Yirgacheffe: A Declarative Approach to Geospatial Data

Michael Winston Dales
University of Cambridge
Cambridge, United Kingdom
mwd24@cam.ac.uk

Alison Eyres
University of Cambridge
Cambridge, United Kingdom
ae491@cam.ac.uk

Patrick Ferris
University of Cambridge
Cambridge, United Kingdom
pf341@cam.ac.uk

Francesca A. Ridley
Newcastle University
Newcastle, United Kingdom
Francesca.Ridley@newcastle.ac.uk

Simon Tarr
IUCN (International Union for
Conservation of Nature)
Cambridge, United Kingdom
Simon.Tarr@iucn.org

Anil Madhavapeddy
University of Cambridge
Cambridge, United Kingdom
avsm2@cam.ac.uk

Abstract

We present Yirgacheffe, a declarative geospatial library that allows spatial algorithms to be implemented concisely, supports parallel execution, and avoids common errors by automatically handling data (large geospatial rasters) and resources (cores, memory, GPUs). Our primary user domain comprises ecologists, where a typical problem involves cleaning messy occurrence data, overlaying it over tiled rasters, combining layers, and deriving actionable insights from the results. We describe the successes of this approach towards driving key pipelines related to global biodiversity and describe the capability gaps that remain, hoping to motivate more research into geospatial domain-specific languages.

CCS Concepts: • **Information systems** → *Geographic information systems*; • **Software and its engineering** → **Domain specific languages**.

Keywords: Declarative, Geospatial, Python, Biodiversity

ACM Reference Format:
Michael Winston Dales, Alison Eyres, Patrick Ferris, Francesca A. Ridley, Simon Tarr, and Anil Madhavapeddy. 2025. Yirgacheffe: A Declarative Approach to Geospatial Data. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Programming for the Planet (PROPL '25), October 12–18, 2025, Singapore, Singapore.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3759536.3763806

1 Introduction

The fields of ecology and conservation science are rapidly adopting data-driven approaches [35], and therefore it is vital to empower their practitioners with useful computing tools. Ecologists are "vernacular" software developers: experts in a domain other than computer science who need to write software towards a scientific goal [36] but should not need to become fully trained computer scientists along the way [13].

The design of geospatial libraries requires trading off simplicity and flexibility [14]. Simple APIs let users have a clear comprehension of what is possible, or they can be flexible at the cost of being harder to master. For vernacular software developers, requiring mastery is a distraction from their primary task [34]. GDAL [15], a highly popular geospatial package, has huge flexibility to process geospatial data, at the cost of a large and verbose API that can be intimidating [40].

The high resolution of earth observation data also requires resource management, including how to work with raster data larger than available memory or how to parallelise batch jobs across heterogenous CPUs and GPUs. Most geospatial libraries shift the burden to the programmer to decide.

As time goes on, the core source code that implements a particular ecological method starts to become obfuscated from its original definition. Practical concerns such as loading data, managing resources, and distributing jobs all take over, inhibiting further iteration to the scientific work as the programmers need to map the implementation back to the method before they can resume.

To address these concerns, we have developed Yirgacheffe, an open-source declarative geospatial library that allows code to closely match the original methods, and automate resource scheduling. Yirgacheffe has the following goals:

Abstract geospatial datasets as an opaque type. When working with geospatial data, the dataset is usually loaded and then operations applied on a per-pixel basis. Yirgacheffe instead treats geospatial datasets as opaque types on which operations are performed without requiring further knowledge of how the data is stored.

Simplify geospatial operators. Geospatial data files can cover arbitrary areas of the planet, with polygons of varying resolutions. Yirgacheffe aligns the datasets and picks the right union or intersection operations depending on the calculation; something often described as confusing [40].

