IOOS GSoC Project Update Meeting #1

Friday June 26, 2026

Project #1: Enhancing noaa_coops

- noaa_coops is a Python client library used to programmatically access ocean data from the NOAA monitoring stations along the US coast and Great Lakes.

- The package was functional but had **gaps**:
  - **large date range requests** failed silently,
  - product types were **inaccessible**,
  - and station metadata attributes were being **lost** due to **incorrect branch routing**.

- This project focuses on closing those gaps to make noaa_coops reliable enough for real research workflows.

Project #1: Enhancing noaa_coops

Deliverables completed before midterm:

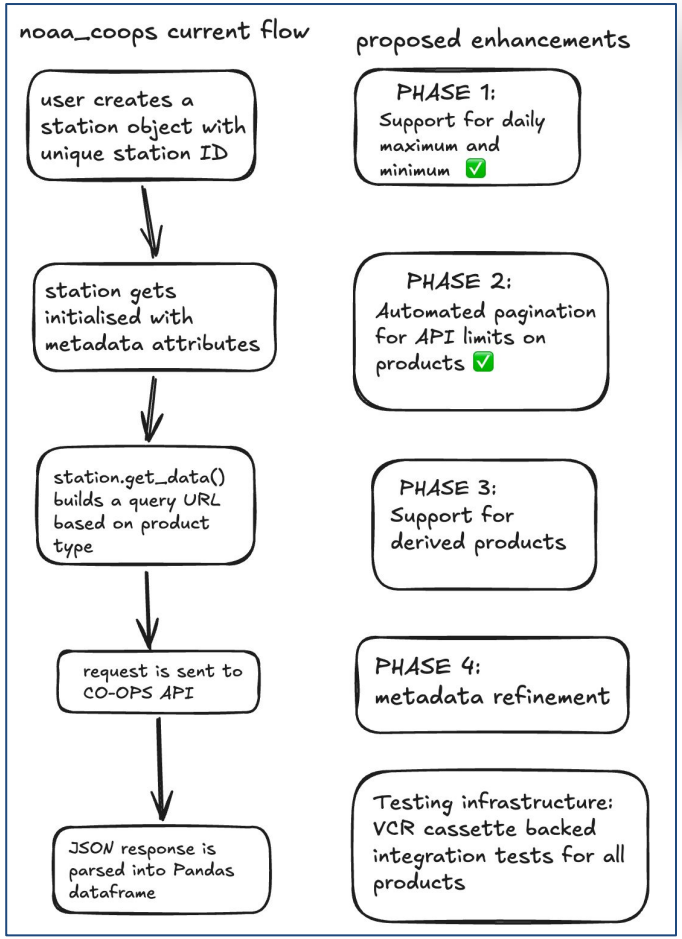
- **OFS water level:** Added validation to prevent "400 Bad Request" errors from missing datums.
- **Daily maximum and minimum**: Built JSON flattening logic to normalize nested API responses.
- **Pagination**: Implemented product-limit routing for multi-block requests.

In Progress:

- **VCR Cassette Tests**: Built an offline integration suite to test network behaviors without hitting live servers.

Planned work post midterm:

- **Derived Product Client**: Integrate a new dedicated client for NOAA's Derived Data Products API.
- **Metadata Refinement**: Overhaul the architecture to parse raw JSON into accessible Python attributes; no loss of any attributes due to exclusive branching.

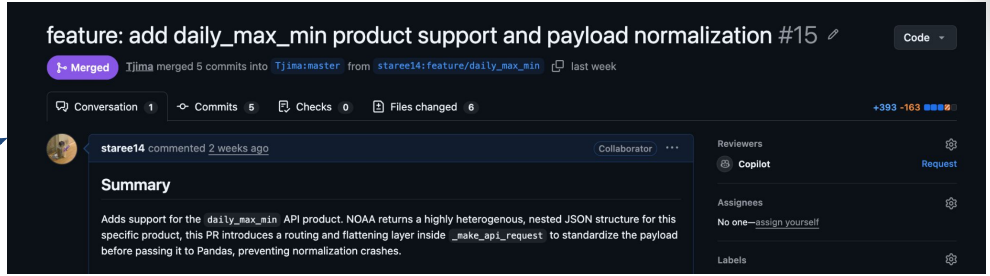


Project #1: Enhancing noaa_coops

Phase 1: Support for daily_max_min

Status: Merged on June 15th

- daily_max_min product was missing.
- NOAA returns a heterogeneous nested JSON structure with two possible timestamp keys depending on interval.
- Built a routing and flattening layer inside _make_api_request() to detect the correct key and normalize the payload before pandas parsing.
- **Result**: uniform DataFrame output regardless of interval type.




```python
# testing daily max min
from noaa_coops import Station
p= Station("9491094") # water levels station
df = p.get_data(
    begin_date="20170101",
    end_date="20170102",
    datum='STND',
    product="daily_max_min",
    interval=6,
    units="english",
    time_zone="gmt",
)
print(df)
```

[3] ✓ 1.7s Python

	record_type	value	pcComplete	flag
2017-01-01 09:42:00	max	9.951	100	0
2017-01-02 22:24:00	max	8.488	100	0
2017-01-01 23:54:00	min	8.425	100	0
2017-01-02 17:24:00	min	7.198	100	0

