Building a Columnar Analysis Demonstrator for ATLAS PHYSLITE Open Data using the Python Ecosystem

KyungEon Choi [1], *Matthew* Feickert [2,*], *Nikolai* Hartmann [3], *Lukas* Heinrich [4], *Alexander* Held [2], *Evangelos* Kourlitis [4], *Nils* Krumnack [2], *Giordon* Stark [5], *Matthias* Vigl [4], and *Gordon* Watts [6] on behalf of the ATLAS Computing Activity

[1]University of Texas at Austin, Austin, Texas, USA
[2]University of Wisconsin-Madison, Madison, Wisconsin, USA
[3]Ludwig Maximilians Universitat, Munich, Germany
[4]Technical University of Munich, Munich, Germany
[5]Santa Cruz Institute for Particle Physics, Santa Cruz, California, USA
[6]University of Washington, Seattle, Washington, USA

Abstract. The ATLAS experiment is in the process of developing a columnar analysis demonstrator, which takes advantage of the Python ecosystem of data science tools. This project is inspired by the analysis demonstrator from IRIS-HEP. The demonstrator employs PHYSLITE OpenData from the ATLAS collaboration, the new Run 3 compact ATLAS analysis data format. The tight integration of ROOT features within PHYSLITE presents unique challenges when integrating with the Python analysis ecosystem. The demonstrator is constructed from ATLAS PHYSLITE OpenData, ensuring the accessibility and reproducibility of the analysis. The analysis pipeline of the demonstrator incorporates a comprehensive suite of tools and libraries. These include uproot for data reading, awkward-array for data manipulation, Dask for parallel computing, and hist for histogram processing. For the purpose of statistical analysis, the pipeline integrates `cabinetry` and `pyhf`, providing a robust toolkit for analysis. A significant component of this project is the custom application of corrections, scale factors, and systematic errors using ATLAS software. The infrastructure and methodology for these applications will be discussed in detail during the presentation, underscoring the adaptability of the Python ecosystem for high energy physics analysis.

ATL-SOFT-PROC-2025-025
09 February 2025

1 Introduction

As the High Luminosity LHC [1] (HL-LHC) era approaches, the ATLAS experiment [2] has created a HL-LHC focused research and development program [3] for software and computing upgrades to address the challenges and opportunities outlined in the ATLAS Software and Computing HL-LHC Roadmap [4]. From the 2022 computing model projections, ATLAS does not anticipate being able to store all analysis computations on disk, as even under the "aggressive" R&D scenario, sustained year-on-year budget increases of more than

*Corresponding author e-mail: matthew.feickert@cern.ch

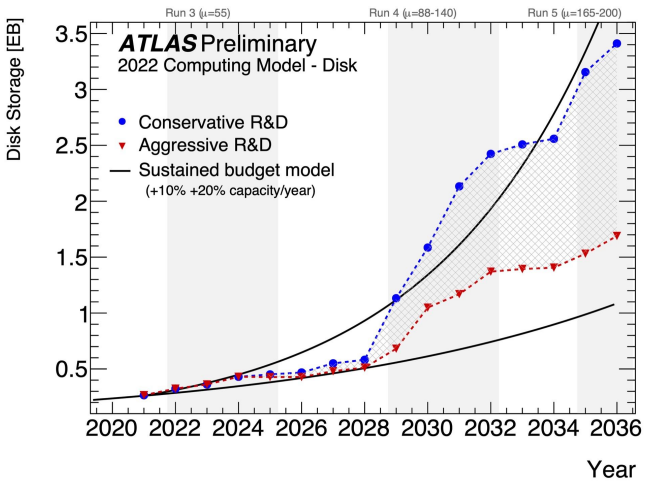

Figure 1: Projected evolution of disk usage from 2020 until 2036, under the conservative (blue) and aggressive (red) R&D scenarios. The grey hatched shading between the red and blue lines illustrates the range of resources consumption if the aggressive scenario is only partially achieved. The black lines indicate the impact of sustained year-on-year budget increases, and improvements in new hardware, that together amount to a capacity increase of 10% (lower line) and 20% (upper line). The vertical shaded bands indicate periods during which ATLAS will be taking data [4].

