Toward a Live, Rich, Composable, and Collaborative Planetary Compute Engine

ALEXANDER BANDUKWALA, Unaffiliated, USA
ANDREW BLINN, University of Michigan, USA
CYRUS OMAR, University of Michigan, USA

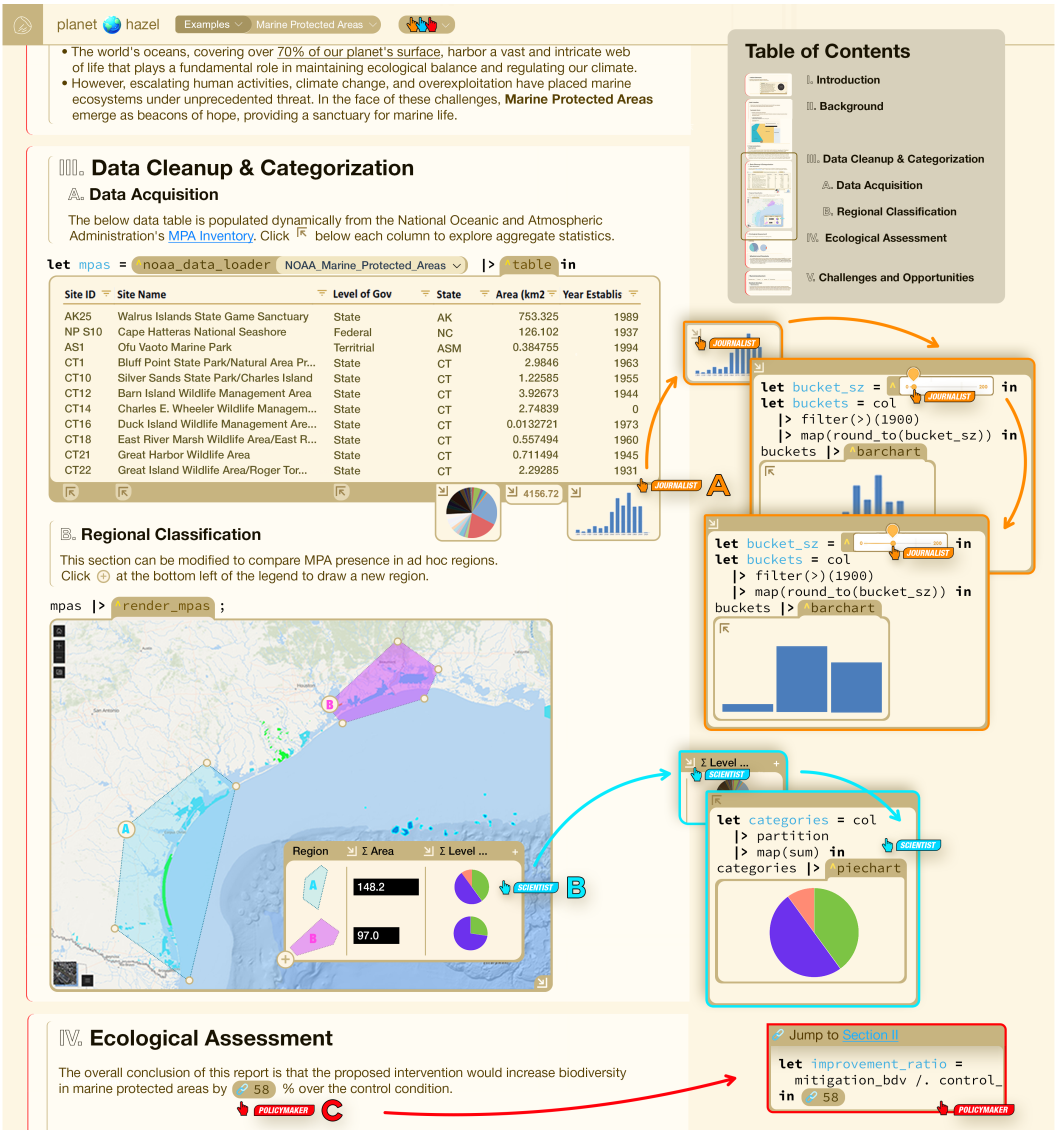
Addressing the climate crisis poses many computing challenges—including continuous data ingestion, transformation, and analysis—intertwined with significant human factors challenges—planetary computing is a large-scale effort involving many stakeholders, many of whom are not CS experts but rather scientists, policymakers, journalists, and members of the public. Contemporary programming ecosystems, which offer a patchwork of aging low-level tools, may not be up to the task of developing a modern *planetary compute engine* [Holcomb et al. 2023]. In particular, we believe that a planetary computing engine must be **live**, **rich**, and **composable** [Horowitz and Heer 2023] to systematically address critical limitations of existing patchwork solutions while enabling new capabilities. Additionally, it must be ubiquitously and accessibly **collaborative**, operating as a shared medium for initial data ingestion and cataloging by data engineers, exploration and interactive analysis by ecologists, curation of methods maintained by statisticians, active decision-making by policy experts, and investigation and critique by journalists and the public. We believe the Hazel project (https://hazel.org/) provides a uniquely practical foundation for designing a next-generation live, rich, composable, and collaborative planetary compute engine. Fig. 1 is a mockup of Hazel, demonstrating our initial vision for some of the key capabilities outlined above. Let us go through these in more detail.