PROPL '25, Singapore, Singapore

ACM ISBN 979-8-4007-2161-8/25/10
https://doi.org/10.1145/3759536.3763806

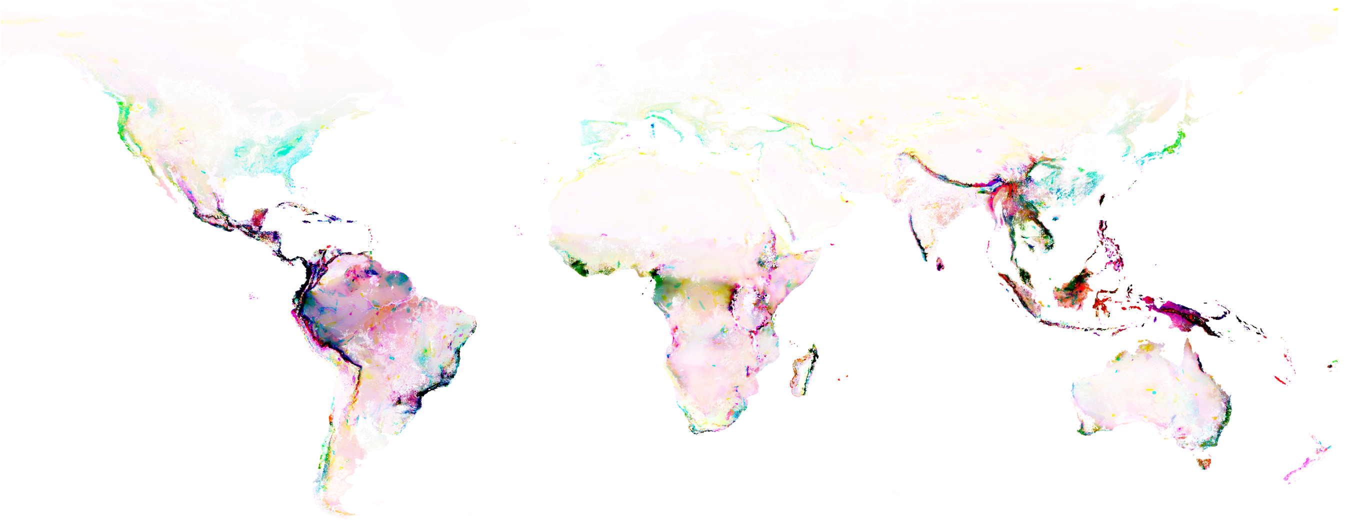

Figure 1. A false colour global LIFE map [12] showing the increased risk of species extinction if a 1.8km^2 cell of land is converted to arable. The full map takes a full day of computation on a modern AMD 128-core EPYC server.

Dynamically schedule resources. Geospatial raster layers used in detailed global analysis often require terabytes of RAM. As a rule of thumb, a global raster at 100m per pixel takes 150 GB per byte per pixel (a global map of `float32` data would be 600 GB). Therefore, for most real-world methods, the loading of entire rasters is infeasible.

There are three main Python geospatial libraries: GDAL, Rasterio, and Shapely. GDAL is the most flexible, working with both raster and vector data of many types. The API is comprehensive, but low-level. Users can manipulate byte arrays from within an image or rasterize polygons, and then use numpy to manipulate those data. GDAL does not provide resource management for memory or parallelisation, nor does it align datasets when reading data blocks. Rasterio provides a simpler API than GDAL which is more Pythonic in nature (GDAL being a SWIG interface over a C++ library); however, it too works at the level of providing access to array data read from areas of an image. Shapely is another Pythonic library, but only works on vector datasets. Neither provide a declarative interface to the data nor resource management.

There are also cloud-hosted services that come close to meeting our objectives, such as Microsoft Planetary Computer or Google Earth Engine (GEE) [16], which provide higher-level abstractions free from pixel-based and hardware concerns. The GEE APIs load both raster and geometry data as opaque objects, and automatically align and scale data to meet output requirements, but do not automatically clip to the minimal necessary work area as per our requirements. While GEE abstracts resource scheduling well, its resource acquisition requires using Google's proprietary hosted platform. This makes the GEE API unsuitable for open-source usage and a potentially bad choice for accessible and reproducible science. Federated alternatives are emerging, such as OpenEO [27] but still in their early stages.

For R, the Terra [19] library supplants the Raster [18] package. Terra shares design goals to ours and treats geospatial datasets abstractly to avoid loading them entirely into memory, but does not automatically align datasets or support parallelism/GPUs. It has some advantages over Yirgacheffe for direct pixel access for methods that require this.

2 Case Studies

Yirgacheffe was developed incrementally alongside the implementation of several large ecology pipelines, each processing large volumes of high-resolution raster data. This co-development, working closely with ecologists, provided valuable insight into the domain requirements.

The LIFE metric [12] examines the impact of land-use change on species extinction risk. LIFE considers 30k species from the IUCN Red List of endangered species [21], and for each analysis considers current, historic and a scenario specific species distributions generated at 100m per pixel from similarly high resolution habitat and elevation maps. An example output, for conversion of land to arable, is shown in Figure 1. An overview of the pipeline can be seen in Figure 2.

The IUCN's STAR metric [25] also uses the IUCN Red List and high-resolution raster data to assess the impact of different threat categories on species. We have taken this method and developed our own implementation using Yirgacheffe. Both LIFE and STAR make significant use of calculating species' Area Of Habitat (AOH), but with different species characteristics that presented different challenges (§2.1).

