2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC) | 978-1-7281-7628-4/20/$31.00 ©2020 IEEE | DOI: 10.1109/SYNASC51798.2020.00056

DS Lab Notebook: A new tool for data science applications

1st Alexandru Ionascu
Department of Computer Science
West University of Timisoara
Timisoara, Romania
alexandru.ionascu96@e-uvt.ro

2nd Sebastian-Aurelian Stefaniga
Department of Computer Science
West University of Timisoara
Timisoara, Romania
sebastian.stefaniga@e-uvt.ro

***Abstract*—The main focus of the technical application of this research relies on a web and mobile-based solution identified as Data Science Notebook named as DS Lab Notebook. Specifically, the main focus will be on tackling already present challenges in data science education and a solution presented around DS Lab, an interactive computational notebook. The core ideas are represented by the concept of extending the traditional computing notebooks, especially from the Jupyter family, with live visualizations, debugging, widgets, and interactivity during the educational process across all the major platforms: web, Android, and IOS. The features are outlined in several use cases that can be useful in the data-science teaching process, with a primary focus in matrix manipulations, scatter plots, and image filters. Python 3 was used as the main programming runtime and the back-end for providing access to variable values and type information is being described in the form of a runtime-independent pipeline relying on code parsing and injection.**

***Index Terms*—Data Science, Jupyter, Real-Time, Interactive systems and tools, Human computer interaction, Development frameworks and environments**

I. Introduction

The research subject was chosen for its complexity and potential of originality in the field of interactive computational notebooks. There are already plenty of solutions as public web platforms regarding the development of analytical or implicitly algorithmic thinking which we perceive them expressly well executed. However, in the subfield of data science education, we believe there is still a wide space to explore since there are few data science educational platforms, hardly differentiated, which are holding an impressive monopoly. It normally relies on a wide variety of scientific methods and algorithms.

In this research work, most of the mentioned use cases for the proposed DS Lab notebook in the following sections are also applicable to the broader notion of research programming and not only. DS Lab Notebook is designed in this way to tackle the most common and difficult situations that occur during the educational process in this field.

II. State of the Art in Computational Notebooks for Data Science Education

Considering the great success of the family of Jupyter notebooks [12] [3] and its derivatives such as Google Colab [5], Azure ML Notebooks [10], this research work with the implementation of DS Lab Notebook is intended to transfer some of the major benefits of computational notebooks into more specific educational domains, centering on data science.

However, the potential benefits come along with the cost of limiting computing capabilities. Leveraging the offline computational environment and real-time visualizations will inevitably lead to performance limitations that will affect visibly the long-running processes and, specifically, the size of the datasets that the end-users will use. We consider that this tradeoff is feasible and not particularly concerning since, in most educational contexts, we don't intend to process large-volume data.

A. DS Lab architecture

In terms of the architecture of the proposed notebook, the main focus is on the front-end implementation, as well as integrating a Python runtime environment. For front-end, React Native is being used, with support for React Native Web, and Typescript as the programming language. The programming runtime, Skulpt Python is the main choice for DS Lab Notebook. The application is based on React and Typescript, but without using a state container, like Redux. Therefore, the depth of the dependency tree between components is objectively minimized. The logical components consist of classes, interfaces, React Components, and libraries. Their tree-structured dependencies are being described in the figure below.

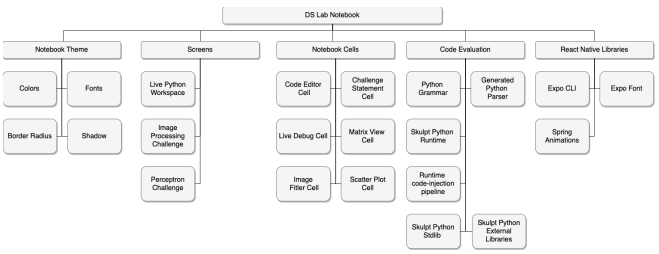

Figure 1: DS Lab Notebook horizontal scaling

In a classical React application, a top-down approach is most commonly used. A relevant example would be the theme-logical component. Traditionally, in a React application, one may use a context and a provider from the Context API. Instead, we avoid state management situations and possible

978-1-7281-7628-4/20/$31.00 ©2020 IEEE
DOI 10.1109/SYNASC51798.2020.00056

overridden values in the components tree. The reason behind this choice is to offer a higher level of transparency, but also simplicity in design. DS Lab Notebook is targeting extensibility, and this decision is essentially a trade-off between the transparency and the number of dependency links.

