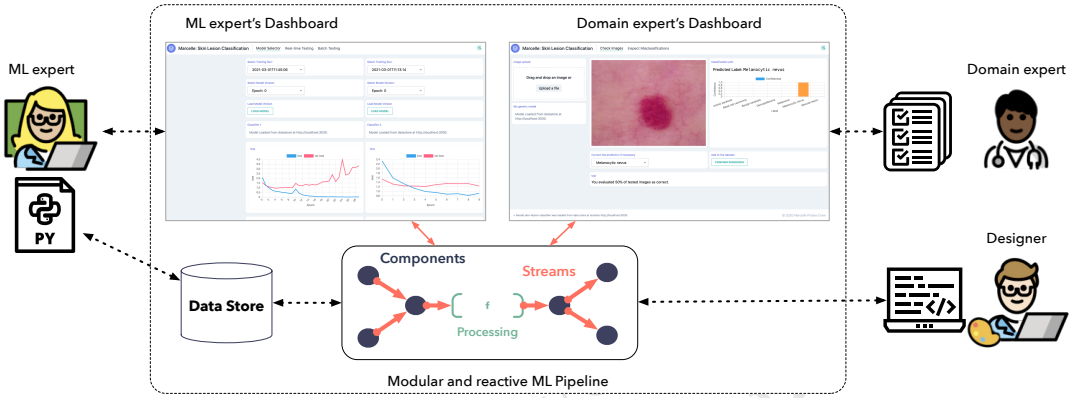

Fig. 1. Marcelle is a toolkit for interactive machine learning addressing the composition of custom workflows. It implements a component-based architecture using reactive programming for pipeline specification. Components provide views that can be composed to form custom interfaces. Marcelle's architecture facilitates collaboration between machine learning experts, designers and end-users.

Human-centered approaches to machine learning have established theoretical foundations, design principles and interaction techniques to facilitate end-user interaction with machine learning systems. Yet, general-purpose toolkits supporting the design of interactive machine learning systems are still missing, despite their potential to foster reuse, appropriation and collaboration between different stakeholders including developers, machine learning experts, designers and end users. In this paper, we present an architectural model for toolkits dedicated to the design of human interactions with machine learning. The architecture is built upon a modular collection of interactive components that can be composed to build interactive machine learning workflows, using reactive pipelines and composable user interfaces. We introduce Marcelle, a toolkit for the design of human interactions with machine learning that implements this model. We illustrate Marcelle with two implemented case studies: (1) a HCI researcher conducts user studies to understand novice interaction with machine learning, and (2) a machine learning expert and a clinician collaborate to develop a skin cancer diagnosis system. Finally, we discuss our experience with the toolkit, along with its limitation and perspectives.

CCS Concepts: • **Human-centered computing** → **Interactive systems and tools**; **HCI theory, concepts and models**; • **Computing methodologies** → *Machine learning*.

Additional Key Words and Phrases: Interactive Machine Learning, Machine Teaching, Architectural Model, Toolkit.

1 INTRODUCTION

Machine Learning (ML) system development is still a matter for specialists. App designers struggle to include ML in their design practice. Domain experts, such as clinicians, have little control over the development of the systems

providing them assistance in decision making. Small businesses developing ML solutions may find it difficult to provide their clients with adaptable tools for calibrating and evaluating models on real-world data. With the growing use of machine-learned models in user-facing applications, it is essential to include a broad spectrum of stakeholders in the process of building and assessing such systems.

Interactive Machine Learning (IML) is a paradigm engaging people in a tight interaction loop with a learning algorithm [17, 32, 33, 40]. In this approach, users refine a model through an iterative process involving several activities, from data acquisition to model training, testing and deployment [32]. Some of the benefits of IML include helping users to efficiently convey expert knowledge to the system [43], allowing system adaptation to a user's specific needs [35], enabling ML as design material [31], or helping users to better understand caveats and strengths of a ML pipeline [42]. IML research is burgeoning: it has given rise to novel interaction techniques, original visualisation strategies, as well as important guidelines for designing IML systems and Human-AI Interactions (HAI) [19, 28, 51, 64, 68]. Systems and studies explicitly adopting a human-centered perspective on ML illustrate the need to support diverse workflows in ML practice. Yet, IML software is often specialized for particular domains [9, 23, 35] or tasks [4, 27, 33, 36] and cannot readily be adapted to different practices. To our knowledge, IML is still missing general-purpose toolkits supporting the design of such systems, that would foster reuse and appropriation as well as collaboration between different stakeholders.

In this paper, we propose a software architecture for toolkits dedicated to the design of human interactions with ML systems. Our goal is to facilitate the development of such interactions for a broad range of users such as designers, domain experts, creatives, or ML researchers. In the proposed architecture, developers can **compose** interactive machine learning **workflows** by assembling **components**. Components provide graphical user interfaces that can be easily composed into custom layouts, and their reactive data streams can be composed into reactive pipelines that link user interactions with machine learning processes. This approach emphasizes flexibility in the definition of workflows, enabling rapid prototyping of pipelines and interfaces that can be tailored for a large diversity of end users.

Our contribution in this paper is twofold. First, we present an architectural model for composing interactions with ML systems. Second, we present an implementation of the model in the form of a programming toolkit, called *Marcelle*. Marcelle provides a high-level JavaScript API allowing developers to quickly prototype and distribute interactive applications. We illustrate Marcelle with two implemented case studies: (1) a HCI researcher conducts user studies to understand how novices interact with ML, and (2) a ML expert and a clinician collaborate to develop a skin cancer diagnosis system.

2 COMPOSING INTERACTIONS WITH ML

Our vision of **composable** human-ML interactions relies on toolkits that are *flexible*, *integrable* in existing practices and supporting *collaboration*. Flexibility means that developers should be able to easily customize how they interact with elements of a ML pipeline (datasets, models, metrics, etc.) during development, and how end-users may interact with these elements. The need for flexibility concerns both workflows, that structure activities and their relationships, and specific interaction techniques, where a careful design of interfaces is essential to provide appropriate feedback and interaction mechanisms. Such tools should be seamlessly *integrated* in the existing practices of developers and end-users. Typically, tools should be easily shared and integrated in diverse practices, from ML experts developing models in Python using dedicated ML libraries to domain experts with context-specific constraints (e.g. using the system in a medical environment). The development process should support *collaboration* between several stakeholders, for example different developers, designers and end-users.

We propose an architectural model to achieve this objective for the design of human interactions with machine learning. The architecture is built upon a modular collection of interactive machine learning **components** with a unified interface, that can be **composed** to form custom processing pipelines and user interfaces. This component-based architecture is **extensible** and facilitates **reuse** of interaction techniques across projects. The architecture is built over **web technologies** to facilitate collaboration, and supports **sharing** of applications, data and models.

To motivate this approach, we describe two case studies. The first case study originates in our own research experience [56]. The second case study stems from recent findings on human-ML collaboration in the medical field [63]. The scenarios of the two case studies are depicted in Figures 2 and 3, respectively.

Scenario 1 *Studying Human Interaction with Machine Learning.* Suzanne is an associate professor. Her research focuses on democratizing ML systems for the general public. To that end, she runs workshops and studies with playful scenarios where participants can teach concepts to a classifier and she collects data on user interactions. One of these scenarios involves input sketches drawn by people. Due to the sanitary restrictions, she needs to run her study remotely, while collecting data from participants.

She starts by prototyping a simple sketch-based classification workflow using a single JavaScript script. She instantiates a few standard *components* of the machine learning process: a dataset with a recording interface, a classifier, and a visualization module for its predictions. She *reuses* a sketchpad component from a previous project, and *extends* her toolkit by creating a component to visualize the model's uncertainty in order to help people to understand how well the classifier is trained. She *composes* these components together to form a processing pipeline, and assembles them in a graphical user interface. She *shares* her application as a simple web page in an online workshop with novices. Their feedback leads her to quickly update her script so that the pipeline and visual layout provides instantaneous feedback after each stroke and supports incremental learning. Her prototype now ready for the study, she sets up a *data store* to record the participants' sketches and models, along with logs of the interactions with the system.

Her experiments show that her design is successful in supporting ML understanding by novices. Later on, she starts collaborating with Shan, a designer, to deploy a version of the application in a science popularization exhibition. Shan proposes and implements a new visual interface using a web framework, which he later links with Suzanne's machine learning pipeline by removing the default app interface from her script, without affecting its functioning.

Scenario 2 *Collaborating with domain experts.* Louise is a newly recruited ML engineer working on state-of-art image classification models for skin cancer diagnosis. She collaborates with Michel, a clinician, to assess the real-world performance of the developed methods. Louise uses her usual Python scripts and logs data through a dedicated logger compatible with the interface she wants to build. On the interface, she composes a few standard components in a *dashboard* to visualize training (e.g. loss curves) and model predictions (e.g. intermediate confusion matrices, direct testing with hand-picked instances). To share her progress, she creates a simpler dashboard for Michel, that reuses the prediction pipeline without details about the training. It allows Michel to test the quality of the classification with his own images. The application runs in a web browser, and *synchronizes* Louise's model updates with Michel's interface. It also enables Michel to correct misclassifications and to store them in order to provide Louise with additional data and annotations.

3 RELATED WORK

In this section we outline background work in IML, and we analyze existing systems and toolkits for ML and IML according to our design considerations highlighted in the scenarios of two case studies.