10% would be required to meet the disk storage quota required, as seen in Figure 1. To prepare for this reality, ATLAS has begun to explore strategies of trading disk for compute by performing "on-the-fly" computations for analysis quantities when possible, rather than reading them from disk. As part of this approach, ATLAS has been researching the use of columnar analysis — array programming for data analysis — for full analysis workflows, given its compute efficient batch data operations and its applications in different software environments and ecosystems.

2 Columnar Analysis Demonstrator

One of the established research projects for columnar analysis is the development of an ATLAS columnar analysis demonstrator. This demonstrator is inspired by the Institute for Research and Innovation in Software for High Energy Physics (IRIS-HEP) [5, 6] Analysis Grand Challenge [7], which leverages the "PyHEP" ecosystem of data science and analysis tools developed by IRIS-HEP and Scikit-HEP [8]. These ecosystems of tools build upon and extend the broader Scientific Python ecosystem, with mature columnar analysis libraries, to provide functionality at each step of the columnar analysis pipeline: data query and access, data file I/O and columnar access, data transformation and histogramming, distributed analysis frameworks, statistical modelling and inference, and analysis reinterpretation.

The ATLAS Run 4 Analysis Model is built around the PHYSLITE common reduced data format [9, 10]. PHYSLITE is a monolithic file format — intended to support around 80% of all physics analyses in Run 4 — that contains already-calibrated physics objects for fast analysis, and allows for direct support without the need to create derived ROOT ntuples for analysis. The demonstrator uses ATLAS PHYSLITE open data [11] — released

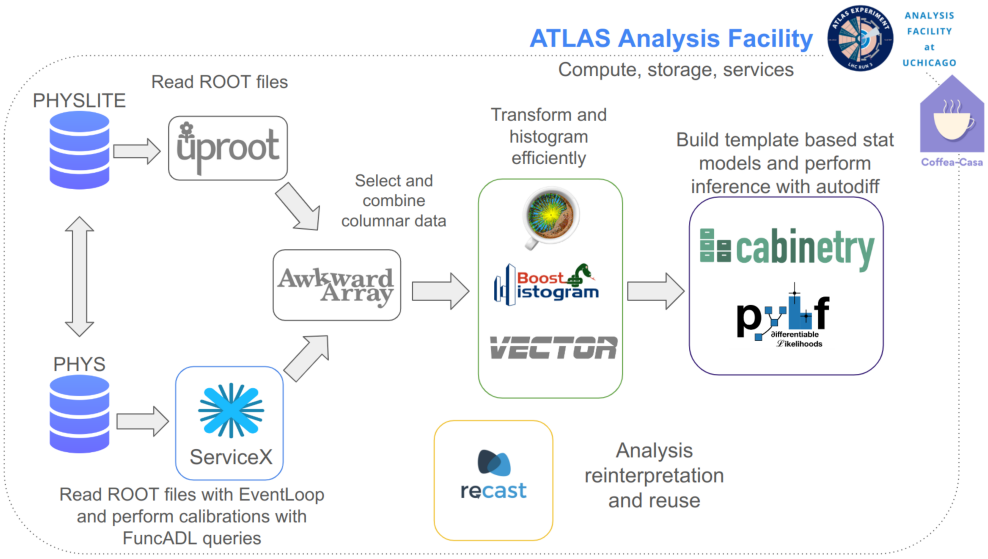

Figure 2: Schematic outline of an ATLAS columnar analysis demonstrator able to operate on PHYS and PHYSLITE data formats using columnar analysis tools from the PyHEP ecosystem. The demonstrator is designed to be deployed and run at an ATLAS analysis facility, such as the University of Chicago Analysis Facility or a Coffea-casa deployment.