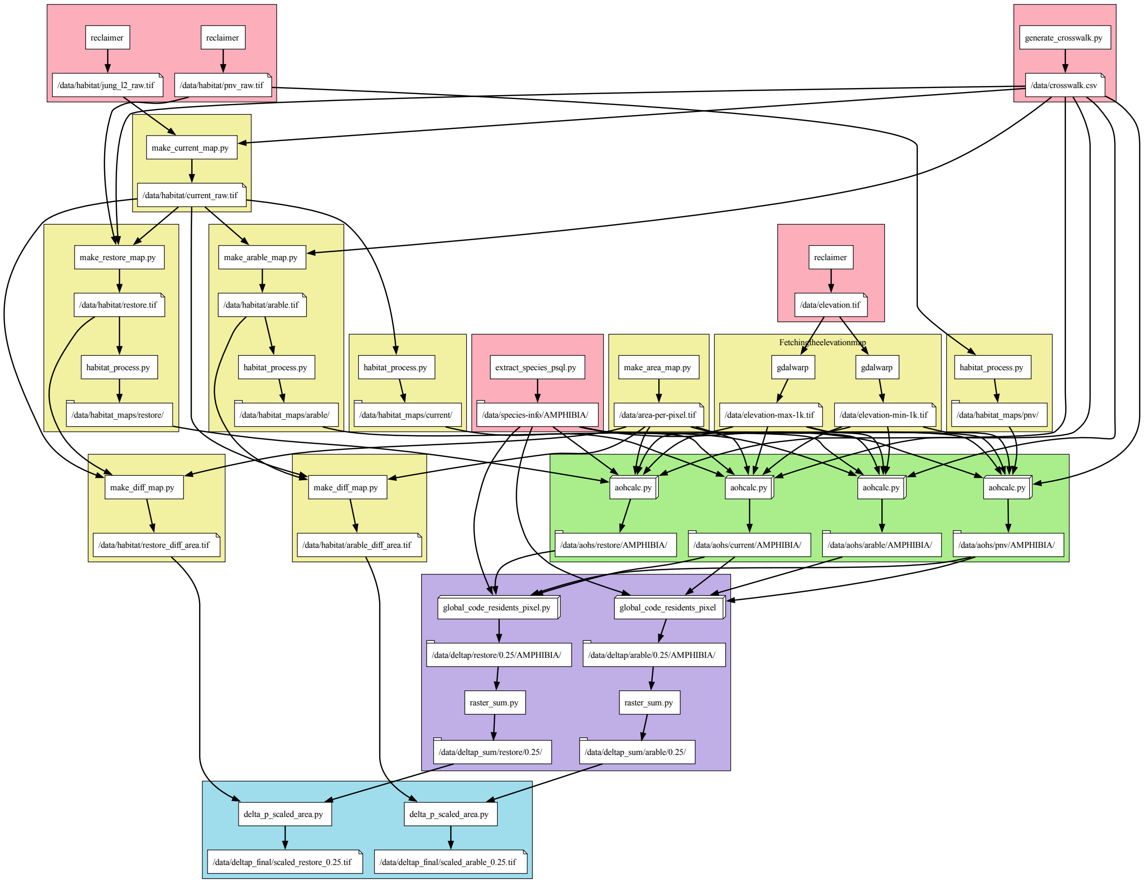
reclaimer
reclaimer
generate_crosswalk.py
/data/habitat/jung_l2_raw.tif
/data/habitat/pnv_raw.tif
/data/crosswalk.csv
make_current_map.py
/data/habitat/current_raw.tif
make_restore_map.py
make_arable_map.py
reclaimer
/data/habitat/restore.tif
/data/habitat/arable.tif
/data/elevation.tif
Fetchingtheelevationmap
habitat_process.py
habitat_process.py
habitat_process.py
extract_species_psql.py
make_area_map.py
gdalwarp
gdalwarp
habitat_process.py
/data/habitat_maps/restore/
/data/habitat_maps/arable/
/data/habitat_maps/current/
/data/species-info/AMPHIBIA/
/data/area-per-pixel.tif
/data/elevation-max-1k.tif
/data/elevation-min-1k.tif
/data/habitat_maps/pnv/
make_diff_map.py
make_diff_map.py
aohcalc.py
aohcalc.py
aohcalc.py
aohcalc.py
/data/habitat/restore_diff_area.tif
/data/habitat/arable_diff_area.tif
/data/aohs/restore/AMPHIBIA/
/data/aohs/current/AMPHIBIA/
/data/aohs/arable/AMPHIBIA/
/data/aohs/pnv/AMPHIBIA/
global_code_residents_pixel.py
global_code_residents_pixel
/data/deltap/restore/0.25/AMPHIBIA/
/data/deltap/arable/0.25/AMPHIBIA/
raster_sum.py
raster_sum.py
/data/deltap_sum/restore/0.25/
/data/deltap_sum/arable/0.25/
delta_p_scaled_area.py
delta_p_scaled_area.py
/data/deltap_final/scaled_restore_0.25.tif
/data/deltap_final/scaled_arable_0.25.tif

Figure 2. The LIFE pipeline flows topdown, with pink being data download, yellow data transforms, green AOH calculations, purple extinction risks, and blue human formatting. All tiers except the first use Yirgacheffe.

Yirgacheffe also underpins the PACT Tropical Moist Forest Accreditation Method [1, 2] (TMF), which calculates the change in sequestered carbon for avoided deforestation projects. This pipeline requires working with a wider range of input data formats, from GEDI point sampling data of forests from space [11], polygons of conservation projects and country borders, and high resolution (30m per pixel) land cover change rasters over many years. This wide variety of input styles, coupled with some of the statistical analysis in the pipeline, was useful in highlighting some limits of the approach Yirgacheffe takes.

2.1 Area Of Habitat

Both the STAR and LIFE metrics are centered on a species' "Area Of Habitat" (AOH) calculations, measured independently for thousands of species [5]. This algorithm takes a range polygon of where on the globe a species might be located, drawn by a human expert so as to minimize omission errors, and is combined with habitat preference and elevation preference maps to minimize commission errors:

$$AOH = R \cap H \cap E \qquad (1)$$

Where R is the species range, H is the location of suitable habitats, and E is location of suitable elevation. Those habitat and elevation preference maps are derived from high-resolution satellite data in combination with known species preference data around their choice of habitats and observed elevation occurrences:

$$\begin{aligned} AOH(s) = \{(x, y) \in R(s) : LC(x, y) \in H(s) \wedge \\ elev(x, y) \in [E_{min}(s), E_{max}(s)]\} \end{aligned} \qquad (2)$$

Whilst AOH is a relatively simple algorithm, the implementation gets more obfuscated when written to run at scale. A sequence of data intensive operations have to happen, starting with the conversion of data formats (range being polygonal data, and elevation and habitat maps being raster data), point data must be extracted, and then numerical methods applied to the results.