After flattening: clean DataFrame with normalized columns, same interface as every other product


```
{
  "metadata": {
    "id": "9491094",
    "name": "Red Dog Dock",
    "lat": "67.5758",
    "lon": "-164.0644"
  },
  "data": [
    {
      "dailyMin6Min": [
        {
          "date6Min": "2017-01-01",
          "time6Min": "23:54",
          "value6Min": 8.425,
          "pcCompleteMin": 100,
          "flag6Min": "0"
        },
        {
          "date6Min": "2017-01-02",
          "time6Min": "17:24",
          "value6Min": 7.198,
          "pcCompleteMin": 100,
          "flag6Min": "0"
        }
      ]
    },
    {
      "dailyMinHourly": [
        {
          "dateHourly": "2017-01-01",
          "timeHourly": "19:00",
          "valueHourly": 8.606,
          "pcCompleteHourly": 100,
          "flagHourly": "0"
        },
        {
          "dateHourly": "2017-01-02",
          "timeHourly": "17:00",
          "valueHourly": 7.277,
          "pcCompleteHourly": 100,
          "flagHourly": "0"
        }
      ]
    }
  ]
}
```

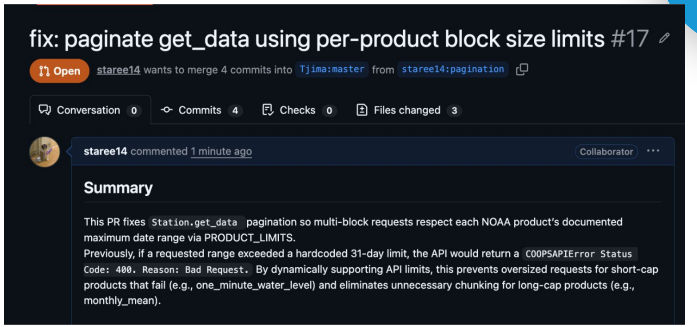

Raw JSON response from NOAA.

Data nested under `dailyMin6Min` and `dailyMinHourly` with non-standard timestamp keys

Project #1: Enhancing noaa_coops

Phase 2: Automated Data Iteration

- Previously, hardcoded block sizes caused requests exceeding API limits to fail.
- Implemented `PRODUCT_LIMITS` : a per-product dictionary mapping each product to its documented maximum date range



Before: `one_minute_water_level` has a 4-day limit — a 6-day request hits the API limit and raises a **`COOPSAPIError`**

```
# one_minute_water_level (block_size=4 days)
from noaa_coops import Station
p = Station("9447130")
df = p.get_data(
    begin_date="20150101",
    end_date="20150106",   # 6 days → 2 blocks of 4
    product="one_minute_water_level",
    datum="MLLW",
    units="metric",
)
print(df)
```

1.5s Python

```
valid.  Range Limit Exceeded: The size limit for data retrieval for this product is 4 days "}}
```

```
# one_minute_water_level (block_size=4 days)
from noaa_coops import Station
p = Station("9447130")
df = p.get_data(
    begin_date="20150101",
    end_date="20150106",   # 6 days → 2 blocks of 4
    product="one_minute_water_level",
    datum="MLLW",
    units="metric",
)
print(df)
```

[2] 2.6s

```
                     v
t
2015-01-01 00:00:00  1.792
2015-01-01 00:01:00  1.778
2015-01-01 00:02:00  1.764
2015-01-01 00:03:00  1.746
2015-01-01 00:04:00  1.736
...                    ...
2015-01-05 23:56:00  3.286
2015-01-05 23:57:00  3.290
2015-01-05 23:58:00  3.290
2015-01-05 23:59:00  3.284
2015-01-06 00:00:00  3.288

[7201 rows x 1 columns]
```

After: Same request automatically splits into 2 blocks of 4 days, **concatenated** into one DataFrame, 7201 rows returned.

Project #2: Filling the gaps for ocean glider data quality control

Stage 1: Bonding period

- Broke down the codebase history for an idea of the project context
  - 4 distinct “development periods” of different contributors
- Meeting the community
  - UG2 conference in Florida
  - 2 additional Zoom calls to hear individual use cases
  - Lab tour and data discussion in California
- Tested the toolbox, went over coding practices, set up mentor comms
- Submitted all my findings to act as a notebook before making PRs

Stage 2: Group 1 Required Tests

- Compared what the manual says it does, vs what the code says it does
- Engaged with community for missing `timing/gap` and `syntax` tests
- Individual PRs for new tests, including a template for documentation

Challenges: Working on it every day, finding exemplar test data

Underwater Gliders 2 (UG2) conference in May

Project #3: Integrating Empirical Dynamic Modeling (EDM) into FIMS

Project Overview and Road Map

Phase 0: Codebase Exploration (Community bonding and Pre-coding phase)
Analyzed the FIMS architecture and added TODO(EDM) integration points to guide future implementation