for research use for the first time in 2024 — to ensure the accessibility and reproducibility of the technical demonstrator analysis. PHYSLITE has tight integration with ROOT features, to the extent that raw PHYSLITE is not easily loadable outside of ROOT in general. Integrating PHYSLITE data access with the Python analysis ecosystem presents unique challenges, such as correctly reading unsplit objects with custom serialization and layout, e.g. `ElementLinks` [12] — a smart pointer type for providing persistent references to a specific element in an object collection ("container") — and trigger data. However, the Awkward Array [13] library supports "behaviors" which allow for efficiently reinterpreting data structures on the fly, and can be used by Uproot [14] for I/O of custom serializations (e.g. PHYSLITE). Through contributions by ATLAS members large amounts of PHYSLITE are now supported by both Uproot and Coffea [15, 16] which allows for most workflows to proceed with small alterations [17]. This allows for the demonstrator workflow to operate on most PHYSLITE files directly with Uproot, as seen in Figure 2. For files in the PHYS ATLAS Run 3 Analysis Model data format [10], FuncADL [18, 19] and ServiceX [20–22] can be used to create a distributed ROOT [23] data transformation service to read the data and perform calibrations. The result of the transform is an in-memory columnar representation equivalent to if the column had been read from PHYSLITE. As PHYSLITE already contains all of the calibrated physics objects, it is preferable to use PHYSLITE whenever possible.

3 Challenges and Opportunities

Data access can become challenging in PHYSLITE when branches with custom objects need to be read or when full systematics need to be handled. As columnar analysis processes events in batches, this requires the ATLAS Combined Performance (CP) tools and algorithms that handle these challenges to also support batched operations and have the appropriate inter-

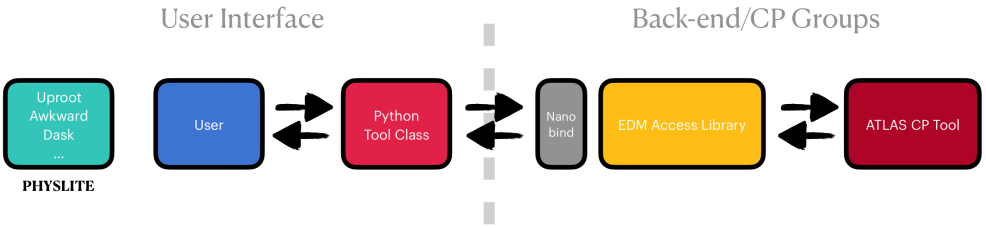

Figure 3: Cartoon of the data interface layers for the Pythonic interface to the columnar CP tools [25]. The user is presented with a high-level Pythonic API for operating on columns of PHYSLITE data (e.g. as NumPy or Awkward arrays) that has a similar design to common Python array libraries. Through `nanobind` [24] zero-copy buffers, the data columns are passed directly to the performant C++ columnar CP tools for computation, and the results are returned to the user level as transformed arrays.

faces. The current Run 3 model for CP tools is to operate on the ATLAS xAOD event data model (EDM) for all the calculations per event and then to write the systematics to disk for future access [10]. This process is I/O intensive and computationally inefficient, but has excellent physics performance and configurability for the current ATLAS analysis model. As a data access layer the xAOD interface also makes assumptions about the underlying I/O layer, memory model, and processing environment, which do not match well to the expectations of columnar frameworks. The challenge for a fully columnar CP tool framework is to adapt to the on-the-fly computation of the columnar paradigm — furthering the "trade disk for compute" strategy — while still being performant enough to not be a bottleneck.

The refactoring process for columnar tools in the ATLAS Analysis Model Group (AMG) has also offered design improvement opportunities to create more Pythonic end-user APIs for the ATLAS CP tools. As seen in Figure 3, creating Pythonic APIs for users allows for integration with the broader scientific Python ecosystem, and through efficient binding layers all data can be passed through to the C++ CP tools for efficient computation, and then the results can be exposed to the users again. By using the `nanobind` [24] C++-to-Python bindings library it becomes possible to have zero-copy operations to and from n-dimensional array libraries in Python, including those which support hardware accelerators like GPUs, and full design control of the high-level user API. The API design ability is quite powerful, as it allows for unification of interfaces to CP tools without requiring individual CP tools to redesign their C++ APIs.

4 Columnar CP Prototypes

As a first (`v1`) prototype of this capability, a standalone columnar implementation of the ATLAS electron and photon (Egamma) CP tool with zero-copy `nanobind` Python bindings was created to compute on-the-fly systematics variations for the dilepton system mass in e^+e^- final states. Uproot was used to load ATLAS $Z \to \ell\ell$ simulation PHYSLITE files into Awkward arrays, and the e^+e^- event selections were applied with Coffea. The columnar Egamma tool was initialized through the Pythonic interface, `atlascp.EgammaTools`, and then passed Awkward arrays of electrons to compute on-the-fly systematic variations of the electron reconstruction efficiency scale factors and energy correction resolution and scale, from which the corrected dilepton mass was computed. The computations were additionally scaled out with `dask-awkward` [26] on the University of Chicago ATLAS Analysis Facility to minimize the compute wall time, with the gathered results visualized in Figure 4.