The Yirgacheffe based implementation of AOH is in Listing 1 and exemplifies our design goals: the geospatial datasets are manipulated directly, allowing the code to show the methodology more clearly; The layers are automatically aligned and intersected; and finally both the raster layers, hundreds of GB each, are chunked automatically, and the work distributed over multiple CPU cores.

```
species_info = json.load("info.json")
with yirgacheffe.read_raster("elevation.tif") as E:
    with yirgacheffe.read_reader("habitat.tif") as H:
        with yirgacheffe.read_shape("info.geojson") as R:
            Ef = (E > info.elevation_min) &
                 (E < info.elevation_max)
            Hf = H.is_in(info.habitats)
            AOH = R & Hf & Ef
            AOH.to_geotiff("aoh.tif", parallelism=True)
```

Listing 1. The Yirgacheffe implementation of AOH. Note that it treats the polygon range data and rasters equally, and the code matches closely the equation in Equation 2, other than the loading and saving of data.

Without Yirgacheffe this formula becomes unwieldy even using a library like numpy to avoid per-pixel operations alongside GDAL. An equivalent snippet to Listing 1 is shown in Appendix A and is much longer since it has to manually process the data, chunk it into schedulable units, and recombine everything to save the results. Files must be read incrementally requiring extra looping, offsets within files must be calculated manually, increasing the opportunity for errors. The `parallel_save` in the Yirgacheffe implementation also automatically takes advantage of multiple CPU cores, processing species on average in 32% the time of the GDAL version on the same inputs on an AMD EPYC 9534 CPU. Whilst a computer scientist could extract more substantial gains, Yirgacheffe does this without requiring users to be parallelism aware.

3 Yirgacheffe Implementation

Yirgacheffe implements a declarative interface to geospatial datasets using an API modeled on numerical libraries like Numpy [17], leveraging the prior experience of the target users. Internally it is built upon existing extensive and well tested libraries like numpy and GDAL, but providing a more opinionated and guided API to help both ease of use and enable automatic resource management.

In Yirgacheffe users create various "layers" that represent either raster data from GeoTIFFs [9], vector data from GeoJSON [6] or GeoPackage [32] datasets, H3 hex tiles [4], or constant values. There are also "group layers" that take individual layers and treat them like a single layer (e.g., useful for tiled raster data). Layers will not be loaded directly, but rather their contents are fetched lazily as required.

Users express numerical operations on and between layers directly to either calculate new layers or aggregations (sum, min, max, etc.), without having to refer to pixel data directly. These operations are expressed using built in language operators where possible, similar to numpy's approach for arrays, making working with large datasets as natural as working with scalar variables; this is in contrast to GEE's OO-based approach where operators are method calls on objects.

Operations supported by Yirgacheffe include conventional Python arithmetic and logic operations, common numpy array operations such as `where` and `isin`, and also includes an operator for 2D convolution matrix processing based on that of pytorch [33]. In the latter case Yirgacheffe will automatically take care of bounds adjustments to compensate for the kernel size.

Yirgacheffe will automatically align pixels within layers used in calculations based on their geospatial location rather than their location within each image array; appropriate empty values will be synthesized if necessary. If layers are not directly comparable, due to using a different map projection or a different pixel scale, Yirgacheffe deliberately does not attempt to insert automatic transforms, as what is correct when scaling and transforming layers depends on the problem being solved: for example, down-scaling a raster containing area values might use mean values, but for elevation maps min or max might be more appropriate. Thus the author of the method has to be involved in those decisions. Yirgacheffe does provide scaling operations, but these must be explicitly added by the user.

To allow arbitrary dimensions of the input data, Yirgacheffe has to align the various input layers for the calculations being done: whether the result be an intersection or a union of the area of the input layers depends on the types of calculation being performed. By default Yirgacheffe infers this using a set of first-order rules, applying either a union or an intersection of the data as appropriate for the underlying operation: e.g., for multiplication we will take an intersection, as data outside defined areas will default to zero; for addition it will take a union approach. These can be manually overridden if necessary, but having defaults that work for most common cases keeps the code closer to the method, and in our test cases it is rare that it has to be manually set.

As with loading data, Yirgacheffe evaluates calculations on layers lazily. When a user writes an expression over a set of layers in their Python code, the resultant variable value is not the answer, but rather an opaque type that contains the expression tree ready to be evaluated. The evaluation of this tree will only happen when the expression is either saved or aggregated. Metadata on the expression, such as the resultant geospatial dimensions, can be queried on the expression as if the layer had been calculated eagerly.

3.1 Resource Management

A key motivator for Yirgacheffe was to provide efficient use of memory for the large LIFE pipeline. While the average size of species ranges is small, a few larger species[1] caused the pipeline to run out of memory when parallel processing.

[1] For example, moose or bear species occupy the entire northern hemisphere

In most of these pipelines, the data is read only once in any given calculation; the later stages of the TMF that used random spatial sampling are the exception. This observation led us to effectively stream process expressions in Yirgacheffe: the result area is sliced, and only the parts of the rasters required for that slice are loaded, and any related polygon area is rasterized, minimizing the computation's memory footprint. The optimal size of that slice is a consideration: too small, and the overheads become significant, but we found in practice processing a few hundred rows of these large rasters lead to negligible performance loss versus loading everything into memory ahead of time. The caveat to that, observed in STAR which deals with marine birds, is species with complex range polygons that follow coast lines: excessive detail in these can slow down the slice rasterisation significantly.