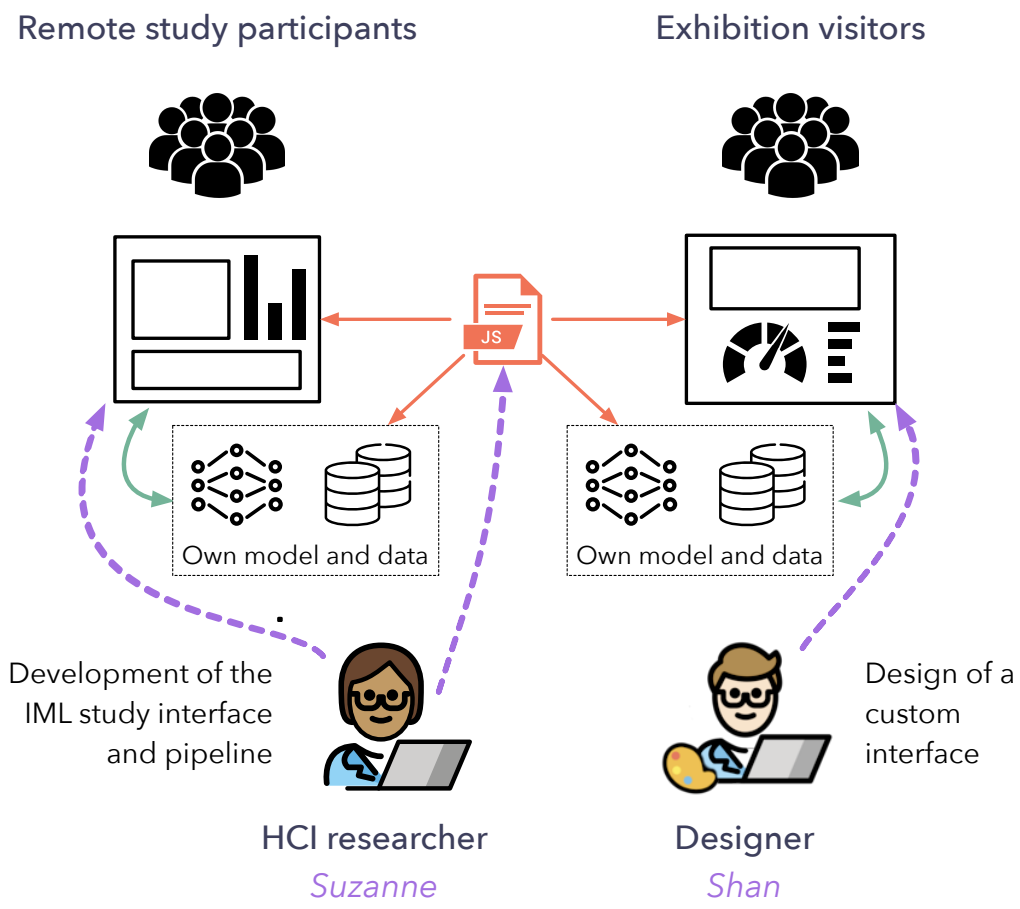

Fig. 2. Summary of the first scenario. The interfaces made in this scenario are dedicated to both remote study participants and exhibition visitors. The HCI researcher Suzanne developed the interface for remote study participants and the ML pipeline (dashed purple arrow). The designer developed the interface for exhibition visitors and reuses the ML pipeline made by Suzanne. Each application has their own data and models (green arrows).

3.1 Background work in Interactive Machine Learning

Research in Interactive Machine Learning has highlighted the benefit of bringing human input in the training process of machine learning systems, leading to the development of original iterative workflows and interaction techniques [17, 32].

IML workflows differ from conventional machine learning workflows in involving shorter cycles of data edition, training and evaluation. Such workflows have been shown to produce more robust models in diverse tasks such as image segmentation [33], web image search [36], or text classification debugging [47]. Aside from improving model performance, IML workflows foster exploration and discovery, which makes it attractive for creative and artistic practices where the learning process may become more important than the learned model itself [57]. Such an approach has been used in digital musical instrument design [26, 34, 35, 37] and movement-based interaction design [39].

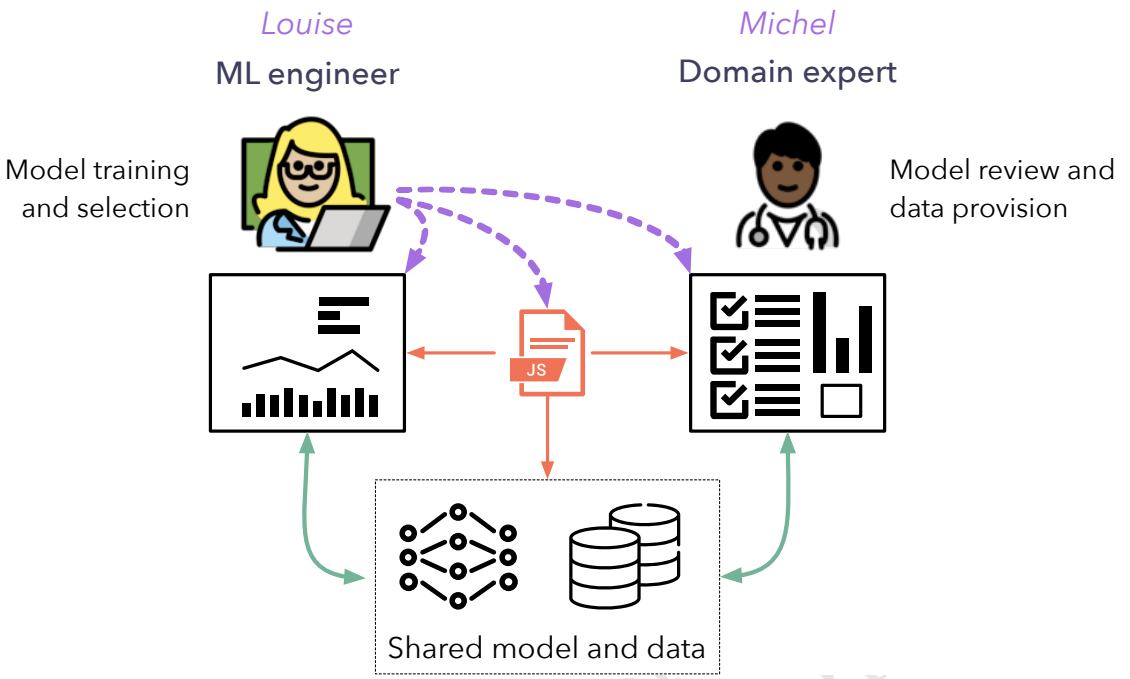

Fig. 3. Summary of the second scenario. The two interfaces are dedicated to ML engineer Louise and the clinician Michel. Louise developed the ML pipeline and both interfaces (dashed purple arrows). Both dashboards share the same model and data (green arrows) to enable collaboration between Louise and Michel.

IML research also gave rise to original interaction techniques. Iterative and exploratory workflows make user's action history [42] and model's state tracking [18] particularly important in ML development. Building ensemble models interactively is faster and as robust as using optimization [62]. Other works have investigated how interpretable feedback on predictions can help users understand and steer a model towards better performance [47, 61]. We refer to Dudley and Kristensson [32] for a comprehensive survey of interface design in IML.

Finally, previous works proposed guidelines to build better IML systems, where quality can be assessed in terms of model performance, user's understanding of the system, or user experience [16]. Dudley and Kristensson [32] identified 6 activities that have to be taken into account within an IML workflow from a behavioral perspective: (1) feature selection, (2) model selection, (3) model steering, (4) quality assessment, (5) termination assessment and (6) transfer. While this list of activities covers the scope of a comprehensive IML workflow, many approaches address a single activity (or a subset of activities), and specific systems are developed in this objective. In the next section, we review a number of IML systems and tools that are publicly available.

3.2 Interactive Machine Learning Tools

Several tools following the IML paradigm have been proposed to assist in model development by non-experts. Teachable Machine [27] and Lobe [4] are standalone applications offering a simple interface to create machine learning classifiers from examples. They require neither programming nor ML knowledge, and can be used as educational ressources [20].

They are mostly dedicated to developers willing to integrate image classification in application. While they facilitate model development, assessment and transfer, they implement a fixed workflow and cannot be customized.

More modular IML tools have been proposed to support creative practices. RunwayML [9] presents itself as a Photoshop-like tool that integrates many state of the art deep learning models. To that extent, RunwayML is less of a toolkit than a complete software tool for ML-assisted creative practices. Wekinator [35] is a standalone application dedicated to the design of new instruments and gesture-based interaction for music performance. It integrates general-purpose models that can be used with a wide variety of inputs and outputs. However, it provides limited visualization and fixed workflows. There exist libraries integrating ML algorithms into computer music programming platforms such as Cycling'74 Max [13], including either general-purpose models as in ml.lib [25] or specialized techniques as in XMM [38]. InteractML [30] is a visual programming extension for Unity3D dedicated to game developers and designers willing to explore embodied interactions. Its modular architecture facilitates the creation of custom workflows for end-users. Finally, the RapidMix API [24] is a JS library for rapid prototyping of creative applications embedding machine learning methods. The API allows for creating custom ML pipelines in the browser but it does not allow for composing user interfaces.

Even though there are many examples of tools with diverse workflows integrating specific interaction techniques, to the best of our knowledge no general-purpose toolkit exists for building standalone IML applications. Developers must dedicate significant resource to developing systems using machine learning frameworks and visualization tools.

3.3 Machine Learning Frameworks

A multitude of modular frameworks has emerged in the machine learning community to make ML more accessible for developers and data scientists. Available as libraries, these tools provide a high-level API for training and evaluating ML models, either with general-purpose algorithms as in Scikit-Learn [54], or specialized for deep learning, for instance with Tensorflow [14] or Pytorch [53]. These libraries still require ML expertise in addition to programming knowledge, and focus on creating training and evaluation pipelines with limited interactivity.

There has been a recent effort to bring ML to web browsers through JavaScript libraries. ONNX.js [49] is a JavaScript library for running inference on Open Neural Network Exchange (ONNX) models in web browsers. The Tensorflow.js [60] library enables inference and training of neural networks in web browsers. Its API is close to Keras [3], and it provides compatibility with models trained with Python. Other libraries such as ml.js [6] and ml5 [5] were designed to provide a high-level API for developers and designers. While these tools can power the development of interactive machine learning systems, they do not explicitly support the design of interactions with ML pipelines.

Finally, there also exist toolkits for data science emphasizing the creation of custom workflows for data analysis tasks. Orange [29] and RapidMiner [41] provide a visual programming environment that allows users to create data analysis pipelines. However, their focus is on data analysis rather model refinement, and they are not designed to enable custom end-user interaction.