Live environments provide the programmer with feedback based on dynamic program behavior continuously, during the editing process [Tanimoto 2013]. This is critical to support exploratory data analysis, as has been shown by the proliferation of live computational notebooks such as Jupyter [Perez and Granger 2007]. In climate analysis, it is also important that analyses can ingest live data from various data sources. Hazel is a functional programming environment that supports *totally live evaluation*, i.e. there are never any gaps in execution even in the presence of localized errors [Omar et al. 2019]. This provides a unique platform for productively performing exploratory climate data analysis and ensuring that these analyses are kept up-to-date as new data is ingested and new analyses are incrementally developed and systematically compared [Omar et al. 2014]. Hazel's mathematically structured (i.e. functional) execution model stands in contrast to that of imperative languages like Python, where unnecessary reliance on state leads to (1) the problem of results being inconsistent with the code as it appears, limiting reproducibility, and (2) difficulties with automatic incremental execution [Chattopadhyay et al. 2020]. In the example in Fig. 1, the data is loaded from an external source and kept live and up-to-date, with updates functioning essentially as edits to a literal table in the program and with downstream re-execution occurring automatically. Scaling up Hazel's live execution engine to support this user workflow even for large and rapidly updating datasets using distributed computing resources is a potentially fruitful research direction.

Climate science uses a number of domain-specific data structures and visual representations, such as maps, diagrams, complex plot structures, and compositions of these. Many existing environments including Jupyter allow for domain-specific data visualizations to be *generated* from an end-user analysis. However, these visualizations are usually only minimally interactive and need

---

Authors' addresses: Alexander Bandukwala, alexander@bandukwala.me, Unaffiliated, USA; Andrew Blinn, blinnand@umich.edu, University of Michigan, USA; Cyrus Omar, comar@umich.edu, University of Michigan, USA.



- The world's oceans, covering over 70% of our planet's surface, harbor a vast and intricate web of life that plays a fundamental role in maintaining ecological balance and regulating our climate.
- However, escalating human activities, climate change, and overexploitation have placed marine ecosystems under unprecedented threat. In the face of these challenges, **Marine Protected Areas** emerge as beacons of hope, providing a sanctuary for marine life.

III. **Data Cleanup & Categorization**

A. **Data Acquisition**

The below data table is populated dynamically from the National Oceanic and Atmospheric Administration's MPA Inventory. Click ⇱ below each column to explore aggregate statistics.

```
let mpas = ^noaa_data_loader NOAA_Marine_Protected_Areas |> ^table in
```