The internal chunking of a calculation also provides another advantage: each of the slices being processed are independent of each other, and so this also provides a mechanism by which parallelism can be applied, albeit within the limited constraints of the Python run-time environment. The Python GIL [28] means that its runtime provides parallelism by using child processes rather than via shared-memory threads. This causes issues for libraries like GDAL, and thus in turn Yirgacheffe, as only primitive Python types can be passed between processes by the run-time, which would break the illusion Yirgacheffe makes of geospatial layers being similarly primitive. Yirgacheffe works around this by using a combination of shared memory for in flight computation data and carefully selecting layer metadata to allow GDAL objects to be closed and reopened on the other side.

3.2 CPU and GPU Support

Initially Yirgacheffe only used numpy for doing numerical processing. numpy is the standard numerical library for Python and is quick and expressive, and is a library a typical data scientist using Python is comfortable with. This allowed Yirgacheffe to develop incrementally along with the case study pipelines: we provide a `numpy_apply` operator on layers, which takes a function as an argument that is called back with data chunks in numpy format. We could therefore adopt Yirgacheffe for the handling of geospatial data before the declarative interface was fully developed.

However, escape hatches that expose internal workings come at a cost. One objective from the outset was to provide multiple backends to Yirgacheffe, supporting not just CPU compute, but also GPU, via CUPY [29] which abstracts the NVIDIA CUDA GPU framework, and MLX [26], which abstracts Apple's Metal GPU library. To do this Yirgacheffe's backend has separate interpreters over the user-specified operations to switch between numerical frameworks. However, any pipeline using the escape hatches in the API could not take advantage of multiple backends until Yirgacheffe supported a sufficient spread of numerical operators and the pipelines were migrated to use those natively.

Although our aim is to avoid having users deal with hardware decisions in their code, Yirgacheffe does not automatically select the backend, because moving a computation from CPU to GPU can have consequences on the precision of results, which means the same pipeline might generate different results on different hardware [39]. Unfortunately, we have observed this behaviour with standard CPU libraries also, due to the use of vector operations [30].

For a discrete GPU where system and graphics memory are not shared, there is overhead involved when moving computation to the GPU. We observed this while processing AOH across a large number of species; for species with a global reach (birds, or large mammals like bear or moose), using CUDA provided a performance benefit, but for the majority of land animals that have smaller ranges it was slower to do the work on the discrete GPU. A unified memory architecture overcomes this, however is only widely available on Apple hardware, but where possible the improvement is notable: calculating the 34821 AOHs for a single scenario in LIFE was approximately 1.6 times faster on an Apple M3 Ultra when using GPU vs CPU (for a discussion on why the gains are relatively modest see Section 4.2).

4 Discussion

Whilst Yirgacheffe has successfully demonstrated that it is possible to both provide APIs that support the task of geospatial programming whilst abstracting away computer science concerns, there were many problems we either didn't have time to solve, or need solved by some other solution over making Yirgacheffe more complicated.

4.1 Language Level

Some aspects of Yirgacheffe can be improved at the interface level of the library as it matures.

Alternative expressivity. Some computational ecology algorithms are not currently served by Yirgacheffe, as it currently focuses on the subset of geospatial problems that involve pixel based comparisons over large areas. In the TMF case study, the declarative interface of Yirgacheffe is ill suited to doing random sampling over geospatial areas for selecting the counter-factual pixels. Whilst from a code point of view an API like that found in Terra would make it possible to code up more cleanly, the similar chunking approach it takes to Yirgacheffe would not help here, and we need more support for different access patterns.

Partial results. Our simplified AOH algorithm (§2.1) diverges in LIFE and STAR as they use variations based on IUCN guidelines, which then results in more complexity in the implementation [8]. For example, the IUCN guidelines recommend if the filtered habitat and elevation layers are empty, then they are ignored for AOH. This means the layers are evaluated twice in the AOH code, once as part of testing for validity, and once again in the final calculation. Either a caching layer, or a logic that would work at a higher level in that simpler operators could potentially avoid this overhead. Note that this isn't the same as caching within a single expression (it can be seen in Listing 1 that the elevation layer is used twice on line 5), but rather this is needed across expressions also.

Managing side-effects. The ecology methods being implemented typically do not have side-effects within their methods descriptions. Yet when these are translated to imperative languages like Python, R or Julia, the code is littered with side effects in places that make it very hard to test the code. As a Python library, Yirgacheffe does nothing to address this, but discourages side-effecting code due to its declarative style. As a separate project we are porting Yirgacheffe to OCaml using modal types [23], parallelism [10, 37] and effects [38] to explore this further: can we build a functional, declarative DSL based on Yirgacheffe that promotes coding that follows more closely the written natural language method specifications that are found in the scientific literature?

Interactive programming. The emerging space of live programming environments [3, 31] is of huge interest to vernacular programmers such as our target audience, as it could make debugging and visualising intermediate results easier. Some of these also provide agentic support to sequence tasks in a user-friendly fashion [7], which would make assembling Yirgacheffe programs easier for non CS-experts.

Floating point precision. Excessive precision in data can lead to problems in both compute performance and storage (§3.1). Due to the limited precision of floating point numbers, we often see species ranges accurate to tens of km stored to the nearest nanometer. This lack of nuance in floating point representations can also lead people to infer more accuracy in the data than was originally intended. This technique has been applied in specific instances (e.g., for atmospheric modeling [22]), but a context-aware variable precision floating point type rather than IEEE floating point would make sense for most geospatial pipelines.

4.2 Hardware Level

Hardware has become heterogeneous in recent years, with multicore architectures and GPUs now widely available.