3.4 Visualization Toolkits for Machine Learning

With the rising interest for visual analytics as a support for ML practice [70], a number of visualization toolkits have been proposed to support model development, debugging and understanding. Examples include Tensorboard [67], ClearML [15], Neptune.ai [8], VisualDL [11] or Visdom [2]. Tensorboard [67] is a visualization toolkit for Tensorflow, running in the browser, dedicated to facilitate machine learning experimentation. Its modular architecture provides flexibility regarding the information to display, that includes metrics, model graphs and assets. Notebook environments

such as Jupyter [46] provide literate programming environments emphasizing narratives. Notebooks support a wide range of visualization libraries, are easily shared, and therefore foster reuse and appropriation. Tensorwatch [58] emphasizes real-time monitoring in Python notebooks. It allows users to query about the training process and produce real-time visualisation. While these tools bring interactivity to the workflows of ML practitioners, they are dedicated to ML monitoring and assessment. Their focus is on computation, whereas our approach emphasizes direct interaction for a larger range of activities. Moreover, they do not allow the development of standalone applications.

Summary. While there exist toolkits for machine learning and visualization, interactive machine learning systems are often domain-specific. To our knowledge, general-purpose toolkits for designing IML applications are still missing, along with defining principles for their architecture.

4 DESIGN PRINCIPLES

We present 5 design principles that define an architecture model enabling composing interactions with ML pipelines, as described in Section 2. The proposed architecture is illustrated in Figure 4. The implementation in *Marcelle* presented in Section 5 follows these principles.

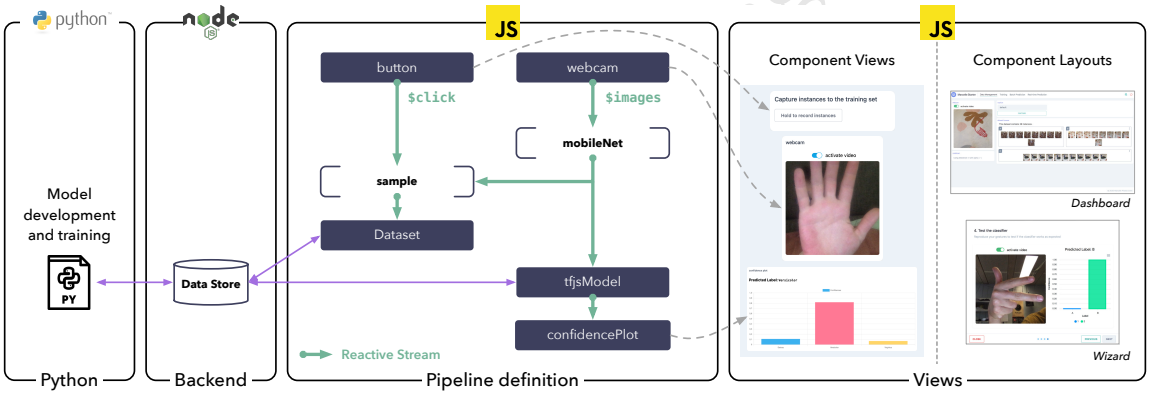

Fig. 4. The proposed architectural model relies on loosely coupled components (dark blue boxes) embedding data, processing and interaction. Components expose event streams that serve as interface to build reactive pipelines (green arrows). Stream processing can be performed outside components (outlined boxes). Components optionally provide views that can be displayed on demand (right panel), offering users ways to interact with machine learning pipelines. These views can be visually composed through layout mechanisms (right panel). Data stores offer persistent storage to components and enable bidirectional communication with Python.

4.1 Component-based Architecture

Building interactive ML application requires assembling interactions to facilitate the manipulation of machine learning concepts and objects. These objects of interest are highly heterogeneous in nature and relate to various activities, such as the ones reported by Dudley and Kristensson [32]: feature selection, model selection, model steering, quality assessment, termination assessment and transfer (i.e. deployment). Depending on the activity, users might need to operate upon various objects: data, algorithms, parameters, models, predictions, explanations, etc. In addition, as the needs of users can be dramatically different, their actions upon the ML pipeline need to be supported by a custom arrangement of interfaces that facilitate the manipulation of these objects.

We propose a modular architecture where the building blocks for designing interactions are *components* embedding data, computation, and interaction. Components enable granular interaction with specific elements of the ML pipeline,[1] and can be flexibly composed to form higher-level interactions and workflows that support the aforementioned activities. Components can typically be data sources (e.g. capturing images from a webcam, uploading files, recording user sketches), data structures (e.g. a dataset used to store training examples), visualizations (e.g. to navigate through a dataset or to visualize predictions), computations (e.g. model training or prediction), or a combination. To emphasize instant feedback and interaction, most components should provide a graphical user interface that can be displayed on demand. Components should implement a unified interface (in the sense of interface-based programming) defining how they can be connected and visualized, without enforcing particular patterns or libraries to program their internal behavior.

4.2 Interaction-Driven Pipelines

While in a typical ML setup the workflow for training and testing might be standardized, interactive machine learning applications involve custom workflows where various types of processing are triggered by user interactions [18, 35, 42, 57, 62]. It is therefore essential to let developers create custom pipelines specifying complex relationships between the user's actions (e.g. capturing a new instance) and the resulting processing (e.g. adding it to a dataset, training a model, updating predictions, etc.). *Reactivity* is key to handle diverse workflows where event streams of heterogeneous nature must be interconnected.

We propose a design where developers can flexibly specify the relationships between components to form *reactive pipelines* that propagate change. Components share a common minimal interface that enables them to expose arbitrary event streams to facilitate communication. Event streams can be observed, filtered, transformed or combined anywhere inside or outside components. Components can therefore observe streams and react to change in other components, they can process streams or create them. Altering streams outside components provides a powerful means for customizing the processing chain within ML pipelines.

4.3 Composable Interfaces

Workflows encompass two main facets: the specification of reactive pipelines describing the relationships between various objects and actions, as described in the previous section, and the visual arrangement of components in the end-user interface. In their review of user interface design for IML, Dudley and Kristensson [32] underline that while there exist common elements, the design of IML interfaces vary considerably according to the data and application. The proposed component specification includes an optional *view* allowing for visualization and enabling user interactions. However, the way views are used in an application (e.g. where and when they are displayed) should not affect the logical relationships between components, that are defined using reactive pipelines.

We propose that components should be displayed on demand. In practice, this means that components' computation should not be affected when their view is hidden (besides preventing from user interactions), and that multiple views could potentially be used to interact with the same component. Components should easily be displayed on their own in web applications, and higher-level composition mechanisms for generating layouts should be provided to further facilitate development.

[1] In our context, granularity means the different level at which the user can interact with the system: from high level class names in a classification task, to the low-level number of neurons in a specific layer of the network for example.

4.4 Data Persistence and Communication

In a collaborative scenario involving users with diverse levels of expertise in ML, it is essential that the objects of various types contained in the application are shared among collaborators (datasets, annotations, models, predictions, logs, etc.). We propose a design that supports flexible data persistence and sharing, allowing developers to specify where to store and how to share various parts of an application's state, including training or test data, models, logs, etc. We propose the use of *data stores*, accessible to components, that are generic enough to store heterogeneous data describing the application's state. Data stores should provide a unified interface while allowing data storage in diverse locations, including remote servers and browsers' web storage.

4.5 Interoperability with ML Libraries

Machine learning practice now relies on a key set of programming languages and libraries that are widely used among researchers and engineers, as reported in Section 3.3. Among them, Python is particularly popular, with libraries such as Scikit-Learn [54], Tensorflow [14] or Pytorch [53], to name a few. The architecture needs to provide interoperability with machine learning frameworks in order to be used by machine learning experts. It is essential to provide an interface to communicate data and models between Python programs and components. For instance, enabling access to data stores from third-party programs would help creating bridges between programming environments. We suggest that architectures supporting interoperability should seamlessly share data and models, but also computational pipelines, including, for instance, feature extraction and preprocessing.

5 IMPLEMENTATION

Marcelle is an IML toolkit that implements the design principles defined in the previous section. Marcelle is distributed as open-source software and available online.[2] The toolkit, written in TypeScript, is distributed as a client-side JavaScript library that can be integrated in web applications or used in standalone. The core library (`marcellejs/core`) is composed of a set of definitions specifying a concrete implementation of the architecture model. It also provides a set of standard components, a flexible data store system, and two mechanisms for interface composition. Marcelle also provides a server package for persistent data storage, a command-line interface for project generation, and a Python package allowing communication with Tensorflow. The overall principle of the library is illustrated in Figure 1 and its architecture is depicted in Figure 4.

5.1 IML Components

Components are the building blocks of Marcelle applications. They embed the state, logic and interaction for particular tasks. They implement a minimal interface enabling visualization and communication with other components. A component is essentially a JavaScript object that (1) exposes a set of reactive streams that can be processed by other components (as described in Section 5.2), and (2) provides methods to display the component's graphical user interface in the DOM.

Components are versatile in scope and can address a large variety of tasks. We classified standard components into 5 categories:

Input components address data acquisition from different sources. For images, this includes recording images from a webcam, from file upload, or from a drawing canvas.

[2] https://marcelle.dev/

Data management components concern data storage, processing and visualization. This includes datasets for storing training or test instances, predictions, or arbitrary logged data.

Models are special components implementing specific machine learning models. They implement a specialized interface for training, inference and import/export to files and data stores. Marcelle currently integrates models from the Tensorflow.js [60] library, but other JavaScript libraries can easily be used.

Visualization tools concern interactive visualization of various objects in the pipeline, such as datasets, model parameters, training progress, or predictions.

Widgets are standard GUI widgets (buttons, menus, input fields, etc.) that are necessary to compose custom user interfaces.

Components often provide a graphical user interface, or *view*, that can be displayed on demand in a web application, using `.mount()` and `.destroy()` methods. Examples of views of components are included in the dashboards presented in Figure 1. Views only communicate with the component using streams: they are reactive to changes but can also push events into the component's streams. This mechanism provides a clear separation between the view and the component's processing. In other words, a component remains functional in a given pipeline even if its view is not displayed, since it is possible to imperatively put events into any component's streams.