Site ID	Site Name	Level of Gov	State	Area (km2	Year Establis
AK25	Walrus Islands State Game Sanctuary	State	AK	753.325	1989
NP S10	Cape Hatteras National Seashore	Federal	NC	126.102	1937
AS1	Ofu Vaoto Marine Park	Territrial	ASM	0.384755	1994
CT1	Bluff Point State Park/Natural Area Pr...	State	CT	2.9846	1963
CT10	Silver Sands State Park/Charles Island	State	CT	1.22585	1955
CT12	Barn Island Wildlife Management Area	State	CT	3.92673	1944
CT14	Charles E. Wheeler Wildlife Managem...	State	CT	2.74839	0
CT16	Duck Island Wildlife Management Are...	State	CT	0.0132721	1973
CT18	East River Marsh Wildlife Area/East R...	State	CT	0.557494	1960
CT21	Great Harbor Wildlife Area	State	CT	0.711494	1945
CT22	Great Island Wildlife Area/Roger Tor...	State	CT	2.29285	1931

B. **Regional Classification**

This section can be modified to compare MPA presence in ad hoc regions. Click ⊕ at the bottom left of the legend to draw a new region.

```
mpas |> ^render_mpas ;
```

```
let bucket_sz = ^ in
let buckets = col
  |> filter(>)(1900)
  |> map(round_to(bucket_sz)) in
buckets |> ^barchart
```

```
let bucket_sz = ^ in
let buckets = col
  |> filter(>)(1900)
  |> map(round_to(bucket_sz)) in
buckets |> ^barchart
```

```
let categories = col
  |> partition
  |> map(sum) in
categories |> ^piechart
```

IV. **Ecological Assessment**

The overall conclusion of this report is that the proposed intervention would increase biodiversity in marine protected areas by 58 % over the control condition.

Jump to Section II

```
let improvement_ratio =
  mitigation_bdv /. control_
in 58
```

Fig. 1. A depiction of an analytical exploration of Marine Protected Areas in Planet Hazel, a hypothetical version of Hazel geared towards *planetary computing*. A spectrum of stakeholders collaborate over live data in real time, composing visualizations, code, and rich text to create a cooperative computational artifact. Technical stakeholders may use general-purpose programming and - by selectively exposing parts of their code as customizable interactive visualizations - extend the capabilities of non-technical stakeholders and foster a skill-continuous medium for analysis, presentation, and conversation. The right-hand sub-figures depict three stakeholder scenarios: In (A) a nontechnical journalist uses an embedded GUI to change a graph without editing code. A data scientist (B) inspects a sub-component of a parametric visualization to update the code defining it. A policymaker (C), checks their understanding of a measurement by following a transcluded figure back to its definition.

to be built and modified using generic plain-text code. Hazel, in contrast, supports *livelits* (live literals)[Omar et al. 2021], which allow programs themselves to contain **live** and **rich** domain-specific representations that can be fed data and manipulated directly, such as by navigating and selecting GIS (geographic information system) data using a map interface or by manipulating sliders, plot parameters, and tables such as in Fig. 1. Each livelit can also *generate data* that can be used in downstream computations. The move towards more domain-oriented editing could lower the need for technical expertise to make simple localized changes to a program while also reducing the cognitive overhead for expert and novice technical users alike.

In addition to richness and liveness, livelits support **composability** by allowing for embedded sub-expressions within livelit GUIs called *splices*, which can be filled by the user with arbitrary symbolic code of a specified type or themselves be livelits. For example, the table in Fig 1. is a livelit containing a footer that can be populated with splices that operate on the associated columns' contents. This allows for a user to quickly visualize the distribution a particular variable may take in a dataset. The interface provides a form of gradual technical sophistication where a novice user can just see the visualization, an intermediate user can expand into the splice definition to alter configuration parameters or switch to a different visualization, and an expert user could define new splices or livelits and access the full expressive capabilities of the language. The designer of the table livelit need not anticipate all of these potential uses due to the magic of compositionality! Compositionality also allows us to extend into data-driven documents—Fig. 1 illustrates *transclusions* [Nelson 1981] of real data into rich explanatory text removing the possibility of data getting out of sync.