Phase 1: (Completed)
Delay Embedding Infrastructure ⟹ Completed and Merged ✅

Phase 2: (In Progress)
Prediction Frameworks ⟹ In Progress, PR #1561 is open for this
(Simplex, S-map, GP-EDM)

Phase 3: (Upcoming)
Statistical Fitting within FIMS

Phase 4: (Upcoming after mid term evaluation)
Core EDM infrastructure complete and EDM model Families

Project #3: Integrating Empirical Dynamic Modeling (EDM) into FIMS

Phase 1: Delay Embedding Infrastructure

Implemented

- Generic DelayEmbeddingMatrix
- Configurable embedding dimension (E)
- Configurable lag (τ)
- Missing-value handling
- Uncertainty propagation
- Rcpp Interface
- FIMSFrame Integration
- GoogleTests + testthat

For example: Time Series: [1 2 3 4 5 6 7 8]
Using:

- Embedding Dimension (E) = 3
- Time Lag (τ) = 1

Embedded State: [1 2 3] Predicting: 4
[2 3 4] 5
[3 4 5] ... 6...

Project #3: Integrating Empirical Dynamic Modeling (EDM) into FIMS

Phase 2: Prediction Framework

Completed

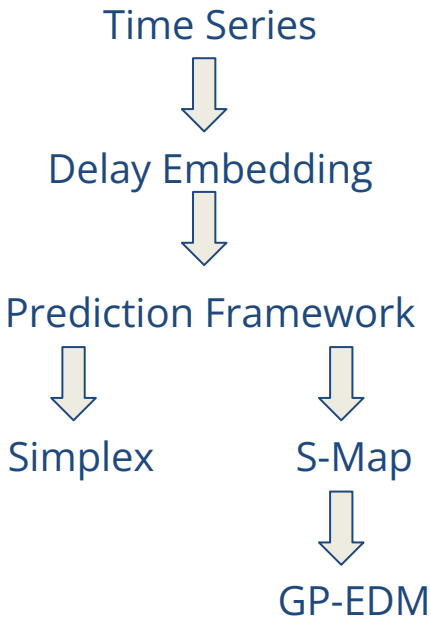
- Generic Prediction Framework
- Modular Distance Weighting Infrastructure
- Simplex Projection
- Standard S-Map (in progress)

Next Steps

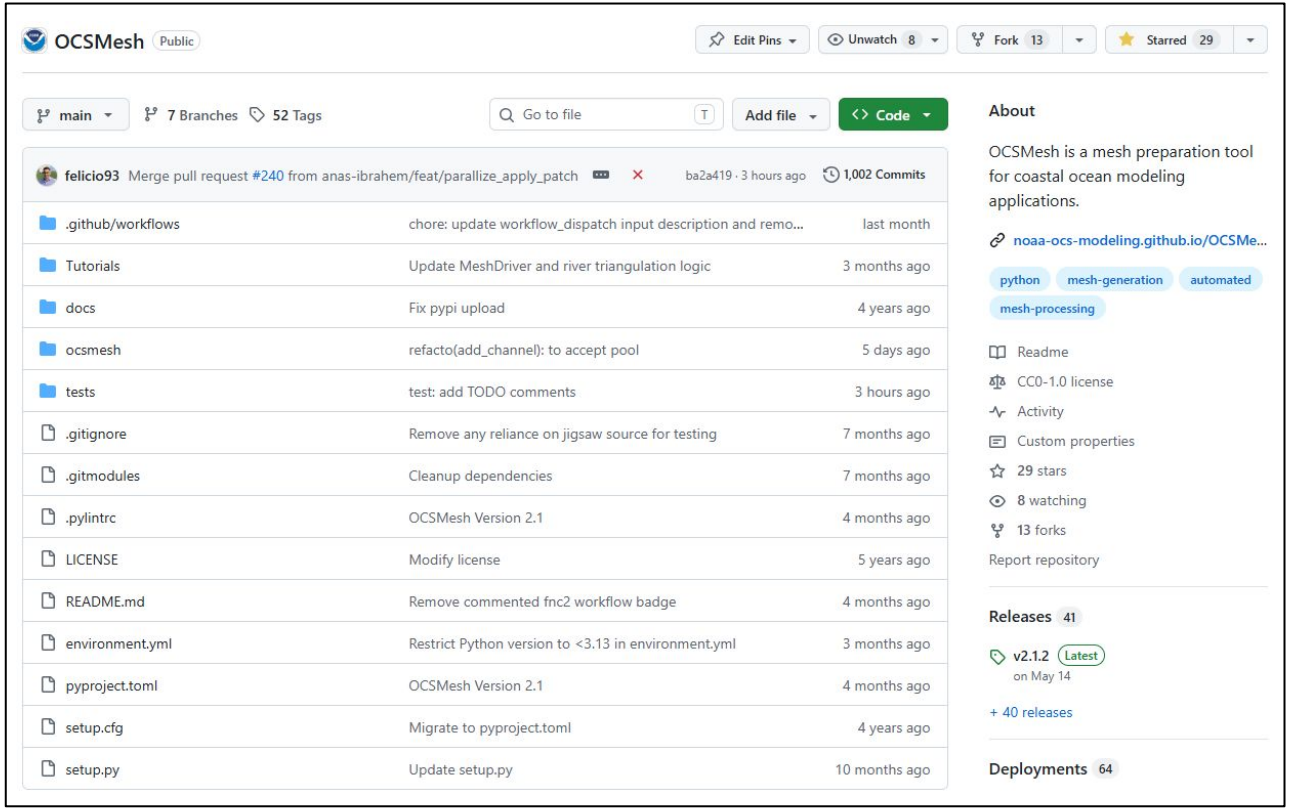
- Complete S-Map implementation
- Implement GP-EDM
- Validate predictions against rEDM
- Respective GoogleTests and testthat

Challenges faced:
Maintaining compatibility with both C++ (GoogleTest) and R (Rcpp/testthat) interfaces.