We use Svelte [10] to power the views in Marcelle's component library. Svelte is a compile-time framework with native support for reactive streams. However, Marcelle does not enforce the use of any framework for creating views of custom components. Creating custom components only requires creating JavaScript objects that expose streams and provide a `.mount()` method to display the view. A developers is free to use Svelte or any other framework to generate views and can use any pattern for generating components, including object literals, factory functions or classes.

5.2 Reactive Pipelines

Reactive pipelines are concerned with the specification of the relationships between user actions and machine learning objects. We propose a distributed model that gives developers explicit control over the information flow, offering flexibility for building applications where user actions must trigger complex sequences of operations. Our approach relies on reactive programming [21], a paradigm that is well-suited for the development of IML applications, that are essentially event-driven. It facilitates the creation, filtering, transformation and consumption of asynchronous data streams that propagate changes over the pipeline. Reactive programming enables encapsulated, loosely coupled modules where event streams act as an interface. By avoiding callbacks, it provides more readable and maintainable code.

Components achieve a large variety of tasks, and, from a reactive programming perspective, can broadly be classified into 3 categories:

Sources are components producing data. Examples of source components include GUI widgets and data input tools.

Sinks are components that respond to event streams without necessarily producing new events, and are mostly used to provide users with feedback.

Processors are both sinks and sources: they rely on input events to produce a set of output streams. This is the broadest category of components and includes feature extractors, models, or datasets.

Our implementation of reactive streams relies on Most.js [7], a high performance reactive programming library. Components expose streams as properties whose name is prefixed with a dollar sign by convention. For instance, a

`webcam` component exposes several streams, including a boolean stream called `$active` specifying whether the webcam is turned on, and a periodic stream called `$images` containing images sampled from the webcam in `ImageData` format.

5.3 Dashboard and Wizard Layouts

Since components provide their own views, creating user interfaces that are tailored for a particular application or user is straightforward. Developers can mount any component to a given element in the DOM. To simplify interface design, Marcelle provides two high-level mechanisms for building user interfaces: *Dashboards* and *Wizards*.

5.3.1 Dashboards. Dashboards provide a way to create applications with multiple pages that display collections of components. Dashboards provide an interface similar to Tensorboard [67]. Dashboards are instantiated using the `dashboard` factory function. They are composed of pages, instantiated with the `.page()` method, that can display any component instance passed to their `.use()` method. Dashboards are hidden by default and can be displayed on demand as a full-screen overlay over the current web page. In order to avoid disturbing the host application when displayed on demand, we use hash-based routing for navigating between the various pages. Navigating to a page will mount its associated components' views. Dashboards also expose streams containing events describing their state: `$visible` specifies whether the dashboard is currently visible, and `$page` is mapped to the current page.

5.3.2 Wizards. Wizards are dedicated to the creation of walk-through guides for beginners or end-users. Wizards are inspired by Teachable machine's *training wizard*[3] that walks users through training their machine learning model. Marcelle wizards are flexible and allow developers to specify what components should be displayed at every step. Our implementation of wizards is similar to that of dashboards: they are instantiated with a factory function and contain a number of panels. Panels are created using the `.page()` method, and contain a title, a textual description and a set of components to display. Wizards are also displayed on demand in a modal window. They expose a stream called `$current` synchronized with the index of the current panel.

5.4 Data Stores

While reactive programming facilitates real-time data communication, most scenarios also require data persistence so that parts of the application's state are stored. For instance, the training data provided by the user should persist, even when changes to the pipeline are made. Marcelle provides flexible *data stores* that can be instantiated with various backends: data can be stored in the browser's local storage or on a remote server. Choosing the backend location only requires passing a URL to the data store. Developers can create different backends to customize where different objects are stored. Some components rely on a data store — for instance, the `dataset` component that needs to store instances, — however data collections can be created on the fly to store custom information when relevant. This is particularly useful to store some of the state of the application (for instance the model's parameters) or session logs of the user's interactions.

Marcelle's backend package (`marcellejs/backend`) provides a simple Node.js server that can be easily configured to use either NeDb or MongoDb databases for storage, with optional authentication. We use the Feathers [1] framework for managing data stores, both on the server and client-side. In development mode, Feathers services are created dynamically when required by the client. They do not require the specification of a database schema, and can therefore

[3] Online demo: https://glitch.com/~tm-wizard

flexibly handle custom data structures. Feathers provides real-time updates, enabling components to react to changes in data stores, even if they come from other clients.

5.5 Python Integration

Training and running inference on ML models in web browsers is possible with dedicated JavaScript libraries. Using Tensorflow.js [60], it is possible to make real-time predictions with potentially large models with several million parameters. Marcelle's dataset architecture is optimized for training, using asynchronous iterators that can stream and process data lazily. Yet, the limited computing power of current web browsers harms scaling to larger datasets and models.

Marcelle partially supports interoperability with standard machine learning frameworks in Python. This gives Marcelle the capacity to scale easily according to the developer's computing resources. The `marcelle` Python package can be used to interact with a backend server, with read and write access to data stores. Our implementation currently supports Tensorflow, a common Python library for Deep Learning development. Complex neural network models can therefore be trained with large datasets from Python. Data can be logged to the backend during training, along with model checkpoints stored in Tensorflow.js format, and additional generic assets such as images or audio files (according to the type of data involved in modeling). For Keras, we provide a callback object that can simply be passed to a model's `.fit()` method. For custom training loops in Tensorflow, we provide a `Writer` where logging can be done imperatively. Marcelle components can then easily access the logged data in the data store, for instance to plot loss and accuracy curves, or to load models at various checkpoints to perform inference in an interactive application.

5.6 Command-Line Interface

Marcelle comes with a Command-Line Interface (CLI) to generate new projects, custom components and backends. We use Vite [12] as a default build tool for projects. Vite is easily configurable, supports multi-page applications and provides a very fast development server with hot module reloading, which is convenient for prototyping. Additionally, the CLI can generate new components and backend services for existing projects.

6 CASE STUDY 1

This section describes how Marcelle can be used to implement Scenario 1, introduced in Section 2, and discusses the benefits of the architecture to support this task. A running demo of the case study is available online[4].

6.1 Implementing the Scenario

6.1.1 Setting up the Application Structure and Interface. Suzanne creates her application using Marcelle's CLI. Then she edits the `src/index.js` file and instantiates the necessary components from the `marcellejs/core` library: a `sketchPad` input, a `mobileNet` component to process input sketch images[5], a `dataStore` specifying where to keep the data, a `dataset`, a `mlpClassifier` (multi-layer perceptron classifier). These components are not interconnected yet and widgets are missing to enable the necessary user interactions. Suzanne needs a text input widget to enter the class labels and a button to add a sketch to the training set. She instantiates two new components: a `textField` and a

[4] https://uist2021demos.marcelle.dev
[5] This follows a common transfer learning strategy (for more details on this approach, see for instance [69]). Mobilenet is commonly used model for pre-training [44].

`button`. Then, she adds a second `button` to trigger model training. Finally, she adds a third `button` to trigger current instance prediction.

To build her app's interface, Suzanne instantiates a `dashboard`. Using the `.use()` method, she adds the desired elements on the interface: the sketchpad, the text field, the two buttons. She also adds a `datasetBrowser` component to monitor the examples added to the training set. The resulting application is generated by a script containing about 20 lines of code, as reported in Listing 1.

```
// Main components
const input = sketchPad();
const featureExtractor = mobileNet();
const store = dataStore('localStorage');
const trainingSet = dataset('TrainingSet', store);
const classifier = mlpClassifier({ layers: [64, 32], epochs: 20 });

// Additional widgets and visualizations
const classLabel = textField();
const captureButton = button({ text: 'Capture this drawing' });
const trainButton = button({ text: 'Train the classifier' });
const predictButton = button({ text: 'Predict label' });
const trainingSetBrowser = datasetBrowser(trainingSet);

// Dashboard definition
const myDashboard = dashboard({ title: 'Sketch App', author: 'Suzanne' });

myDashboard
  .page('Main')
  .sidebar(input)
  .use([classLabel, captureButton], trainingSetBrowser)
  .use(trainButton, predictButton);

myDashboard.start();
```

Listing 1. Script containing component definitions and interface composition for the interactive sketch recognition application. Components are instantiated using factory functions, and can be composed into a dashboard.

6.1.2 Connecting the Pipeline. Having the structure of her application, Suzanne creates the pipeline using the components' reactive streams. As an example, we report a simple prediction pipeline in Listing 2a. Here, Suzanne wants to plot the classifier predictions as a bar chart. She uses a `classificationPlot` component that takes a prediction stream as input. Using reactive programming operators, Suzanne creates a stream of predictions from the input sketches. She starts by sampling the `$images` stream of the `sketchpad` with the button `$click` stream, and passes the images to the