Axes of parallelism. When a method for the same algorithm needs to be applied to many items of data, there are two strategies: parallelise the algorithm and apply it in turn to each data item, or apply the algorithm to many data items in parallel. Yirgacheffe supports the former, but not the latter, and for some methodologies the latter strategy is more efficient. Whilst Yirgacheffe cannot automatically make this scheduling decision for the programmer, the lack of support for the second approach means the ecologist users must either manually implement Python multiprocessing or move to an external solution (e.g., GNU Parallel[2] or Littlejohn[3]).

Memory conservation. Yirgacheffe can manage the memory requirements of a single instance to ensure the machine is kept safe, but risks still exist that the machine can run out of memory. This leads at best to pipeline failure, or worse to unnoticed failures that generate incorrect results due to missing error handling in user code. Managing this is beyond the direct scope of Yirgacheffe, as it requires either better language support for parallelism, or ideally better cooperation from the operating system's scheduler [20]. Persisting the data to disk volumes on Docker [24] would also allow for easier debugging and sharing of intermediate results, but requires careful attention to space usage and snapshotting due to the huge amount of data involved in a typical pipeline.

Too many layers of abstraction: The layering of multiple numerical libraries has a performance cost. For example, we use the MLX library to provide Apple Metal support, which like Yirgacheffe provides a declarative interface for large opaque units of data (numpy style arrays). MLX lazily evaluates operations, allowing it to build a GPU kernel that covers as much of the operation as it can (e.g., to fit the entire AOH calculation into a single kernel). However, Yirgacheffe similarly builds up a lazy expression tree and evaluates it in a depth-first fashion, meaning when using it as a backend, MLX never gets to see the full operation and can only build kernels for single operators. This we believe is why we only see the modest performance gains when using a GPU outlined previously, and it is an area we intend to address in future releases. MLX does not currently provide a programmatic interface to building expressions, but we could compile Yirgacheffe expressions to Python code for MLX dynamically, allowing it then to do its compilation with a full view of the expression whilst leaving Yirgacheffe to manage the input data alignment and chunking.

5 Conclusions

Yirgacheffe is our declarative geospatial library that empowers ecologists to solve data science problems more clearly and concisely by handling both the geospatial and resource scheduling required in this domain. It has been used to build pipelines for calculating several global published conservation metrics that process petabytes of raster data.

However, many challenges still remain for the programming language and systems research communities, such as balancing expressivity with incremental results and more flexibility for heterogenous hardware. We welcome contributions at https://github.com/quantifyearth/yirgacheffe.

[2] See https://www.gnu.org/software/parallel/

[3] See https://github.com/quantifyearth/littlejohn

A Supplementary Source Code

The code below is the logic in Listing 1 without Yirgacheffe.

```
species_info = json.load("info.json")
elevation = gdal.Open("elevation.tif")
habitat = gdal.Open("habitat.tif")
range_polygon = ogr.Open("info.geojson")
habitat_left, habitat_xstep, _, habitat_top, _, habitat_ystep =
      habitat.GetGeoTransform()
elevation_left, elevation_xstep, _, elevation_top, _,
      elevation_ystep = elevation.GetGeoTransform()
layer = range_polygon.GetLayer()
envelopes = []

layer.ResetReading()
feature = layer.GetNextFeature()
while feature:
    geometry = feature.GetGeometryRef()
    if geometry:
        envelopes.append(geometry.GetEnvelope())
    feature = layer.GetNextFeature()
if len(envelopes) == 0:
    raise ValueError('No geometry found')

abs_xstep, abs_ystep = abs(habitat_xstep), abs(habitat_ystep)
range_origin_x = floor(min(x[0] for x in envelopes) / abs_xstep)
range_origin_y = ceil(max(x[3] for x in envelopes) / abs_ystep)
range_left = range_origin_x * abs_xstep
range_top = range_origin_y * abs_ystep
range_right = ceil(max(x[1] for x in envelopes) / abs_xstep) *
      abs_xstep
range_bottom = floor(min(x[2] for x in envelopes) / abs_ystep)
      * abs_ystep
range_width = round((range_right - range_left) / abs_xstep)
range_height = round((range_top - range_bottom) / abs_ystep)

result = gdal.GetDriverByName("GTiff").Create(
    output_path, range_width,
    range_height, 1,
    gdal.GDT_Byte, ['COMPRESS=LZW','BIGTIFF=YES'])

YSTEP = 512
for yoffset in range(0, range_height, YSTEP):
    ystep = YSTEP if (yoffset + YSTEP) < range_height else
          (range_height - yoffset)
    dataset = gdal.GetDriverByName('mem').Create('mem',
          range_width, ystep, 1, gdal.GDT_Byte, [])
    dataset.SetProjection(habitat.GetProjection())
    dataset.SetGeoTransform([
        range_left, habitat_xstep,
        0.0, range_top + (yoffset * habitat_ystep),
        0.0, habitat_ystep
    ])
    gdal.RasterizeLayer(dataset, [1], layer, burn_values=[1],
          options=["ALL_TOUCHED=TRUE"])
    range_data = dataset.GetRasterBand(1).ReadAsArray(0, 0,
          range_width, ystep)

    habitat_data = habitat.GetRasterBand(1).ReadAsArray(
        round((range_left - habitat_left) / habitat_xstep),
        round((range_top - habitat_top) / habitat_ystep) +
              yoffset,
        range_width, ystep )
    filtered_habitat_data = np.isin(habitat_data, info.habitats)
    elevation_data = elevation.GetRasterBand(1).ReadAsArray(
        round((range_left - elevation_left) / habitat_xstep),
        round((range_top - elevation_top) / habitat_ystep) +
              yoffset,
        range_width, ystep )
    filtered_elevation_data = (elevation_data >
          info.elevation_min) & (elevation_data <
          info.elevation_max)
    aoh = filtered_habitat_data * filtered_elevation_data *
          range_data
    result.GetRasterBand(1).WriteArray(aoh, 0, yoffset)
result.Close()
```