Prediction Pipeline



Project #4: Scalable OCSMesh - Parallelization and Spatial Partitioning



Phase 1 - Due July 1st

- 1.1 - First PR:
  - Parallelize apply_contraints()

Merged on Jun 5th

- 1.2 - Cascading Parallelism:
  - Parallelize apply_patch()
  - Parallelize apply_channel()

Merged on Jun 23th

- 1.3 Two-State Parallelization:
  - Parallelize write_hfun_to_disk
  - Profile the code and identify remaining bottlenecks

Under Development - On Time

- Nice to have:
  - Drop support to Python 3.9 and add support to 3.13, 3.14(x)
  - Create better tests for hfun

Not Started - On Time

Phase 2 - Due Aug 1st

- 2.0 - Create IOOS Sandbox (MPI Instance) — **Under Development - Due Jul 1st**
- 2.1 - Initialization and Fixes:
  - Create test_del_mpi.py to reproduce failure under mpirun -n 2, then apply/prove the fix — **Not Started - Due Aug 1st**
- 2.2 Initial MPI Implementation:
  - Implement MPI parallelization for one method (e.g., _apply_flow_limiters_mpi) (rank detection, scatter/gather)
  - Roll out MPI dispatch pattern for all remaining methods — **Not Started - Due Aug 1st**
- 2.3 MPI Validation:
  - Develop a full test suite for the MPI implementation. — **Not Started - Due Aug 1st**

Phase 3 - Due Sep TBD

- 3.1 - Partitioning Generator:
  - Find a way to efficiently split the mesh domain (e.g., regular tiles, node-count-based partitions, hfun-based clusters, etc.)

  Not Started - Due Sep
- 3.2 Partitioning-based input reading

  Details TBD - Due Sep
- 3.3 Partitioning-based assembly and mesh mesh gen

  Details TBD - Due Sep
- Full Validation

  Details TBD - Due Sep

Project #5: Enhancing CrocoLakeTools with IOOS Data Sync from ERDDAP

The problem:

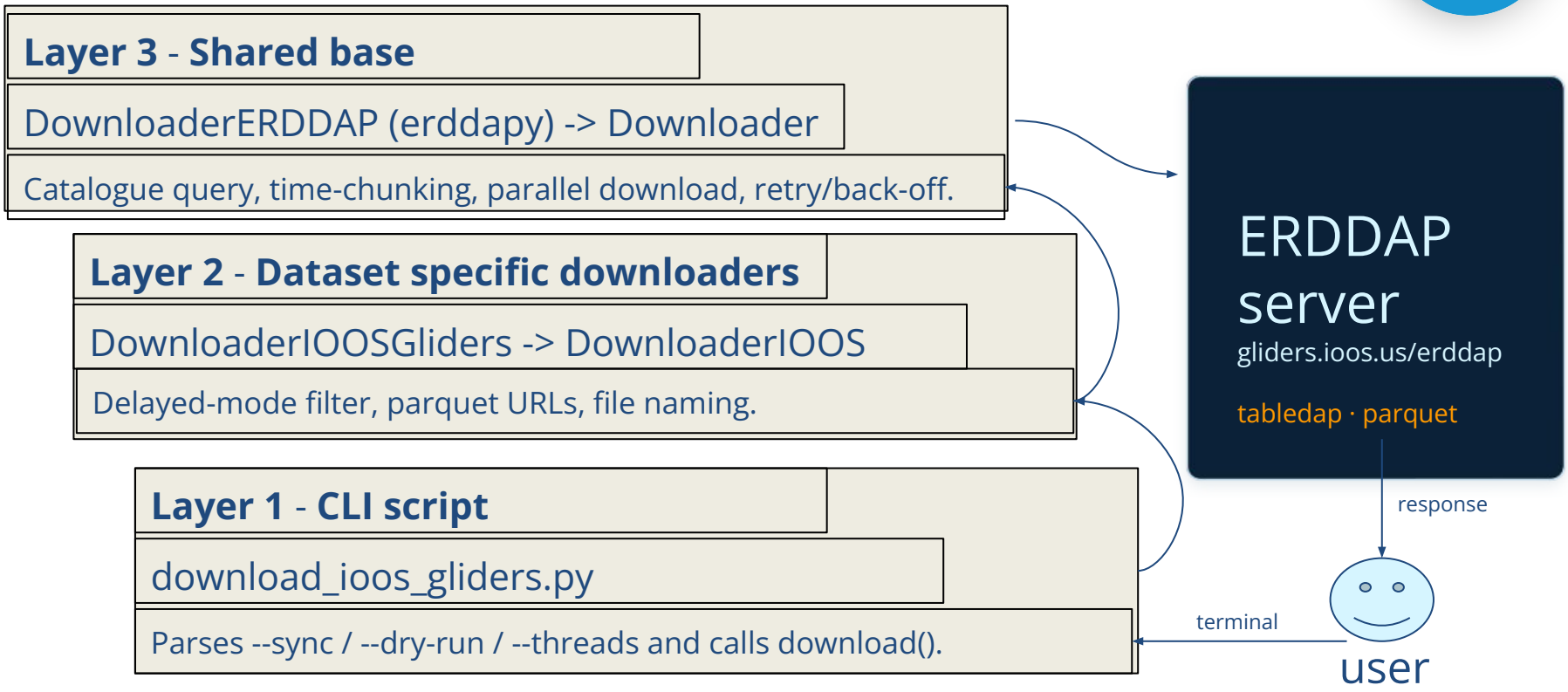
- ❏ Static downloaders can't track a dynamic server.
- ❏ gliders.ioos.us/erddap hosts thousands of glider data deployments. Existing datasets get updated time to time.
- ❏ We need to have a syncing mechanism along with the basic download option.