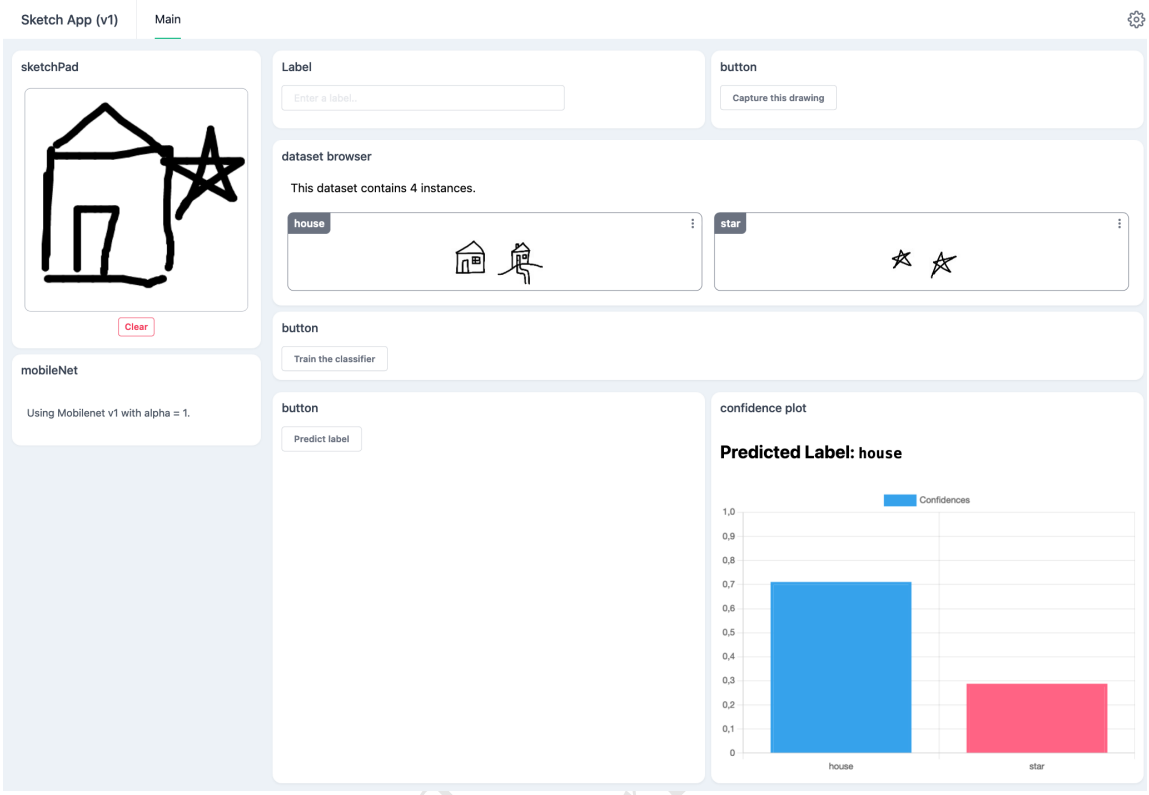

Fig. 5. Visual interface of the initial prototype for Case Study 1. This dashboard is generated by Listing 1.

feature extractor. The resulting `$features` stream is then fed to the classifier to create a stream of predictions that can be visualized. The initial prototype for the sketch application is shown in Figure 5.

6.1.3 Improved Workflow. After sharing her application as a web page and gathering preliminary feedback about the system, Suzanne chooses to improve the workflow by providing instantaneous feedback after each stroke drawn by end-users and by supporting incremental learning. In order to do this, Suzanne only has to edit the script by adapting how the prediction stream is built. Concretely, predictions should be updated at each stroke on the sketchpad, but also when the model is updated after training. The predicted label should also pre-fill the label input to facilitate data annotation. As an example, the updated prediction pipeline is reported in Listing 2b. Finally, she rearranges the layout of the components in the dashboard.

Finally, Suzanne can deploy her app online and log data on a dedicated server by changing the data store's location from `localStorage` to the URL of a remote server. She uses the CLI to generate this server.

6.1.4 Collaboration with a Designer. After successfully conducting the study, the application is reused as a installation in a museum exhibition for scientific popularization. Shan is included in the project and works on a new visual interface using the web framework he is familiar with. The separation between the ML pipeline and the visual interface in Marcelle facilitates the integration of the Shan's designed interface on top of the existing ML pipeline.

```
const $features = predictButton.$click
  .sample(input.$images)
  .map((imgData) => featureExtractor.process(imgData))
  .awaitPromises();

const $predictions = $features
  .map((features) => classifier.predict(features))
  .awaitPromises();

const predictionViz = confidencePlot($predictions);
```

(a) Initial workflow: predictions are only computed when the user requires them.

```
const $features = input.$images
  .map((imgData) => featureExtractor.process(imgData))
  .awaitPromises();

const $trainingSuccess = classifier.$training
  .filter((x) => x.status === 'success');

const $predictions = $features
  .merge($trainingSuccess.sample($features))
  .map((features) => classifier.predict(features))
  .awaitPromises();

const predictionViz = confidencePlot($predictions);

$predictions.subscribe(({ label }) => {
  classLabel.$text.set(label);
});
```

(b) Improved workflow: predictions are computed at every stroke and when the model is updated.

Listing 2. Programming pipelines with reactive streams. In this example, a stream of predictions is built by passing input images to a feature extractor and a classifier.

6.2 Discussion

This scenario is largely inspired from the authors' own research experience [56]. We developed a similar application for a recent research study and initiated a collaboration for an exhibition dedicated to the general public. Here, we discuss Marcelle's potential to support HCI research.

6.2.1 Rapid iterative prototyping for IML experimental studies. Marcelle is particularly well suited to rapidly iterate on a design. While conducting this research project, we designed two versions of the app in less than a month of interval. The first version was dedicated to a workshop held on the live-stream platform Twitch, and the second version to individual think-aloud sessions with participants. The first prototype was made to be used by many in a non-controlled experiment. The second prototype, however, was designed for a more controlled study. We included the usual elements of experimental design: instructions, a consent form or questionnaires. We also included a task where participants had to sort between images that have been well recognised or confused by the classifier. Dashboard pages were used to structure the different steps of a study. Instructions, forms and questionnaires were integrated using simple text components linking to external online forms. However, it would have been possible to create custom components for questionnaires with answers recorded in a data store.

In Marcelle, the combination of streams and data stores enables flexible logging mechanisms of participants' actions and application states. Any stream can be observed and recorded to a data store. For example, in our study we created additional datasets for recording every sketch drawn by participants, after every stroke drawn. We also logged, for each recorded drawing, the model prediction, and each action (for instance, adding a drawing to the dataset and training the model) with a time-stamp.

6.2.2 Share transparent and reproducible research. Marcelle applications run on the web and are easily shareable. Although online research studies are common, sharing ML models that can be trained by end-users in the browser remains generally challenging. This is particularly important at the time of writing, since the possibilities of conducting in-person studies are constrained by sanitary restrictions due to the pandemic crisis. With a help of a video-mediated communication tool, we conducted both a workshop and individual think-aloud study remotely during the first months of the pandemic [56]. Secondly, shareability makes the research process more transparent and reproducible: one can interact with the system used in an article, and another researcher could reuse and build upon the IML pipeline to conduct new research studies.

7 CASE STUDY 2

This section describes how Marcelle can be used to implement Scenario 2, introduced in Section 2, and discusses the benefits of the architecture for supporting collaboration between ML practitioners and domain experts. A running demo of the case study is available online[6].

7.1 Implementating the Scenario

7.1.1 Integrating Marcelle in a Python Workflow. Louise is a ML engineer working with Keras, a deep learning library in Python. From the `marcelle` python library, she imports the `KerasCallback` class. She creates the callback and passes it to the `fit` method of her custom Keras model. Listing 3 depicts an example of use of the callback in a Python script. Louise specifies where to log the data by indicating the URL of a Marcelle backend, she also specifies the model formats and some training parameters. This data is progressively logged to the Marcelle remote data store during training.

7.1.2 Creating the ML Expert's Dashboard. After bootstrapping an application using Marcelle's CLI tool, Louise instantiates a few components in the main script. She uses a component called `trainingHistory` to access the log data of her training experiments in Python. She adds a `select` widget for select specific runs and a `trainingPlot` to

[6] https://uist2021demos.marcelle.dev

```
model.fit(
    ...
    callbacks=[marcelle.KerasCallback(
        model_checkpoint_freq=1,
        disk_save_format="h5",
        remote_save_format="tfjs",
        run_params=self.params,
    )],
)
```

Listing 3. Communicating with Marcelle data stores from Python is possible using Tensorflow or Keras. For Keras, Marcelle provides a simple callback to log values and record model checkpoints along the training.

display loss and accuracy curves. To further inspect how her model behaves with specific instances, she creates a simple prediction pipeline combining a `tfjsModel` model to load a trained Keras model into the app, and a `imageUpload` component to upload images. Model checkpoints are loaded from the remote data store in the Tensorflow.js format, and inference runs in her web browser. Finally, by abstracting her prediction pipeline into a function, she sets up the side-by-side comparison of two model checkpoints. The resulting interface is shown in Figure 6 (left and middle panels).

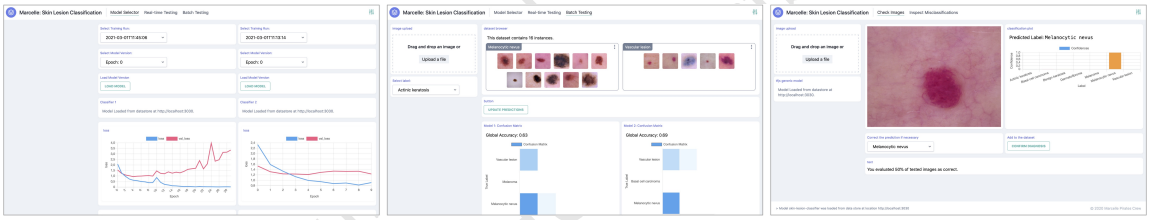

Fig. 6. Dashboards for case study 2. The left and middle dashboards are designed for the ML expert, enabling her to interactively compare various training experiments. The dashboard on the right is designed for the clinician to run predictions and correct misclassifications on his own images.

7.1.3 Creating the Clinician's Dashboard. As Louise's recent results are promising, she reaches out to Michel to gather his feedback on the diagnosis performance. She copies her script to generate a simpler dashboard: she removes components related to monitoring the training and complements the prediction pipeline to gather feedback from Michel. She adds a new select menu and a button allowing the clinician to correct incorrect predictions, as shown in Figure 6 (right panel). She creates two datasets to separately record instances that are correctly or incorrectly classified. That way, Michel can analyze what images are misclassified. Finally, she uses a data store to synchronize the model used in both dashboards: from her developer dashboard, she can select a particular model version, that is then used by default in Michel's interface.

7.2 Discussion

This scenario is inspired by recent publications emphasizing the potential of human-computer collaboration for medical diagnosis, for instance for skin cancer recognition [63]. While our implementation is a proof of concept, we discuss how Marcelle can support the work of ML experts and facilitate their collaboration with end users, in particular domain experts.

7.2.1 Interacting with Large Models. IML has historically been proposed as a way to involve the user in the model's training, and has therefore considered rather small datasets and shallow models. Interactively training deep models, on the contrary, is not realistic in many cases, and boils down to fine-tuning as in transfer learning. However, it is possible to directly interact with their predictions, which can help developers assess the behavior of models.

Marcelle is well integrated with Tensorflow.js, making it possible to load models trained with Tensorflow, a widely used ML framework, in order to perform inference in the browser. This enables both model developers and domain experts to interactively test how the model behaves, in particular with instances representing edge cases. Marcelle can easily incorporate new visualizations, and could support end-user interpretation of ML models using methods from the explainable ML literature [55, 65].