B. Programming environment

We have used React Native, a framework that promises code sharing, but not necessarily full cross-platform support in all the scenarios. The components behind each React Native implementation behave natively since their rendering is done with the native SDKs. The cross-platform behavior is guaranteed with several limitations. However, the React Native core components are comprehensively enough to build complex and mature cross-platform applications. For DS Lab Notebook, Expo CLI was used, and usage of native modules is being sacrificed for more important objectives such as React Native Web out of the box official support. With the recent inclusion of React Native Web in the official support of Expo CLI, approaching an imminent v1.0 release, now it is possible to use a singular codebase across every targeted platform. DS Lab Notebook is proposing a comparably unexplored architecture for building educational and computational notebooks. For a cross-platform application, a Python environment must be supported on all the platforms: Android, IOS, Web. We note that both Android and IOS can support C/C++ modules natively.

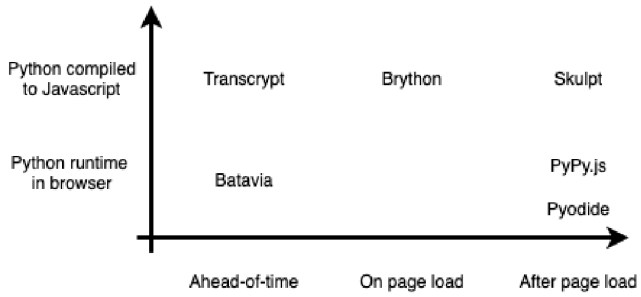

Figure 2: Python environments in the browser [9].

Brython [2] project is in the category of on-page-load compilation. Technically, this approach is neither useful as a runtime, but, it can be eventually extended to support after-page-load compilation. Skulpt, PyPy.js [14], and Mozilla's Pyodide [13] are the only viable solutions for accomplishing this task. The objective is essentially bringing interactive data visualization in the browser, with a full Python implementation, along with the Python scientific stack. This is possible to the CPython project being compiled to WebAssembly. Also, other WebAssembly compilations were possible for NumPy, SciPy, Matplotlib, and Pandas. A Jupyter notebook running Pyodide as an IPython back-end is not only possible but also officially supported.

C. Computational Notebooks Comparison

The popularized notebooks are not exactly suited for educational purposes, but to get an insight into the objectives of the DS Lab computational notebook, we will try to compare it with the already existing ones. DS Lab is planning to approach several subjects from a different angle: UX, visualizations, targeted platforms, and distribution. DS Lab is targeting to become one of the very few computational notebooks compatible with mobile platforms. Distribution is also crucial, as many standalone computational notebooks are requiring a multi-step complex installation, and the remote-execution notebooks are available directly through the web, DS Lab is a a solution designed to tackle both situations. We approach DS Lab as a standalone computational notebook and is both available through the web directly or can be delivered under built-in store applications on mobile. Other computational notebooks that share similar advantages or disadvantages can be added to the list, Nteract Notebook [11], Apache Zeppelin [1], and the Root Data Analysis Framework [16] from CERN. However, Apache Zeppelin and the Root Data Analysis Framework is mostly notable for big-data scenarios. Even though, they present several enhancements that might be useful for any computational-notebook, a fair comparison is adequately described along with two Jupyter-style notebooks: Azure ML Notebooks and Google Colab. A comprehensive description can be found in the table below.

Feature	DS Lab	Azure Notebook	Google Colab
UX	Live-coding, no execution buttons	Nteract-based UX-features	Standard
Visualization	Only custom widgets, extensible	Standard and custom widgets	Standard, non-extensible
Platform	Web / Mobile	Web	Web
Collaboration	1:1 Student and Instructor	Limited	Real-time collaboration
Services	No, but possible	Azure Services	Google Services
Big Data	No	with Azure services	generally, no
Environment	Local	Remote	Remote
Distribution	Web / Mobile	Web	Web
Code Editor	Limited but extensible	Monaco Editor, core of VS Code	Limited, non-extensible

Table I: Notebook Comparison

III. DS LAB TECHNICAL IMPLEMENTATION

The philosophy behind DS Lab is equal access to education around the globe. By applying the inversion principle, we may ask what is the main issue that prevents us from achieving this idea. One example would be from the University of Beijing with the JSCPP interpreter [4]. Even though we mention this example in the context of a course that was mostly being

thought online, internet connection reliability was a considerable issue for many students, hence the solution to build a standalone, local-first C/C++ runtime in the browser. Tablets and smartphones in education have been largely popularized in the recent pandemic response considering the need for remote education. This situation can be speculated to provide equal access to education regardless of the device or platform. Web and mobile are both supported in DS Lab Notebook, and even the technical implementations are delivered under the same codebase, with only minimal platform-specific changes.