ON THE LIVE SERVER

~2000+ datasets

a hardcoded list is not a scalable solution.

Solution: A three-layer, reusable ERDDAP constrained architecture (Project #5)



Tackled Challenges (Project #5)

ERDDAP caps response size

A whole deployment in one request is rejected, in practice only payloads up to ~35 MB come back cleanly.

solution

Downloading datasets in chunks

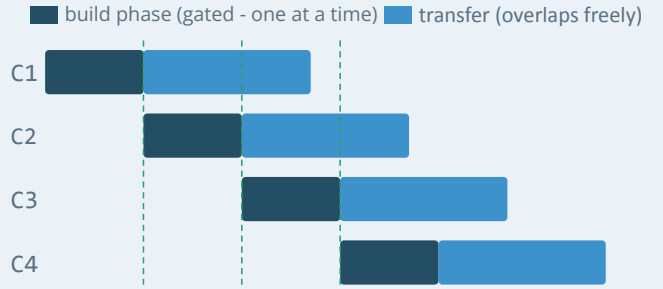
Download 24-hour windows, then merge locally into one parquet. If a window still returns 413(large payload), step the schedule down: 24h -> 12h -> 6h.

Naive parallelism makes the server go down

Even on 4 request in parallel, memory blows past the limit, the server returns 503 and during testing it went fully unresponsive.

solution

Gated parallelism, overlapping transfers



each dashed line = first byte received -> gate opens -> next build starts

How we are handling download errors (Project #5)

Signal	What it really means	How the downloader handles it
`HTTP 413`	Single response too large	Re-chunk smaller on the 24h -> 12h -> 6h schedule
`HTTP 503`	Server out of memory / overloaded	Back off and retry - 15s, 30s, 60s
`HTTP 429`	Too many requests per unit time	Same retry-with-back-off path
`ReadTimeout`	Build slower than the client timeout	Read timeout raised to 600s, then retry
`HTTP 404`	Empty time window - no data	Not considered an error: skip the window, keep going

Progress so far (Project #5)

- During community bonding period, had multiple meetings and discussions with enrico for requirement analysis and performed erddap server interaction experiments.
- Performed base downloader class refactoring along with oleanderXBT downloader.
  - https://github.com/boom-lab/crocolaketools/pull/48 (Merged)
- Opened issue (https://github.com/ioos/erddapy/issues/467) to add `parquet` as supported format in erddapy that was resolved on (https://github.com/ioos/erddapy/pull/46) by Filipe.
- Implemented the IOOS Downloader with erddap base class handler and sync feature.
  - https://github.com/boom-lab/crocolaketools/pull/50 (Review in progress).
  - Enrico tested the PR in his local, it is working as expected. Right now working on some minor refactoring suggested by Enrico.
  - Working on suitable variable selection, there are around 3500 unique variables across the datasets. Not every variable is needed for crocolake for now.
- Mitigation of Crocolakeloader dependency from crocolaketools.
  - https://github.com/boom-lab/crocolaketools/pull/53 (Review in Progress)
  - https://github.com/boom-lab/crocolakeloader/pull/10 (Merged)
- Created a CI pipeline for syncing Crocolaketools and CrocolakeLoader. (Under development)
- Will start with the Animal Telemetry stretch goal, once the existing PR pass through the review process and gets merged .