7.2.2 Strengthening Collaboration to Build More Reliable Models. In this medical scenario, Marcelle presents advantages for facilitating the communication between a model developer and domain experts. First, Marcelle supports an incremental approach where more elaborate visualization modules can be progressively integrated to match the user's needs. Second, data stores provide mechanisms to communicate data in both directions. For instance, the ML engineer can seamlessly update the model and synchronize it with the clinician's interface. In response, the clinician's interaction can also record data and annotations to serve as feedback to the model developer, effectively strengthening their collaboration.

This collaborative authoring of ML models can have an important impact on ML transparency. There are many documented instances where intrinsically biased ML models have been harmful to individual and communities [52]. Because these critical failures are not systematically handled *by design* [50], it is essential to empower people with diverse areas of expertise to effectively understand, assess and act upon machine learning systems in various ways in order to foster accountability and fairness.

8 GENERAL DISCUSSION

We described Marcelle at both a conceptual and an implementation levels, and we showed how it can be used in two realistic use cases. In this section we discuss the practical use of Marcelle from personal experience, its limitations and the planned development roadmap.

8.1 Working with Marcelle

Working with Marcelle is straightforward for developers familiar with modern JavaScript and web development. From our experience using Marcelle, this was particularly handy to develop diverse activities using the toolkit and to easily deploy the outcomes online. Here we provide examples stemming from the use of Marcelle in a pedagogical context and workshop activities.

As part of Master's degree program, we developed a series of examples to introduce students to machine learning concepts through learning-by-doing. Using a standalone software could have brought accessibility issues (licence,

compatibility with OS). Installing the toolkit boils down to the installation of *Node.js*, available across OS. However, for accessibility purposes, we also deployed every example on the *Glitch* platform so that every student could access the example, *remix* the code and edit online[7]. We found this process particularly efficient from a pedagogical perspective, for us and for the students. We believe that Marcelle can contribute to the recent pedagogical endeavor on ML education in providing educational resources to teach the general public about machine learning [45, 59, 66].

In parallel to the use of Marcelle for pedagogy, we also ran informal workshops with HCI researchers interested in using ML, and with ML researchers. During these workshops, we designed more advanced activities such as building custom components to illustrate the customization potential of the toolkit. Marcelle's architecture was designed with loosely coupled components to facilitate extension. Using the CLI, it is possible to generate component templates inside existing projects. Once mature, components or component collections can be published using the NPM registry, enabling other developers to integrate them in their applications. As an example, we implemented a custom component for point-based visualization using U-MAP [48]. The code for the component is only 100 lines long and it is quick to implement thanks to seamless integration of third-party libraries[8]. Screenshots of applications created with Marcelle are available in Figure 7.

Workshop attendees were generally highly positive. For instance, a ML researcher working on image denoising expressed his need for interactive systems allowing him to explore how various models react to input noise and found in Marcelle an attractive solution. Another researcher related a current bottleneck in her collaboration with a radiologist that can be addressed with Marcelle: *"the generation of image segmentation maps has to be done via a python script, which is not "user friendly" at all for radiologists, and even for me as an AI researcher, this is a bit tedious as I have to specify the input/output path, launch the script, then open the outputs in a software or in a Jupyter notebook. It is a real bottleneck for results analysis as this implies I have to generate the predictions on my own, then send to the radiologist the predictions, and this is a very inefficient workflow."*

During these workshops, we also asked what would be the aspects that would limit the use of Marcelle in their work. To this question, some mentioned that Marcelle may not be suited to develop advanced tools dedicated to a specific task. One attendee took the example of image segmentation in 3D. She mentioned that having the possibility of rotating the volume in 3D, and having the individual input slices with the overlaid volume at the side could be hard to get, or a lot of implementation work. Another potential breakdown reported by attendees is the learning curve in handling JavaScript, which could spill over into their main activity as ML researchers.

8.2 Limitations and Future Work

8.2.1 Interoperability. Interoperability is key to foster Marcelle's usability. Current interoperability with common ML frameworks suffers several limitations. First, while Tensorflow.js provides good compatibility with models trained using Tensorflow's Python implementation, inference over models created with other frameworks such as Scikit-learn or Pytorch cannot be executed in web browsers. Marcelle currently integrates ONNX.js [49], a JavaScript library for running Open Neural Network Exchange (ONNX) models in web browsers, which can only be used for inference. However, at the time of this writing, ONNX.js runtime only supports a limited set of operators used in deep learning models. Based on preliminary prototypes, we plan to develop Python libraries enabling real-time communication over WebSocket with inference pipelines running server-side.

[7] Some examples are available at at: https://glitch.com/@marcelle.crew/marcelle-examples
[8] A demo of the UMAP component is available online: https://demos.marcelle.dev/umap/

Second, Marcelle provides functions to log data from a Python script during training, however we do not yet support consistent synchronization of data and models between Marcelle and Python. In particular, we plan to further integrate data stores within Python frameworks, by providing appropriate Python interface to use data store objects within Python scripts. As Marcelle is currently compatible with Tensorflow.js, an option would be to provide custom Tensorflow datasets.

Finally, we plan to provide better integration for Marcelle in notebook environments such as Jupyter by embedding Marcelle's components into notebook cells. Embedding Marcelle into such widely used programming tool could potentially increase adoption, further facilitate collaboration and diversify workflows with ML pipelines.

8.2.2 Data Types and Tasks. Marcelle currently provides good support for image processing tasks, in particular classification and segmentation. While we designed its architecture to handle arbitrary data types – including images, audio, text, time series and structured data, – developing components covering such a large spectrum is time-consuming for a small team of researchers. We plan to incrementally integrate support for other data types and tasks. Our choice of a minimalist interface for components is advantageous for integrating new data formats, input components, models and visualizations, as it does not require changes at the architecture level. However, this might lead to a large number of specialized components rather than a more limited set of components supporting polymorphism. Focusing on polymorphic components – for instance a generic prediction visualization that would provide appropriate feedback for classification, segmentation, or object detection results, – would facilitate the use of the toolkit by novices, but would hinder customization and appropriation.

9 CONCLUSION

We have presented Marcelle, a programming toolkit implementing our vision for composable interactions with machine learning. Through two case studies, we illustrated how Marcelle's component-based architecture and high-level API enable the efficient development of interfaces dedicated to various users: machine learning experts, designers, domain experts or the general public. Because components embed both processing and interaction, composing workflows boils down to (1) creating reactive pipelines linking user actions to machine learning tasks and (2) assembling components into user interfaces using flexible layouts. From our own experience, this approach makes it possible to quickly iterate over designs, and to reuse and share particular interaction techniques.

Finally, from a theoretical point of view, our conception of interactive machine learning components share similarities with the notion of information substrates [22]. Substrates are computational medium that hold information, computation and interaction. They can be composed, extended or manipulated, and they transcend application boundaries. Our architecture provides a starting point for formalizing interactive machine learning substrates that would be shared and repurposed across users and applications. We see in this perspective a promising way to conceptualise our interactions with machine learning as design material, and to induce fundamental questions for the field of Human-Computer Interaction.

ACKNOWLEDGEMENTS

This research was supported by the ELEMENT project (ANR-18-CE33-0002) from the French National Research Agency. We want to acknowledge and thank everyone who was involved in each stage of the research, in particular the anonymous reviewers, students of the Interactive Machine Learning course in Université Paris-Saclay and workshop

participants. We especially want to express our sincere gratitude to Frédéric Bevilacqua, Alessandro Delmonte, Clara Rigaud, Yvonne Jansen, Antoine Loriette, and Benjamin Matuszewski.

REFERENCES

[1] [n.d.]. Feathers | A framework for real-time applications and REST APIs. https://feathersjs.com/. (Accessed on 04/06/2021).

[2] [n.d.]. GitHub - fossasia/visdom: A flexible tool for creating, organizing, and sharing visualizations of live, rich data. Supports Torch and Numpy. https://github.com/fossasia/visdom. (Accessed on 04/06/2021).

[3] [n.d.]. Keras: The Python Deep Learning API. https://keras.io/. (Accessed on 07/27/2021).

[4] [n.d.]. Lobe | Machine Learning Made Easy. https://lobe.ai/. (Accessed on 04/06/2021).

[5] [n.d.]. ml5js: Friendly Machine Learning For The Web. https://ml5js.org/. (Accessed on 04/06/2021).

[6] [n.d.]. ml.js: Machine learning tools in JavaScript. https://github.com/mljs/ml. (Accessed on 04/06/2021).

[7] [n.d.]. Most.js: Monadic Event Stream. https://github.com/mostjs/core/. (Accessed on 04/07/2021).

[8] [n.d.]. Neptune.ai | Metadata Store for MLOps. https://neptune.ai/. (Accessed on 04/06/2021).

[9] [n.d.]. Runway | Make the Impossible. https://runwayml.com/. (Accessed on 04/06/2021).

[10] [n.d.]. Svelte: Cybernetically enhanced web apps. https://svelte.dev/. (Accessed on 04/07/2021).

[11] [n.d.]. VisualDL: Deep Learning Visualization Toolkit. https://github.com/PaddlePaddle/VisualDL. (Accessed on 04/06/2021).

[12] [n.d.]. Vite: Next Generation Frontend Tooling. https://vitejs.dev/. (Accessed on 04/06/2021).

[13] Cycling '74. [n.d.]. Max is software for experimentation and invention. https://cycling74.com/. (Accessed on 04/06/2021).

[14] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 265–283. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[15] Allegro AI. [n.d.]. ClearML | MLOps for Data Science Teams. https://clear.ml/. (Accessed on 04/06/2021).