The proposed solution DS Lab is centering on the same the problem of coordinating student-teacher experience with the additional key points: offline-first approach, improved mobile experience, live coding and debugging, extensibility, and new visualization widgets.

A. *Offline approach*

Although the distribution channel requires an initial internet connection, most of the available traditional tools require a steady connection. The Python runtime behind DS Lab Notebook implementation is Skulpt [17]. There are a couple of configurations necessary apart from the out of the box Skulpt. Skulpt implementation is relatively limited, but more than 90% of the Python 3+ standard features are covered. The single missing point is the full standard library implementation, which is only marginal problematic since in the educational process authenticity can be compromised in most specific curriculums. From a technical point of view, Skulpt Python supports suspensions, and by that means longer running programs or infinite loops won't impact the entire interface. Data for evaluation purposes is being implemented as Javascript functions accessible through the Python runtime interface, which are essentially parsing stdin. Functions like: nextInt, nextFloat, nextString, nextChar are being added to the list of builtins, and are accessible to be used during code-challenges, as the data can be passed to stdin.

From the preliminary benchmarking presented by Brython documentation and test runner [18], Skulpt is generally slower than the other environments. As a rule of thumb, Skulpt is expected to run 20-30x slower than CPython, and in the worst tested scenarios, 120x slower approximately. Considering these numbers, we can safely say that small-to-medium sized datasets won't necessarily be problematic.

B. *Improved mobile experience*

All the mentioned traditional computational notebooks have close to zero support for mobile phones, and insignificant support for tablets. In this case, our work is proposing a new solution in this regard. Another aspect relies on mobile UX. The notebook-approach for educational challenges is particularly useful for delivering almost identical representations on all devices. Also, some peculiar UX changes are being done apart from traditional notebook solutions. A proper example of a data science training platform would be comparable to figure 3, representing a simple challenge for a perceptron with linearly separable data. During the development iterations of DS Lab notebook, the traditional computational-notebook style design has been preferred. Mobile compatibility can be examined in figure 4.

C. *Extensibility, live coding and live debugging*

Extensibility is an expected feature for the area of computational notebooks. The DS Lab extensibility is composed of two parts: custom components and even custom designed challenges for notebooks. DS Lab Notebook is targeting by design the following categories of extensions: code editor extensions; cell plots such as scatter plots, bar charts, pie charts, tables, etc.; image cells: for visualization, for applying filters; Markdown / Tex / Math.js cells for writing mathematical descriptions; Interactive widgets: visualization cells that support user interaction through additional components (such as buttons, inputs, progress-bars, etc.). Live-coding or live-debugging is not a particularly new concept. However, DS Lab is promoting this concept with the belief that many students are experiencing breakthrough moments while seeing the result of their work in real-time.

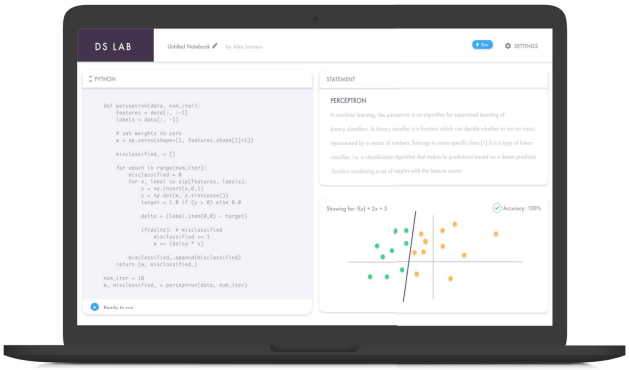

Figure 3: Standard web layout concept for a perceptron challenge, with execution buttons.

There are two relevant examples of this specific area. The first one is an academic example, with Python Tutor [7]. It is probably one of the first live debugging research articles, which later on has been successfully implemented in a large number of top universities, and millions of students have used it by today. The second example is the Playgrounds [19], while performance and integration are superior for Apple's solution, DS Lab is shifting their approach which is focusing on Swift programming language, to Python.