Project #6: Add sex structure to FIMS statistical catch-at-age model

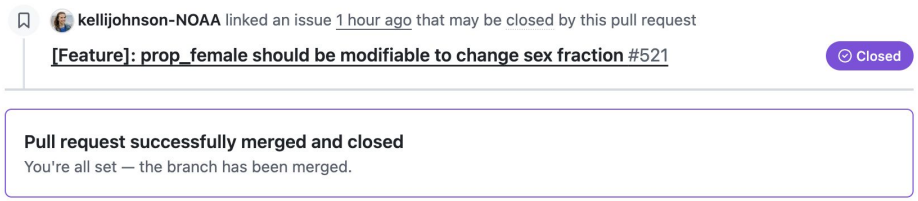
Project Objectives:

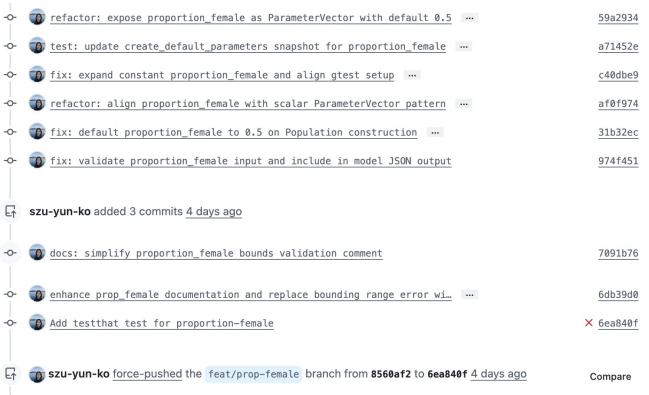
- Extend NOAA-FIMS (Fisheries Integrated Modeling System) to support user-defined sex structure in population models.
- Support three different model types:
  - **User-configured sex-ratio at age:** simply allow users to define a sex ratio rather than defaulting to 50/50 (female/male).
  - **Implicit two-sex model:** using user-configured female proportion to derive sex-specific population abundances to use in computations.
  - **Explicit two-sex model:** dynamically model life history differently between different sex groups, including sex-specific mortality, abundances, weight-at-age…etc.
- Implement the models by adding extendable data structures that can accommodate future partitioning needs.
  - The design mainly references New Zealand's CASAL2 Software.

Project #6: Add sex structure to FIMS statistical catch-at-age model

Major Milestones:

- Implement the user-configurable sex-ratio option. (PR #1494 Merged)
  - Status: This PR has been reviewed by three mentors and approved on June 23th.
  - While working on this PR, I dived into the fundamental data structures used in FIMS, and also familiarized myself with FIMS' data flow and existing infrastructure (e.g. TMB).

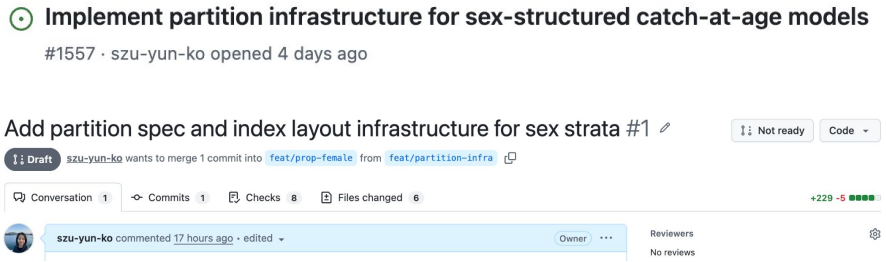


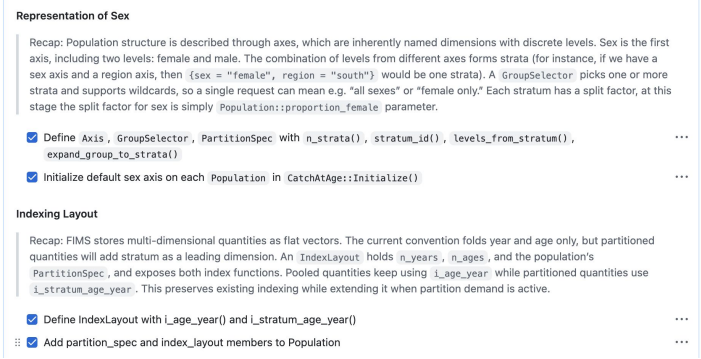


Project #6: Add sex structure to FIMS statistical catch-at-age model

Major Milestones:

- Implement partition infrastructure for sex-specific models (e.g. explicit two-sex model) (Issue #1557, Draft PR)
  - Created an issue to document the design directions and TODOs. The TODOs divide the task into three categories: the representation of sex-structure, the indexing and the partitioned storage.
  - All the TODOs are implemented (ON TRACK), and review will start after minor fixes.





Project #6: Add sex structure to FIMS statistical catch-at-age model

Plans up to midterm evaluation (July 11th):

- Complete the implementation for partitioning infrastructure (Issue #1557) and demand-based calculations (Issue #1587), convert to PR and start the review process (UNDER DEVELOPMENT).