[16] Saleema Amershi. 2011. Designing for Effective End-User Interaction with Machine Learning. In *Proceedings of the 24th Annual ACM Symposium Adjunct on User Interface Software and Technology* (Santa Barbara, California, USA) *(UIST '11 Adjunct)*. Association for Computing Machinery, New York, NY, USA, 47–50. https://doi.org/10.1145/2046396.2046416

[17] Saleema Amershi, Maya Cakmak, W. Bradley Knox, and Todd Kulesza. 2014. Power to the people: The role of humans in interactive machine learning. *AI Magazine* 35, 4 (12 2014), 105–120. https://doi.org/10.1609/aimag.v35i4.2513

[18] Saleema Amershi, Max Chickering, Steven M. Drucker, Bongshin Lee, Patrice Simard, and Jina Suh. 2015. ModelTracker: Redesigning Performance Analysis Tools for Machine Learning. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 337–346. https://doi.org/10.1145/2702123.2702509

[19] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N. Bennett, Kori Inkpen, Jaime Teevan, Ruth Kikin-Gil, and Eric Horvitz. 2019. Guidelines for Human-AI Interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland UK) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300233

[20] Geering Up at UBC. 2019. Bringing AI into the Classroom. https://www.actua.ca/en/bringing-ai-into-the-classroom. [Online; accessed 16-Sep-2020].

[21] Engineer Bainomugisha, Andoni Lombide Carreton, Tom van Cutsem, Stijn Mostinckx, and Wolfgang de Meuter. 2013. A Survey on Reactive Programming. *Comput. Surveys* 45, 4, Article 52 (Aug. 2013), 34 pages. https://doi.org/10.1145/2501654.2501666

[22] Michel Beaudouin-Lafon. 2017. Towards Unified Principles of Interaction. In *Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter* (Cagliari, Italy) *(CHItaly '17)*. Association for Computing Machinery, New York, NY, USA, Article 1, 2 pages. https://doi.org/10.1145/3125571.3125602

[23] Stuart Berg, Dominik Kutra, Thorben Kroeger, Christoph N. Straehle, Bernhard X. Kausler, Carsten Haubold, Martin Schiegg, Janez Ales, Thorsten Beier, Markus Rudy, Kemal Eren, Jaime I. Cervantes, Buote Xu, Fynn Beuttenmueller, Adrian Wolny, Chong Zhang, Ullrich Koethe, Fred A. Hamprecht, and Anna Kreshuk. 2019. Ilastik: Interactive Machine Learning for (Bio)Image Analysis. *Nature Methods* 16, 12 (Dec. 2019), 1226–1232. https://doi.org/10.1038/s41592-019-0582-9

[24] Francisco Bernardo, Michael Zbyszyński, Mick Grierson, and Rebecca Fiebrink. 2020. Designing and Evaluating the Usability of a Machine Learning API for Rapid Prototyping Music Technology. *Frontiers in Artificial Intelligence* 3 (2020), 13. https://doi.org/10.3389/frai.2020.00013

[25] Jamie Bullock and Ali Momeni. 2015. Ml.Lib: Robust, Cross-Platform, Open-Source Machine Learning for Max and Pure Data. In *Proceedings of the International Conference on New Interfaces for Musical Expression* (Baton Rouge, Louisiana, USA) *(NIME 2015)*. The School of Music and the Center for Computation and Technology (CCT), Louisiana State University, Baton Rouge, Louisiana, USA, 265–270.

[26] Baptiste Caramiaux, Jules Françoise, Norbert Schnell, and Frédéric Bevilacqua. 2014. Mapping through listening. *Computer Music Journal* 38, 3 (2014), 34–48.

[27] Michelle Carney, Barron Webster, Irene Alvarado, Kyle Phillips, Noura Howell, Jordan Griffith, Jonas Jongejan, Amit Pitaru, and Alexander Chen. 2020. Teachable Machine: Approachable Web-Based Tool for Exploring Machine Learning Classification. In *Extended Abstracts of the 2020 CHI*

Conference on Human Factors in Computing Systems (Honolulu, HI, USA) *(CHI EA '20)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3334480.3382839

[28] Eric Corbett, Nathaniel Saul, and Meg Pirrung. [n.d.]. *Interactive Machine Learning Heuristics*. Technical Report. https://learningfromusersworkshop.github.io/papers/IMLH.pdf

[29] Janez Demšar, Tomaž Curk, Aleš Erjavec, Črt Gorup, Tomaž Hočevar, Mitar Milutinovič, Martin Možina, Matija Polajnar, Marko Toplak, Anže Starič, Miha Štajdohar, Lan Umek, Lan Žagar, Jure Žbontar, Marinka Žitnik, and Blaž Zupan. 2013. Orange: Data Mining Toolbox in Python. *Journal of Machine Learning Research* 14, 35 (2013), 2349–2353. http://jmlr.org/papers/v14/demsar13a.html

[30] Carlos Gonzalez Diaz, Phoenix Perry, and Rebecca Fiebrink. 2019. Interactive Machine Learning for More Expressive Game Interactions. In *2019 IEEE Conference on Games (CoG)*. 1–2. https://doi.org/10.1109/CIG.2019.8848007

[31] Graham Dove, Kim Halskov, Jodi Forlizzi, and John Zimmerman. 2017. *UX Design Innovation: Challenges for Working with Machine Learning as a Design Material*. Association for Computing Machinery, New York, NY, USA, 278–288. https://doi.org/10.1145/3025453.3025739

[32] John J. Dudley and Per Ola Kristensson. 2018. A Review of User Interface Design for Interactive Machine Learning. *ACM Transactions on Interactive Intelligent Systems* 8, 2, Article 8 (June 2018), 37 pages. https://doi.org/10.1145/3185517

[33] Jerry Alan Fails and Dan R. Olsen. 2003. Interactive Machine Learning. In *Proceedings of the 8th International Conference on Intelligent User Interfaces* (Miami, Florida, USA) *(IUI '03)*. Association for Computing Machinery, New York, NY, USA, 39–45. https://doi.org/10.1145/604045.604056

[34] Rebecca Fiebrink and Baptiste Caramiaux. 2018. The Machine Learning Algorithm as Creative Musical Tool. In *The Oxford Handbook of Algorithmic Music*, Roger T. Dean and Alex McLean (Eds.). Oxford University Press, 181–208. https://research.gold.ac.uk/id/eprint/23154/

[35] Rebecca Fiebrink, Perry R. Cook, and Dan Trueman. 2011. Human Model Evaluation in Interactive Supervised Learning. In *Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems - CHI '11*. ACM Press, Vancouver, BC, Canada, 147. https://doi.org/10.1145/1978942.1978965

[36] James Fogarty, Desney Tan, Ashish Kapoor, and Simon Winder. 2008. CueFlik: Interactive Concept Learning in Image Search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Florence, Italy) *(CHI '08)*. Association for Computing Machinery, New York, NY, USA, 29–38. https://doi.org/10.1145/1357054.1357061

[37] Jules Françoise and Frédéric Bevilacqua. 2018. Motion-Sound Mapping through Interaction: An Approach to User-Centered Design of Auditory Feedback Using Machine Learning. *ACM Transactions on Interactive Intelligent Systems* 8, 2, Article 16 (June 2018), 30 pages. https://doi.org/10.1145/3211826

[38] Jules Françoise, Norbert Schnell, Riccardo Borghesi, and Frédéric Bevilacqua. 2014. Probabilistic Models for Designing Motion and Sound Relationships. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. Goldsmiths, University of London, London, United Kingdom, 287–292. https://doi.org/10.5281/zenodo.1178764

[39] Marco Gillies. 2019. Understanding the Role of Interactive Machine Learning in Movement Interaction Design. *ACM Transactions on Computer-Human Interaction* 26, 1, Article 5 (Feb. 2019), 34 pages. https://doi.org/10.1145/3287307

[40] Marco Gillies, Rebecca Fiebrink, Atau Tanaka, Jérémie Garcia, Frédéric Bevilacqua, Alexis Heloir, Fabrizio Nunnari, Wendy Mackay, Saleema Amershi, Bongshin Lee, Nicolas d'Alessandro, Joëlle Tilmanne, Todd Kulesza, and Baptiste Caramiaux. 2016. Human-Centred Machine Learning. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI EA '16)*. Association for Computing Machinery, New York, NY, USA, 3558–3565. https://doi.org/10.1145/2851581.2856492

[41] Markus Hofmann and Ralf Klinkenberg. 2013. *RapidMiner: Data Mining Use Cases and Business Analytics Applications*. Chapman & Hall/CRC.

[42] Fred Hohman, Kanit Wongsuphasawat, Mary Beth Kery, and Kayur Patel. 2020. Understanding and Visualizing Data Iteration in Machine Learning. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376177

[43] Andreas Holzinger. 2016. Interactive machine learning for health informatics: when do we need the human-in-the-loop? *Brain Informatics* 3, 2 (6 2016), 119–131. https://doi.org/10.1007/s40708-016-0042-6

[44] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861

[45] M. Kahng, N. Thorat, D. H. Chau, F. B. Viégas, and M. Wattenberg. 2019. GAN Lab: Understanding Complex Deep Generative Models using Interactive Visual Experimentation. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2019), 310–320. https://doi.org/10.1109/TVCG.2018.2864500

[46] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and Jupyter development team. 2016. Jupyter Notebooks ? a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, Fernando Loizides and Birgit Scmidt (Eds.). IOS Press, 87–90. https://eprints.soton.ac.uk/403913/

[47] Todd Kulesza, Margaret Burnett, Weng-Keen Wong, and Simone Stumpf. 2015. Principles of Explanatory Debugging to Personalize Interactive Machine Learning. In *Proceedings of the 20th International Conference on Intelligent User Interfaces* (Atlanta, Georgia, USA) *(IUI '15)*. Association for Computing Machinery, New York, NY, USA, 126–137. https://doi.org/10.1145/2678025.2701399

[48] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. 2018. UMAP: Uniform Manifold Approximation and Projection. *Journal of Open Source Software* 3, 29 (2018), 861. https://doi.org/10.21105/joss.00861

[49] Microsoft. [n.d.]. ONNX.js: run ONNX models using JavaScript. https://github.com/Microsoft/onnxjs/. (Accessed on 04/06/2021).