In terms of technical implementation, the primary tool for this task is Jison [8], a Javascript-based implementation of GNU Bison [6]. The input for Jison is a Python 3.4 grammar based on the official Python documentation. The same approach was being used by a discontinued compiler project: PyScript [15]. The whole pipeline for the debugging and live coding back-end can be seen in figure 6. The generated parser can provide an entire tree of code locations, from which we

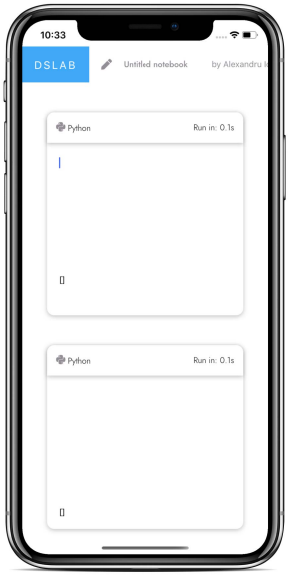

Figure 4: Traditional notebook layout, compatible with mobile portrait mode and no execution buttons, running on IPhone X.

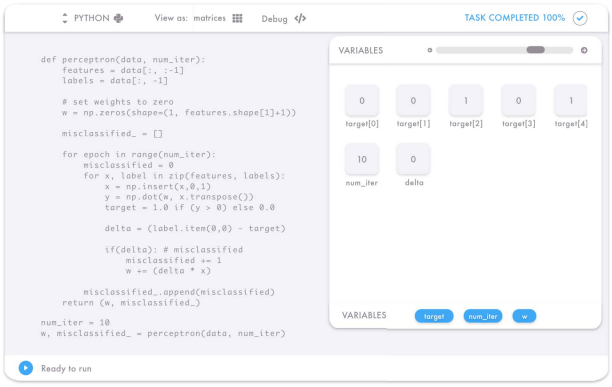

Figure 5: DS Lab Notebook live visualization panel, with time-machine effect slider and variable-scope filtering.

can filter the assignments operators where we can then inject stdout printing and can extract each change in value. To extract all assignments, we can explore the resulted tree from parsing in a depth-first-search algorithm.

Since runtime code injection can introduce security flaws, multiple 56 bit hashes are generated. At one level, for each injected line there must be a unique identifier. At a second level, the injected output must be separated by the user standard output. Another two 56 bit hashes will be randomly generated as separators, to prevent any future security exploits.The outlined idea is that the pipeline can be extended for any programming runtime. This is especially useful for supporting

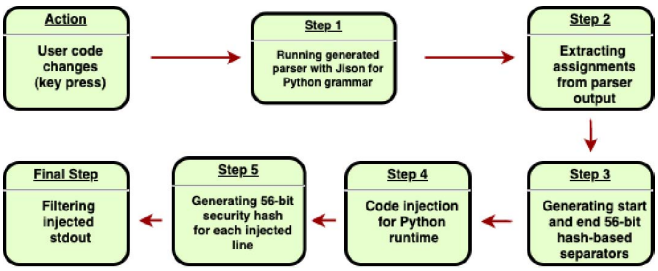

Figure 6: Code injection and execution pipeline

more Python versions, or even new programming languages. Even though the proposed pipeline is limited in certain ways (such as multithreading), there are immediate benefits for using this approach in terms of speed and extensibility.

D. New widgets visualization

As mentioned in the previous parts, the standard Python visualization libraries are limited in Skulpt implementation and especially on the mobile environment. One way to overcome this issue to include Plotly or Mathplotlib is via generic webviews as export of the original plotting result. Another argument for new visualization widgets is to leverage the offline Python environment. As seen in figure 7, one can extract the Python function for an image filter and apply it directly. Even more, if the image filter component is sufficiently small as in the number of pixels, the filter transformation can be applied in real-time, at every key press.