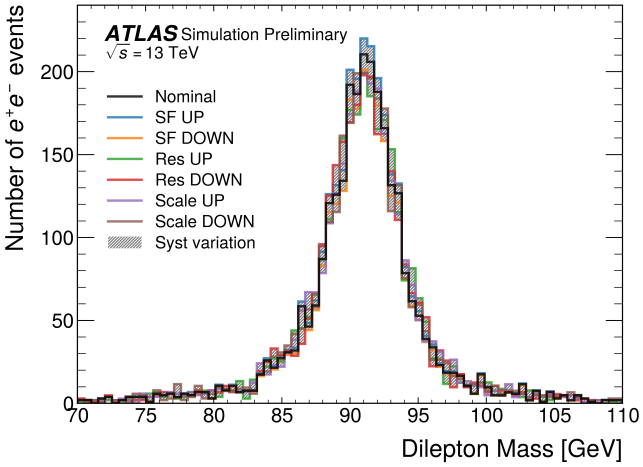

Figure 4: Histograms of the dilepton mass of selected $Z \to e^+e^-$ events in ATLAS simulation under on-the-fly systematic variations of the electron reconstruction efficiency scale factor (SF) — the product of the reconstruction, identification, and isolation scale factors — and energy correction resolution (Res) and scale (Scale) [25].

This preliminary research was required as there was no "'zero action" option to determine the viability of the proposed design *a priori*. The `v1` prototype established the foundations of what was possible with new tooling and that Pythonic interfaces to CP tools could be written without large amounts of work or deep knowledge of underlying CP tool design. This showed a promising direction, but additional work is needed to achieve the necessary performance, integration, and support required for use.

A `v2` "Columnar Athena" prototype [27] has been started to expand on the scope of the `v1` prototype. This moves the development of the columnar tools and interfaces from individual standalone examples into a unified system under the ATLAS Athena framework [28] and migrates the ATLAS CP tools to a columnar backend without breaking the existing workflows using the EDM models. It additionally adds infrastructure support for development of columnar analysis tools by adding `nanobind` to the ATLAS Externals tooling distributed as part of the ATLAS Athena Analysis Releases. While currently under active development, the `v2` prototype will allow for full scale integration and performance tests of the columnar CP tools and interfaces.

5 Preliminary Performance Tests

During the ongoing refactor for the `v2` prototype, preliminary integrated benchmarks have been created to measure the time spent in each tool (excluding I/O) in comparison with the xAOD model. While direct one-to-one comparisons are not possible given the inherent differences in the memory and I/O models for data access, the tests have been designed to be as close as possible. The benchmarks compare the same version of each CP tool, only use the `C++` code (no Python is involved so as to isolate the `C++` performance), and the time for the xAOD model includes the event store access overhead (which is per-event for the xAOD model and per-batch for the columnar model). The time for I/O and connecting columns is

also not included in the performance comparisons, as this has not been optimized in the current tests and will not provide useful information, and so is excluded from the benchmark. The benchmarks show substantial speedups for the migrated tools with the columnar implementations ranging from being *2-4x faster* than the xAOD interface. The specific reasons for the speedups are currently being investigated fully, but preliminary checks show a relation with event store access and differences in accessing individual variables.

6 Future Work and Decisions

The development of the `v2` prototype of the columnar CP tools is ongoing and will continue to be publicly developed and benchmarked with future updates to the community from ATLAS. The `v1` and `v2` prototypes have already demonstrated that adopting a columnar implementation for the backend CP tools allows for a columnar analysis paradigm to be possible, and that the ongoing integration of `nanobind` bindings bridges the gap between `C++` and Python for high performance analysis while allowing freedom of Pythonic API design to allow for higher level analysis thinking by end users. This development is critical for a full columnar analysis ATLAS demonstrator that is able to interface fully with the scientific Python and PyHEP ecosystems.