[50] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model Cards for Model Reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency* (Atlanta, GA, USA) *(FAT* '19)*. Association for Computing Machinery, New York, NY, USA, 220–229. https://doi.org/10.1145/3287560.3287596

[51] Giselle Nodalo, Jose Ma. Santiago, Jolene Valenzuela, and Jordan Aiko Deja. 2019. On Building Design Guidelines for An Interactive Machine Learning Sandbox Application. In *Proceedings of the 5th International ACM In-Cooperation HCI and UX Conference* (Jakarta, Surabaya, Bali, Indonesia) *(CHIuXiD'19)*. Association for Computing Machinery, New York, NY, USA, 70–77. https://doi.org/10.1145/3328243.3328253

[52] Cathy O'Neil. 2016. *Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy*. Crown Publishing Group, USA.

[53] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

[54] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[55] Wojciech Samek, Gregoire Montavon, Andrea Vedaldi, Lars Kai Hansen, and Klaus-Robert Muller. 2019. *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning* (1st ed.). Springer Publishing Company, Incorporated.

[56] Téo Sanchez, Baptiste Caramiaux, Jules Françoise, Frédéric Bevilacqua, and Wendy E. Mackay. 2021. How Do People Train a Machine? Strategies and (Mis)Understandings. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW1, Article 162 (April 2021), 26 pages. https://doi.org/10.1145/3449236

[57] Hugo Scurto, Bavo Van Kerrebroeck, Baptiste Caramiaux, and Frédéric Bevilacqua. 2021. Designing Deep Reinforcement Learning for Human Parameter Exploration. *ACM Transactions on Computer-Human Interaction* 28, 1, Article 1 (Jan. 2021), 35 pages. https://doi.org/10.1145/3414472

[58] Shital Shah, Roland Fernandez, and Steven Drucker. 2019. A System for Real-Time Interactive Analysis of Deep Learning Training. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Valencia, Spain) *(EICS '19)*. Association for Computing Machinery, New York, NY, USA, Article 16, 6 pages. https://doi.org/10.1145/3319499.3328231

[59] Daniel Smilkov, Shan Carter, D Sculley, Fernanda B Viégas, and Martin Wattenberg. 2016. Direct-Manipulation Visualization of Deep Networks. In *KDD 2016 Workshop on Interactive Data Exploration and Analytics (IDEA)*. San Francisco, California, USA. arXiv:1708.03788

[60] Daniel Smilkov, Nikhil Thorat, Yannick Assogba, Ann Yuan, Nick Kreeger, Ping Yu, Kangyi Zhang, Shanqing Cai, Eric Nielsen, David Soergel, Stan Bileschi, Michael Terry, Charles Nicholson, Sandeep N. Gupta, Sarah Sirajuddin, D. Sculley, Rajat Monga, Greg Corrado, Fernanda B. Viegas, and Martin Wattenberg. 2019. TensorFlow.js: Machine Learning for the Web and Beyond. (2019). https://arxiv.org/abs/1901.05350

[61] Thilo Spinner, Udo Schlegel, Hanna Schäfer, and Mennatallah El-Assady. 2020. explAIner: A Visual Analytics Framework for Interactive and Explainable Machine Learning. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2020), 1064–1074. https://doi.org/10.1109/TVCG.2019.2934629

[62] Justin Talbot, Bongshin Lee, Ashish Kapoor, and Desney S. Tan. 2009. EnsembleMatrix: Interactive Visualization to Support Machine Learning with Multiple Classifiers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Boston, MA, USA) *(CHI '09)*. Association for Computing Machinery, New York, NY, USA, 1283–1292. https://doi.org/10.1145/1518701.1518895

[63] Philipp Tschandl, Christoph Rinner, Zoe Apalla, Giuseppe Argenziano, Noel Codella, Allan Halpern, Monika Janda, Aimilios Lallas, Caterina Longo, Josep Malvehy, John Paoli, Susana Puig, Cliff Rosendahl, H. Peter Soyer, Iris Zalaudek, and Harald Kittler. 2020. Human–Computer Collaboration for Skin Cancer Recognition. *Nature Medicine* 26, 8 (Aug. 2020), 1229–1234. https://doi.org/10.1038/s41591-020-0942-0

[64] Emily Wall, Soroush Ghorashi, and Gonzalo Ramos. 2019. Using Expert Patterns in Assisted Interactive Machine Learning: A Study in Machine Teaching. In *Human-Computer Interaction – INTERACT 2019*, David Lamas, Fernando Loizides, Lennart Nacke, Helen Petrie, Marco Winckler, and Panayiotis Zaphiris (Eds.). Springer International Publishing, Cham, 578–599.

[65] Danding Wang, Qian Yang, Ashraf Abdul, and Brian Y. Lim. 2019. Designing Theory-Driven User-Centric Explainable AI. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3290605.3300831

[66] Z. J. Wang, R. Turko, O. Shaikh, H. Park, N. Das, F. Hohman, M. Kahng, and D. H. Polo Chau. 2021. CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2021), 1396–1406. https://doi.org/10.1109/TVCG.2020.3030418

[67] Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mané, Doug Fritz, Dilip Krishnan, Fernanda B. Viégas, and Martin Wattenberg. 2018. Visualizing Dataflow Graphs of Deep Learning Models in TensorFlow. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (2018), 1–12. https://doi.org/10.1109/TVCG.2017.2744878

[68] Qian Yang, Aaron Steinfeld, Carolyn Rosé, and John Zimmerman. 2020. Re-Examining Whether, Why, and How Human-AI Interaction Is Uniquely Difficult to Design. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376301

[69] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How Transferable Are Features in Deep Neural Networks?. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2* (Montreal, Canada) *(NIPS'14)*. MIT Press, Cambridge, MA, USA, 3320–3328.

[70] Jun Yuan, Changjian Chen, Weikai Yang, Mengchen Liu, Jiazhi Xia, and Shixia Liu. 2021. A Survey of Visual Analytics Techniques for Machine Learning. *Computational Visual Media* 7, 1 (March 2021), 3–36. https://doi.org/10.1007/s41095-020-0191-7

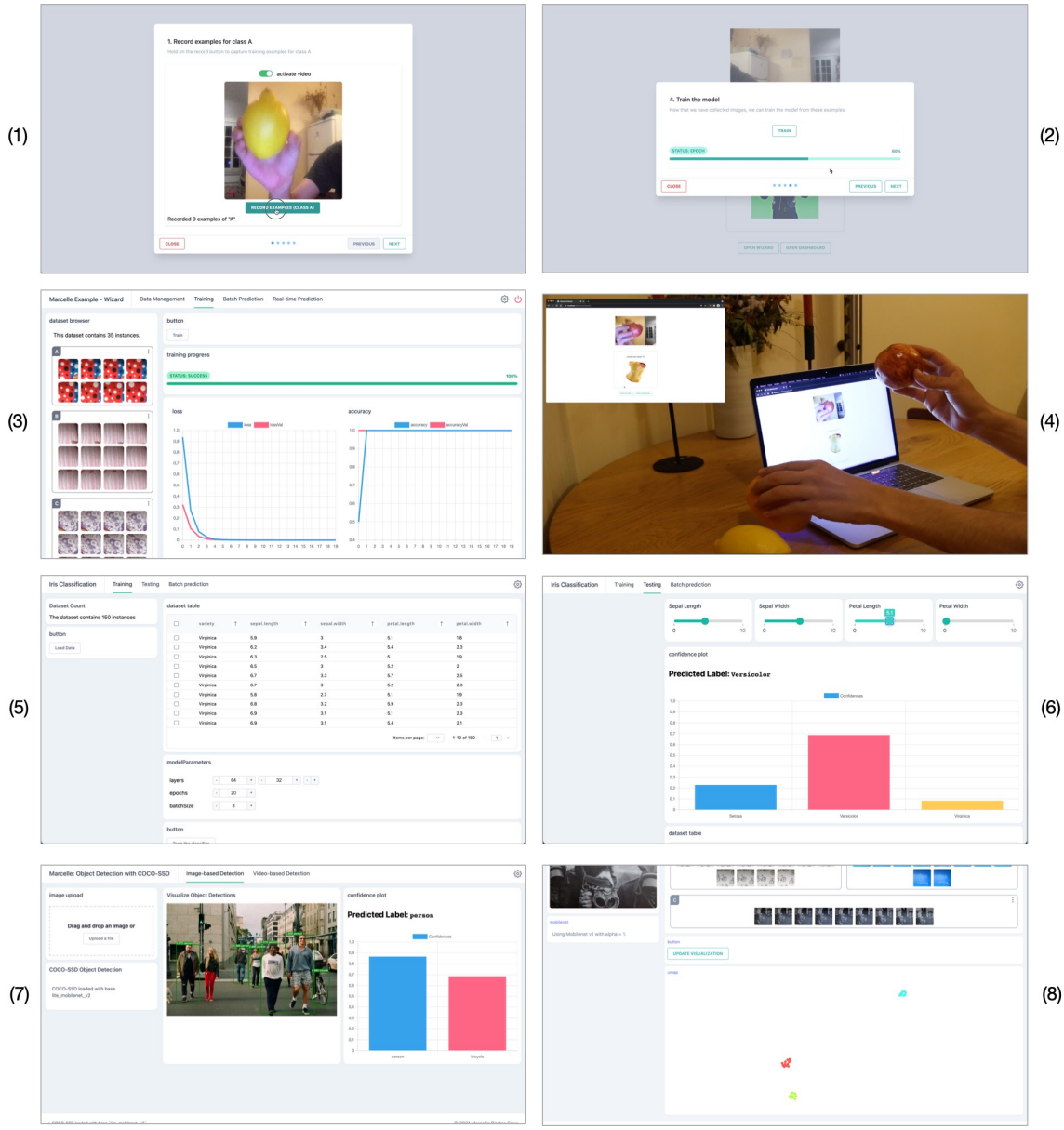

Fig. 7. Screenshots of existing Marcelle applications and examples. (1-2) Wizard pages dedicated to capturing images from a webcam and training an image classifier. (3) Dashboard view of the image classifier training, including a visualization of the dataset and of training logs. (4) Picture of a user interacting with the final application, a web page integrating the classifier to run real-time predictions and play music loops accordingly. (5-6) Dashboard dedicated to training and interactively testing a flower classifier from the IRIS dataset. (7) Application running object detection from uploaded images using a pre-trained model. (8) Custom component implementing U-MAP visualization of an image dataset.