Hazel is built atop a typed functional language that integrates the PL community's state-of-the-art understanding of **composability** in computing using a small number of orthogonal logical data primitives, namely products, sums, and functions (with `|>` serving as pipelining, i.e. reverse function application). This simplicity and connection to basic mathematics means that scientists will not need to learn as many new ideas, like object-oriented programming, to fully understand the computational model (it is, essentially, a refinement of the Excel formula language!) Modern compilation techniques are able to fuse, parallelize, and distribute pure functional code much more easily than imperative code [Chin 1992]. This could build on ideas from the Ark project [Holcomb et al. 2023] which enables the definition of dataflow pipelines to streamline the ingestion, transformation, and publication of climate analyses by explicitly breaking out pipelines into pure computations and data inputs. The system remains accessible to non-expert users by allowing for analyses to be performed in external systems.

Climate science is inherently a multidisciplinary international collaboration. To support this, we believe there needs to be a large-scale collaborative compute engine—essentially, a computational Wikipedia—that allows all stakeholders to operate in a common environment without strict siloing of capability or information, nor unnecessary friction at interfaces. Adding multi-user editor support to Hazel would allow for different stakeholders (perhaps also including aligned AI agents partnered with human stakeholders) to **collaborate** on analyzing and cleaning up data, making and comparing policy proposals, and improving basic methods in real-time. Direct collaboration using real data speeds up the rate and minimizes the risk of miscommunication leading to better outcomes. A shared environment also creates opportunities for spontaneous collaboration on innovative solutions that are otherwise impossible. For example, the figure shows a journalist making use of modifiable parameters in a report being collaboratively edited by a scientist. Critical to ensuring liveness in the presence of collaboration is resilience to errors, which as mentioned above is a hallmark of the Hazel environment: one collaborator leaving a syntax or type error somewhere in the Wikipedia-sized "planetary program" will not break everyone's build.

The proposed talk will demonstrate some of Hazel's current capabilities in this direction and discuss several future research directions. We will conclude with a call to action for how the community can work towards pursuing this vision and other avenues for future research and collaboration.

REFERENCES

Souti Chattopadhyay, Ishita Prasad, Austin Z. Henley, Anita Sarma, and Titus Barik. 2020. What's Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities. 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, 1–12. https://doi.org/10.1145/3313831.3376729

Wei-Ngan Chin. 1992. Safe Fusion of Functional Expressions. *SIGPLAN Lisp Pointers* V, 1 (jan 1992), 11–20. https://doi.org/10.1145/141478.141494

Amelia Holcomb, Michael Dales, Patrick Ferris, Sadiq Jaffer, Thomas Swinfield, Alison Eyres, Andrew Balmford, David Coomes, Srinivasan Keshav, and Anil Madhavapeddy. 2023. A Case for Planetary Computing. arXiv:2303.04501 [cs.DC]

Joshua Horowitz and Jeffrey Heer. 2023. Live, Rich, and Composable: Qualities for Programming Beyond Static Text. arXiv:2303.06777 [cs.PL]

Theodor Nelson. 1981. *Literary Machines.* Mindful Pr.

Cyrus Omar, Jonathan Aldrich, and Richard C. Gerkin. 2014. Collaborative Infrastructure for Test-Driven Scientific Model Validation. In *Companion Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE Companion 2014)*. Association for Computing Machinery, New York, NY, USA, 524–527. https://doi.org/10.1145/2591062.2591129

Cyrus Omar, David Moon, Andrew Blinn, Ian Voysey, Nick Collins, and Ravi Chugh. 2021. Filling typed holes with live GUIs. In *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, Stephen N. Freund and Eran Yahav (Eds.). ACM, 511–525. https://doi.org/10.1145/3453483.3454059

Cyrus Omar, Ian Voysey, Ravi Chugh, and Matthew A. Hammer. 2019. Live Functional Programming with Typed Holes. *Proc. ACM Program. Lang.* 3, POPL, Article 14 (Jan. 2019), 32 pages. https://doi.org/10.1145/3290327

Fernando Perez and Brian E. Granger. 2007. IPython: A System for Interactive Scientific Computing. *Computing in Science Engineering* 9, 3 (2007), 21–29. https://doi.org/10.1109/MCSE.2007.53

Steven L. Tanimoto. 2013. A perspective on the evolution of live programming. In *International Workshop on Live Programming (LIVE)*.