When the ATLAS CP tools were created 10-15 years ago during Run 1, they were designed to be as framework independent as possible, be able to be run by themselves, and to interface with the ATLAS EDM. In the time since, they have been battle tested, and through through hundreds of analyses are now extremely well understood and provide excellent physics performance. This creates a strong desire in ATLAS to maintain these valuable tools and their existing functionality, making the rewrite cost of them to a `correctionlib` [29] paradigm currently too high. The addition of columnar support to the CP tools for the new analysis model though requires cracking open the "black box" implementations of the tools. Being confronted with legacy code decisions further highlights the columnar prototype design decisions and the design opportunities during the tool migrations. It also prompts more ambitious questions, like what would be required to make the ATLAS CP tools a standalone `pip` [30] or `conda` [31] installable Python package? While this might seem like an ambition for Run 5 of the LHC or beyond, given the significant improvements in packaging technology in the Python ecosystem, like `scikit-build-core` [32], this proposal is perhaps achievable on a significantly shorter time scale. As an example, using this strategy, the ROOT team is currently researching the ability to distribute ROOT as a Python package [33]. If these approaches are successful, it will allow for a greater level of modularity in the way ATLAS analyses are performed and a path towards greater interoperability with other tools as the ATALS CP tools will have become another viable tool in the broader PyHEP ecosystem.

7 Acknowledgements

Matthew Feickert, Alexander Held, and Gordon Watts are supported by the U.S. National Science Foundation (NSF) under Cooperative Agreement OAC-1836650 and PHY-2323298 (IRIS-HEP). Lukas Heinrich and Evangelos Kourlitis are supported by the Excellence Cluster ORIGINS, which is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC-2094-390783311 and by the German Federal Ministry of Education and Research Project 05H2021 (ErUM-FSP T02) - "Run 3 von ATLAS am LHC: Analysis Infrastructure for the ATLAS Detektor at the LHC".

References

[1] I. Zurbano Fernandez et al., High-Luminosity Large Hadron Collider (HL-LHC): Technical design report, CERN Yellow Reports: Monographs **10/2020** (2020), https://cds.cern.ch/record/2749422/. 10.23731/CYRM-2020-0010

[2] ATLAS Collaboration, The ATLAS Experiment at the CERN Large Hadron Collider, JINST **3**, S08003 (2008). 10.1088/1748-0221/3/08/S08003

[3] ATLAS Collaboration, ATLAS HL-LHC Computing Conceptual Design Report, CERN-LHCC-2020-015; LHCC-G-178 (2020), https://cds.cern.ch/record/2729668

[4] ATLAS Collaboration, ATLAS Software and Computing HL-LHC Roadmap, CERN-LHCC-2022-005; LHCC-G-182 (2022), https://cds.cern.ch/record/2802918

[5] P. Elmer, M. Neubauer, M.D. Sokoloff, Strategic Plan for a Scientific Software Innovation Institute (S2I2) for High Energy Physics (2017), https://arxiv.org/abs/1712.06592, 1712.06592.

[6] J. Albrecht et al. (HEP Software Foundation), A Roadmap for HEP Software and Computing R&D for the 2020s, Comput. Softw. Big Sci. **3**, 7 (2019), 1712.06982. 10.1007/s41781-018-0018-8

[7] A. Held, O. Shadura, The IRIS-HEP Analysis Grand Challenge, PoS **ICHEP2022**, 235 (2022). 10.22323/1.414.0235

[8] E. Rodrigues et al., The Scikit HEP Project – overview and prospects, EPJ Web Conf. **245**, 06028 (2020), 2007.03577. 10.1051/epjconf/202024506028

[9] J. Schaarschmidt, J. Catmore, J. Elmsheuser, L. Heinrich, N. Krumnack, S. Mete, N. Ozturk, PHYSLITE - A new reduced common data format for ATLAS, EPJ Web Conf. **295**, 06017 (2024), https://cds.cern.ch/record/2870350/. 10.1051/epjconf/202429506017

[10] ATLAS Collaboration, Software and computing for Run 3 of the ATLAS experiment at the LHC (2024), https://arxiv.org/abs/2404.06335, 2404.06335.