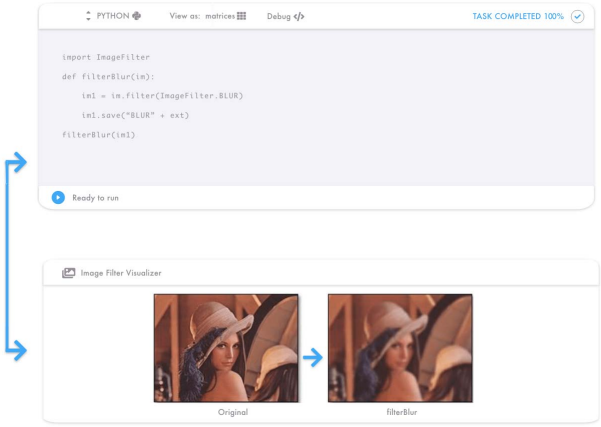

Figure 7: Real-time image filter visualization widget.

A direct consequence of the presented live debugging feature in the context of data science education is matrix-form visualization. With Skulpt-based Python runtime, we can extract type information about all the objects and we can create new particular visualizations that can help in various challenges, whether we talk about an academic process or private workshops. In the data-intense situations, special rendering algorithms and optimizations are required. It is worth mentioning in this case, no such technique is being provided

since we acknowledge the fact this approach is feasible only in a limited data size context.

IV. Conclusions

Data science was the main focus for DS Lab notebook, where data plotting and image processing techniques were exemplified, but the core architecture of the extensible notebook and the presented features may lead as well to other educational or practical use cases as well in different computer science subfields. DS Lab notebook is expected to provide a practical solution for distributing entry-level data science educational notebooks. The concepts presented within the presented notebook can enhance the live or remote educational experience, but also massive online open courses, code training platforms can benefit from them. To conclude, in this research work, several key-points have been promoted in the context of computational notebooks. The several limitations of mobile platforms and offline-environments can lead to unprecedented user experience. Live programming is yet to be further explored in this subfield while the most promoted educational consequence is minimizing the time when the end-user can obtain palpable progress. We also look forward to future experiments to improve mobile coding keyboards, which would maximize the student's efficiency. Finally, as further work, we encourage exploring a distributed GPU real-time processing framework to leverage the trend of having GPU-powered devices and provide further more advanced use cases of the computational notebook.

References

[1] Apache Zeppelin, official website, URL: https://zeppelin.apache.org, accessed at 06.15.2020.

[2] Brython, a Python 3 implementation for client-side web programming, URL: https://brython.info, accessed at 06.19.2020.

[3] Cardoso, Alberto & Leitao, Joaquim & Teixeira, Cesar. (2019). Using the Jupyter Notebook as a Tool to Support the Teaching and Learning Processes in Engineering Courses.

[4] Felix, Hao, JSCPP, C++ interpreter written in JavaScript, URL: https://github.com/felixhao28/JSCPP, accessed at 06.19.2020.

[5] Google Colab, official website, URL: https://colab.research.google.com, accessed at 06.14.2020.

[6] GNU Bison, official website, URL: https://www.gnu.org/software/bison, accessed at 06.10.2020.

[7] Guo, Philip. (2013). Online python tutor: Embeddable web-based program visualization for cs education.

[8] Jison, JavaScript parser generator, URL: https://zaa.ch/jison/docs, accessed at 06.10.2020.

[9] Khalid, Yasoob, Running Python in the Browser, URL: https://yasoob.me/2019/05/22/running-python-in-the-browser, accessed at 06.26.2020

[10] Microsoft Azure Notebooks, official website, URL: https://notebooks.azure.com, accessed at 06.14.2020.

[11] Nteract, official repository and source code, URL: https://github.com/nteract/nteract, accessed at 06.14.2020.

[12] Project Jupyter, URL: https://jupyter.org, accessed at 06.03.2020.

[13] Pyodide, the Python scientific stack compiled to WebAssembly, official repository and source code, URL: https://github.com/iodide-project/pyodide, accessed at 06.02.2020.

[14] PyPy compiled into JavaScript, official repository and source code, URL: https://github.com/pypyjs/pypyjs, accessed at 06.05.2020.

[15] PyScript, Python to Javascript compiler, URL: https://github.com/ avi-noamr/pyscript, accessed at 06.10.2020.

[16] Root Data Analysis Framework, official website, URL: https://root.cern.ch, accessed at 06.14.2020.

[17] Skulpt, in-browser implementation of Python, URL: https://skulpt.org, accessed at 06.07.2020.

[18] Brython Speed Test, URL: http://brython.info/speed, accessed at 08.07.2020.

[19] Swift Playgrounds, official website, URL: https://developer.apple.com/swift-playgrounds, accessed at 06.14.2020.