References

[1] Balmford, A., Coomes, D., Dales, M., Ferris, P., Hartup, J., Jaffer, S., Keshav, S., Lam, M., Madhavapeddy, A., Message, R., Rau, E.-P., Swinfield, T., and Wheeler, C. PACT tropical moist forest accreditation methodology. Tech. rep., University of Cambridge, 2023.

[2] Balmford, A., Keshav, S., Venmans, F., Coomes, D. A., Groom, B., Madhavapeddy, A., and Swinfield, T. Realizing the social value of impermanent carbon credits. *Nature Climate Change 13*, 11 (nov 2023), 1172–1178.

[3] Blinn, A., Li, X., Kim, J. H., and Omar, C. Statically contextualizing large language models with typed holes. *Proc. ACM Program. Lang. 8*, OOPSLA2 (Oct. 2024).

[4] Brodsky, I. H3: Uber's hexagonal hierarchical spatial index. https://www.uber.com/en-GB/blog/h3/, 2018.

[5] Brooks, T. M., Pimm, S. L., R., A. H., Buchanan, G. M., Butchart, S. H. M., Foden, W., Hilton-Taylor, C., Hoffmann, M., Jenkins, C. N., Joppa, L., Li, B. V., Menon, V., Ocampo-Peñuela, N., and Rondinini, C. Measuring terrestrial area of habitat (AOH) and its utility for the IUCN red list. *Trends in Ecology & Evolution* (2019).

[6] Butler, H., Daly, M., Doyle, A., Gillies, S., Schaub, T., and Hagen, S. The GeoJSON format. RFC 7946, Aug. 2016.

[7] Croisdale, G., Huang, E., Chung, J. J. Y., Guo, A., Wang, X., Henley, A. Z., and Omar, C. Deckflow: Iterative specification on a multimodal generative canvas, 2025.

[8] Dales, M. Area of habitat calculation code for biodiversity assessment pipelines. https://github.com/quantifyearth/aoh-calculator, 2023.

[9] Devys, E., Habermann, T., Heazel, C., Lott, R., and Rouault, E. OGC GeoTIFF standard. Tech. Rep. 19-008r4, Open Geospatial Consortium, Sept. 2019.

[10] Dolan, S., White, L., and Madhavapeddy, A. Multicore OCaml. In *the 4th ACM OCaml Users and Developers Workshop* (sep 2014).

[11] Dubayah, R., Armston, J., Healey, S. P., Bruening, J. M., Patterson, P. L., Kellner, J. R., Duncanson, L., Saarela, S., Ståhl, G., Yang, Z., et al. GEDI launches a new era of biomass inference from space. *Environmental Research Letters 17*, 9 (2022), 095001.

[12] Eyres, A., Ball, T. S., Dales, M., Swinfield, T., Arnell, A., Baisero, D., Durán, A. P., Green, J. M. H., Green, R. E., Madhavapeddy, A., and Balford, A. LIFE: A metric for mapping the impact of land-cover change on global extinctions. *Philosophical Transactions of The Royal Society B* (2025).

[13] Ferris, P., Dales, M., Swinfield, T., Jaffer, S., Keshav, S., and Madhavapeddy, A. Uncertainty at scale: how CS hinders climate research. *Undone Computer Science* (2024).

[14] Gabriel, R. The Rise of "Worse is Better". https://dreamsongs.com/RiseOfWorseIsBetter.html, 1991.

[15] GDAL/OGR contributors. *GDAL/OGR Geospatial Data Abstraction Software Library*. Open Source Geospatial Foundation, 2024.

[16] Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., and Moore, R. Google Earth Engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment 202* (2017), 18–27.

[17] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature 585*, 7825 (Sept. 2020), 357–362.

[18] Hijmans, R. J. *raster: Geographic Data Analysis and Modeling*, 2025. R package version 3.6-32.

[19] Hijmans, R. J. *terra: Spatial Data Analysis*, 2025. R package version 1.8-57.

[20] Humphries, J. T., Natu, N., Chaugule, A., Weisse, O., Rhoden, B., Don, J., Rizzo, L., Rombakh, O., Turner, P., and Kozyrakis, C. ghOSt:

Fast & flexible user-space delegation of Linux scheduling. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (New York, NY, USA, 2021), SOSP '21, Association for Computing Machinery, p. 588–604.

[21] IUCN. The IUCN red list of threatened species. version 2023-1. https://www.iucnredlist.org, 2023.

[22] Klöwer, M., Razinger, M., Dominguez, J. J., Düben, P. D., and Palmer, T. N. Compressing atmospheric data into its real information content. *Nature Computational Science* (2021), 713–724.

[23] Lorenzen, A., White, L., Dolan, S., Eisenberg, R. A., and Lindley, S. Oxidizing OCaml with modal memory management. *Proc. ACM Program. Lang. 8*, ICFP (Aug. 2024).