[11] ATLAS Collaboration, The First Release of ATLAS Open Data for Research, ATL-OREACH-PROC-2024-005 (2024), https://cds.cern.ch/record/2911158

[12] N. Hartmann, J. Elmsheuser, G. Duckeck, Columnar data analysis with ATLAS analysis formats, EPJ Web Conf. **251**, 03001 (2021). 10.1051/epjconf/202125103001

[13] J. Pivarski, I. Osborne, I. Ifrim, H. Schreiner, A. Hollands, A. Biswas, P. Das, S. Roy Choudhury, N. Smith, M. Goyal, Awkward Array (2018), https://doi.org/10.5281/zenodo.4341376

[14] J. Pivarski, H. Schreiner, A. Hollands, P. Das, K. Kothari, A. Roy, J. Ling, N. Smith, C. Burr, G. Stark, Uproot (2017), https://doi.org/10.5281/zenodo.4340632

[15] L. Gray, N. Smith, A. Novak, P. Fackeldey, B. Tovar, Y.M. Chen, G. Watts, I. Krommydas, coffea (2023), https://doi.org/10.5281/zenodo.3266454

[16] N. Smith et al. (CMS), Coffea: Columnar Object Framework For Effective Analysis, EPJ Web Conf. **245**, 06012 (2020), 2008.12712. 10.1051/epjconf/202024506012

[17] K. Choi, M. Feickert, L. Gray, A. Held, V. Kourlitis, A. Peixoto, J. Pivarski, O. Shadura, G. Watts, US ATLAS / IRIS-HEP Analysis Software Training Event 2024 (2024), Note in particular contributions on PHYSLITE and Coffea., https://indico.cern.ch/event/1376945/

[18] G. Watts, func_adl (2024), https://github.com/iris-hep/func_adl

[19] M. Proffitt, G. Watts, FuncADL: Functional Analysis Description Language, EPJ Web Conf. **251**, 03068 (2021), 2103.02432. 10.1051/epjconf/202125103068

[20] B. Galewsky, A. Eckart, G. Watts, S. Thapa, P. Onyisi, M. Weinberg, I. Vukotic, ServiceX (2024), https://github.com/ssl-hep/ServiceX

[21] G. Watts, K. Choi, P. Onyisi, K. Mahajan, B. Galewsky, M. Feickert, ServiceX Client Library (2024), https://github.com/ssl-hep/ServiceX_frontend

[22] B. Galewsky, R. Gardner, L. Gray, M. Neubauer, J. Pivarski, M. Proffitt, I. Vukotic, G. Watts, M. Weinberg, ServiceX A Distributed, Caching, Columnar Data Delivery Service, EPJ Web Conf. **245**, 04043 (2020). 10.1051/epjconf/202024504043

[23] R. Brun, F. Rademakers, ROOT: An object oriented data analysis framework, Nucl. Instrum. Meth. A **389**, 81 (1997). 10.1016/S0168-9002(97)00048-X

[24] W. Jakob, nanobind: tiny and efficient C++/Python bindings (2022), https://github.com/wjakob/nanobind

[25] M. Feickert, N. Hartman, L. Heinrich, A. Held, V. Kourlitis, N. Krumnack, G. Stark, M. Vigl, G. Watts, Using Legacy ATLAS C++ Calibration Tools in Modern Columnar Analysis Environments (2024), 22nd International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2024), https://indico.cern.ch/event/1330797/contributions/5796636/

[26] D. Davis, L. Gray, M. Durant, A. Hollands, dask-awkward (2024), https://github.com/dask-contrib/dask-awkward

[27] ATLAS Collaboration, Columnar Athena (2024), https://gitlab.cern.ch/atlas-asg/columnar-athena

[28] ATLAS Collaboration, Athena (2023), https://gitlab.cern.ch/atlas/athena

[29] N. Smith, correctionlib (2024), https://github.com/cms-nanoAOD/correctionlib

[30] The pip developers, pip (2024), https://github.com/pypa/pip

[31] conda contributors, conda: A system-level, binary package and environment manager running on all major operating systems and platforms., https://github.com/conda/conda

[32] H. Schreiner, III, J.C. Fillion-Robin, M. McCormick, Scikit-build-core (2024), SciPy 2024, https://doi.org/10.25080/FMKR8387

[33] V.E. Padulano, pip install ROOT: experiences making a complex multi-language package accessible for Python users (2024), 27th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2024), https://indico.cern.ch/event/1338689/contributions/6010410/