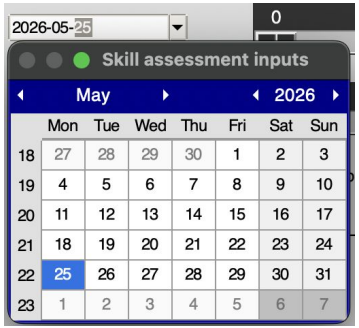
Plans after midterm evaluation:

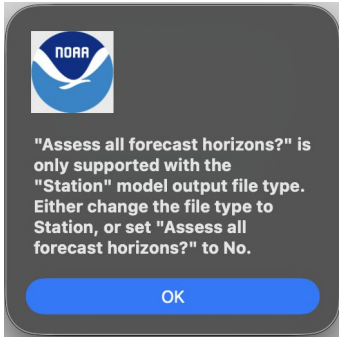
- Complete explicit two-sex model life history modeling (~ Mid July)
  - Natural Mortality, Fishing Mortality, Abundance, Selectivity…etc.
- Wire the explicit model with the partitioning infrastructure (~Early Aug.)
  - At this stage we will bind the two parts (storage and computation) together.
- Implement implicit two-sex model (before final evaluation)
  - This should be simple after the infrastructure has been laid out.

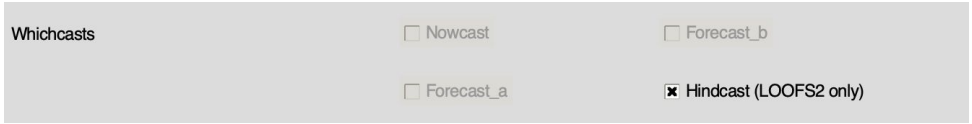
Project #7: Enhance NOS skill assessment package's user and developer experience

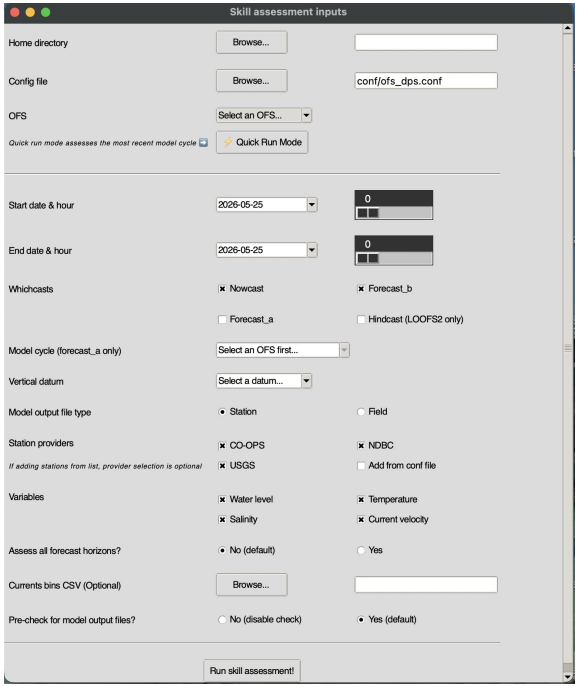
Phase 1 - User Experience Enhancement

PR 1 - GUI: add SECOFS, config picker, calendar popup fix, date/horizon validation, and LOOFS2 whichcast UI blocker(Merged)