[24] Madhavapeddy, A., Scott, D. J., Ferris, P., Gibb, R. T., and Gazagnaire, T. Functional networking for millions of Docker desktops (experience report). *Proceedings of ACM Programming Languages 9*, ICFP (aug 2025), 256:597–256:615.

[25] Mair, L., Bennun, L., Brooks, T., Butchart, S., Bolam, F., Burgess, N., Ekstrom, J., Milner-Gulland, E., Hoffmann, M., Ma, K., Macfarlane, N., Raimondo, D., Rodrigues, A., Shen, X., Strassburg, B., Beatty, C., Gómez-Creutzberg, C., Iribarrem, A., Irmadhiany, M., Lacerda, E., Mattos, B., Parakkasi, K., Tognelli, M., Bennett, E., Bryan, C., Carbone, G., Chaudhary, A., Eiselin, M., da Fonseca, G., Galt, R., Geschke, A., Glew, L., Goedicke, R., Green, J., Gregory, R., Hill, S., Hole, D., Hughes, J., Hutton, J., Keijzer, M., Navarro, L., Nic Lughadha, E., Plumptre, A., Puydarrieux, P., Possingham, H., Rankovic, A., Regan, E., Rondinini, C., Schneck, J., Siikamäki, J., Sendashonga, C., Seutin, G., Sinclair, S., Skowno, A., Soto-Navarro, C., Stuart, S., Temple, H., Vallier, A., Verones, F., Viana, L., Watson, J., Bezeng, S., Böhm, M., Burfield, I., Clausnitzer, V., Clubbe, C., Cox, N., Freyhof, J., Gerber, L., Hilton-Taylor, C., Jenkins, R., Joolia, A., Joppa, L., Koh, L., Lacher, T., Langhammer, P., Long, B., Mallon, D., Pacifici, M., Polidoro, B., Pollock, C., Rivers, M., Roach, N., Rodríguez, J., Smart, J., Young, B., Hawkins, F., and McGowan, P. A metric for spatially explicit contributions to science-based species targets. *Nature Ecology and Evolution 5*, 6 (June 2021), 836–844.

[26] MLX contributors. MLX: An array framework for Apple silicon. https://github.com/ml-explore/mlx.

[27] Mohr, M., Pebesma, E., Dries, J., Lippens, S., Janssen, B., Thiex, D., Milcinski, G., Schumacher, B., Briese, C., Claus, M., Jacob, A., Sacramento, P., and Griffiths, P. Federated and reusable processing of earth observation data. *Scientific Data 12*, 1 (Feb. 2025).

[28] Muttin, S., and Dash, D. Threading and multiprocessing module and the limitations due to the GIL in Python. *International Journal of Scientific Research and Engineering Development* (2021).

[29] Nishino, R., and Loomis, S. H. C. CuPy: A NumPy-compatible library for NVIDIA GPU calculations. *31st Conference on Neural Information Processing Systems 151*, 7 (2017).

[30] BUG: Power calculation rounding error for array values on AMD EPYC 9534. https://github.com/numpy/numpy/issues/25269.

[31] Omar, C., Voysey, I., Chugh, R., and Hammer, M. A. Live functional programming with typed holes. *Proc. ACM Program. Lang. 3*, POPL (Jan. 2019).

[32] Open Geospatial Consortium. OGC GeoPackage encoding standard. http://www.opengis.net/doc/IS/geopackage/1.4, 2024.

[33] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems* (2019).

[34] Pertseva, E., Chang, M., Zaman, U., and Coblenz, M. A theory of scientific programming efficacy. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (New York, NY, USA, 2024), ICSE '24, Association for Computing Machinery.

[35] Reynolds, S., Beery, S., Burgess, N., Burgman, M., Butchart, S., Cooke, S. J., Coomes, D. A., Danielsen, F., Minin, E. D., Durán, A. P., Gassert, F., Hinsley, A., Jaffer, S., Jones, J. P., Li, B. V., Aodha, O. M., Madhavapeddy, A., O'Donnell, S., Oxbury, B., Peck, L., Pettorelli, N., Rodríguez, J. P., Shuckburgh, E., Strassburg, B., Yamashita, H., Miao, Z., and Sutherland, B. The potential for AI to revolutionize conservation: a horizon scan. *Trends in Ecology & Evolution* (dec 2024), S0169534724002866.

[36] Shaw, M. Myths and mythconceptions: what does it mean to be a programming language, anyhow? *Proceedings of the ACM on Programming Languages 4*, 234 (2020), 1–44.

[37] Sivaramakrishnan, K., Dolan, S., White, L., Jaffer, S., Kelly, T., Sahoo, A., Parimala, S., Dhiman, A., and Madhavapeddy, A. Retrofitting parallelism onto OCaml. *Proceedings of the ACM on Programming Languages 4*, ICFP (aug 2020), 1–30.

[38] Sivaramakrishnan, K., Dolan, S., White, L., Kelly, T., Jaffer, S., and Madhavapeddy, A. Retrofitting effect handlers onto OCaml. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (jun 2021), ACM, pp. 206–221.

[39] Whitehead, N., and Fit-Florea, A. Precision and Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs. Tech. rep., NVIDIA, 2011.

[40] Ziegler, P., and Chasins, S. E. A need-finding study with users of geospatial data. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2023), CHI '23, Association for Computing Machinery.

Received 2025-07-07; accepted 2025-08-11