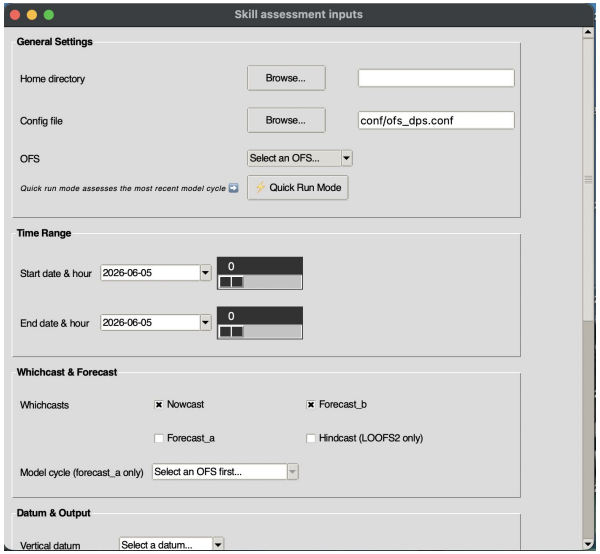


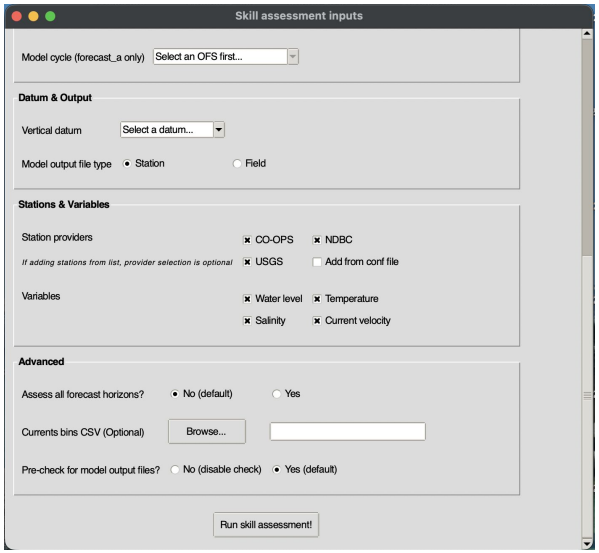




Project #7: Enhance NOS skill assessment package's user and developer experience

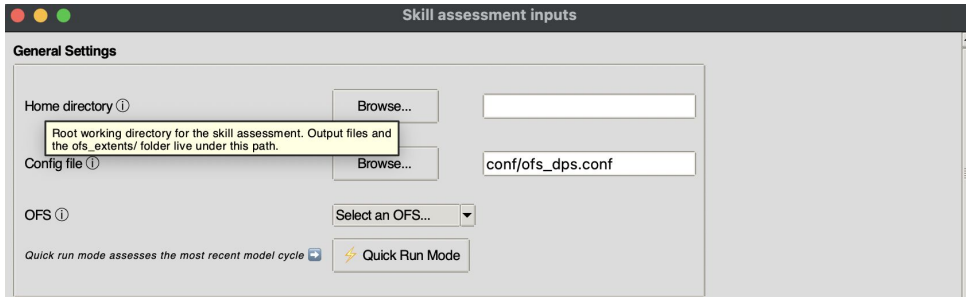
PR 2 - refactor(gui): Phase 2 foundation refactor
GUI helpers, tests, GuiParams, labeled sections (Merged)

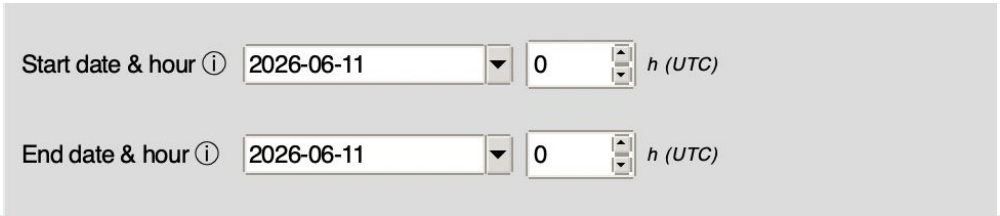


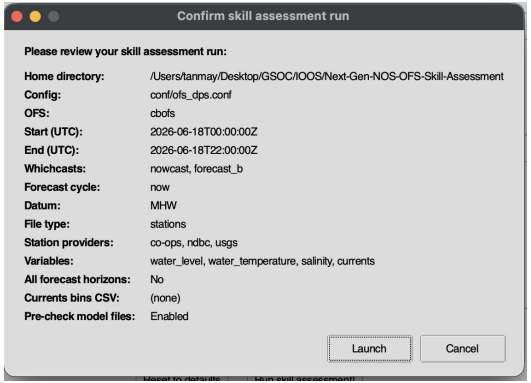
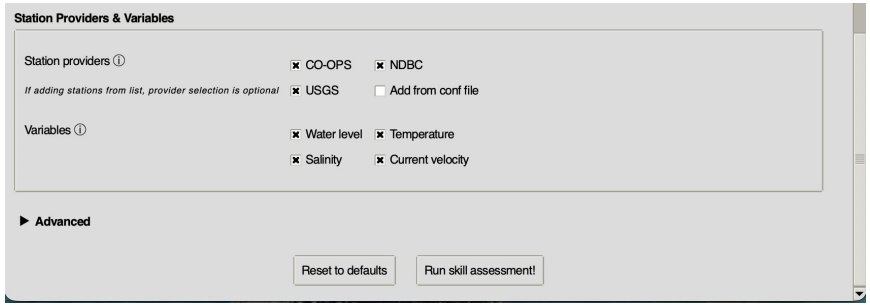
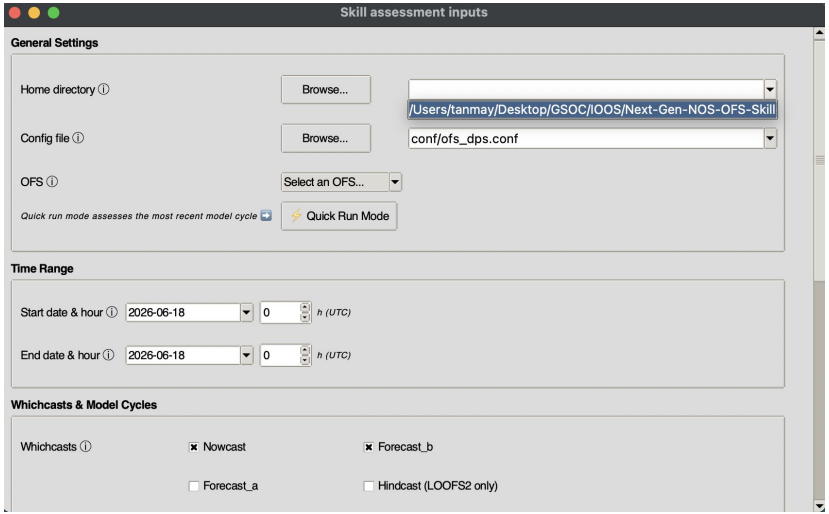


Project #7: Enhance NOS skill assessment package's user and developer experience

PR 3 - feat(gui): Phase 3 - UX modernization (shared theme, spinboxes, tooltips, validation highlighting, ttk migration) - (Merged)





Project #7: Enhance NOS skill assessment package's user and developer experience

PR 4 - Additional GUI modernization and Modular sub-GUIs (Ongoing)