IOOS GSoC Project Update Meeting #2

Friday July 24, 2026

Project #1: Enhancing noaa_coops

1) Derived Product Client (Phase 1)

Status: Completed, Merge pending - Issue #4,

- NOAA exposes a separate **Derived Product API (DPAPI)** for sea level trends, extreme water levels, sea level rise projections, and more previously inaccessible through `noaa_coops`.
- **Challenge**: DPAPI has 5 distinct URL patterns depending on product type (e.g. `/htf/`, `/htb/`, `/product/`, query-param-based) → no single URL builder can handle all of them.
- Built `_build_dpapi_url()` to categorize products into their pattern groups and route accordingly.
- Added `get_derived_product()` as a single bound Station method returns a pandas DataFrame for consistency.

```
c = Station("9447130")
df=c.get_derived_product(
    product="extremewaterlevels",
    units="metric",
    level_type="low")
df.head()
```

2] ✓ 2.5s

	type	date	status	stationId	stationName		stationTitle	state
0	GEV_LO	01/05/1916	YES	9447130	Seattle	9447130	Seattle, WA	WA
1	GEV_LO	07/07/1929	YES	9447130	Seattle	9447130	Seattle, WA	WA
2	GEV_LO	12/18/1929	YES	9447130	Seattle	9447130	Seattle, WA	WA
3	GEV_LO	11/30/1936	YES	9447130	Seattle	9447130	Seattle, WA	WA
4	GEV_LO	01/08/1947	YES	9447130	Seattle	9447130	Seattle, WA	WA

epoch	epochMidyear	latitude	longitude	annotation
1983-2001	1992	47.602639	-122.339306	NaN
1983-2001	1992	47.602639	-122.339306	NaN
1983-2001	1992	47.602639	-122.339306	NaN
1983-2001	1992	47.602639	-122.339306	NaN
1983-2001	1992	47.602639	-122.339306	NaN

Key design decisions (after discussion with mentors)

- Product-specific sub-parameters (`detail` for sea level trends, `level_type` for extreme water levels) let one method serve multiple response shapes
- Two DataFrame shapes depending on product: repeated-metadata rows for exploded sub-lists (matches the existing RFA pattern), single-row for summary-style products

Project #1: Enhancing noaa_coops

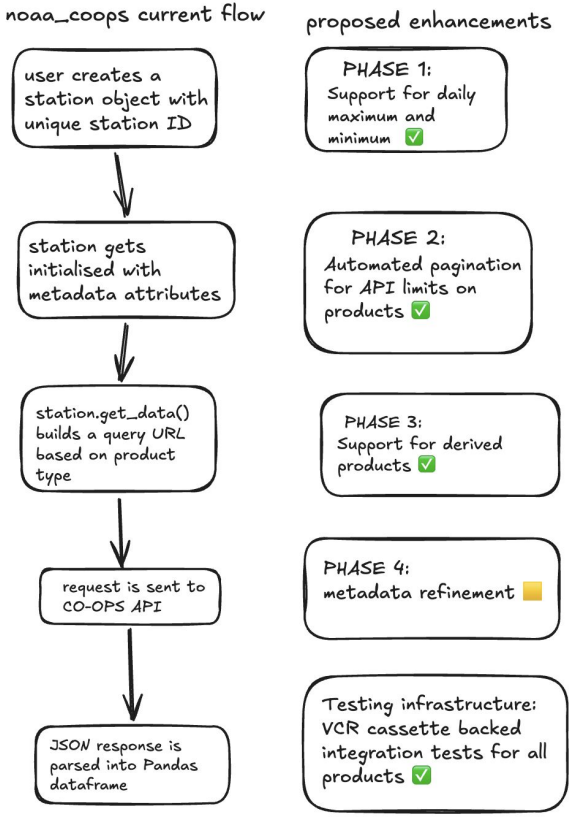
2) Metadata Refinement

Status: In progress - Issue #5

- NOAA's mdapi returns different populated fields depending on station type (water-level, tide-prediction, currents, predicted-currents), but the response shape doesn't cleanly signal which type a station is.
- **Challenge:** the original code branched with exclusive `if/elif` on key presence but some keys are both present on nearly every response regardless of station type.
- This silently dropped real tide-prediction data on stations that also carried a (mostly-null) `datums` block.
- Verified empirically across ~10 live stations (reference, subordinate, historic, decommissioned) → so exclusive branching was structurally wrong, not just buggy.
- Rewrote the branch logic as independent value-based checks instead of exclusive key-presence checks, so both attribute sets populate whenever real data exists for each.

Key design decisions:

- Discriminate by nested **value** not key presence

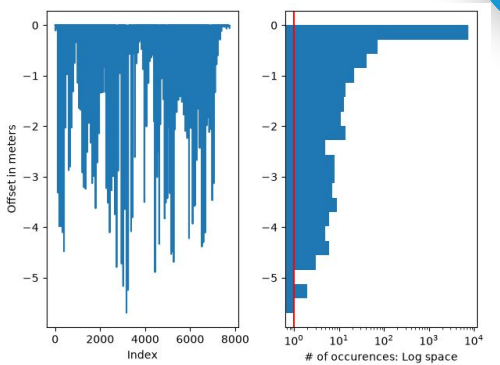


Project #2: Filling the gaps for ocean glider data quality control

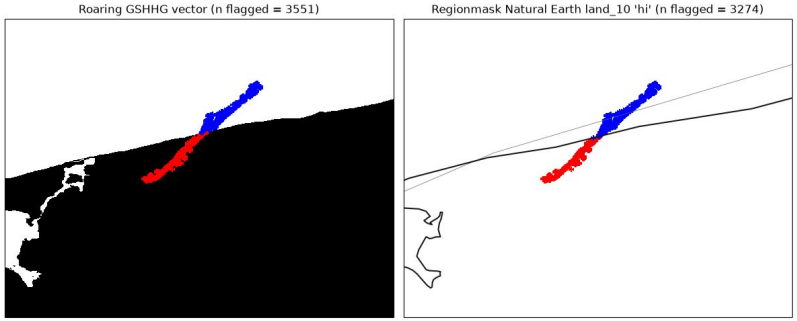
Stage 2: Group 1 Required Tests

- Created PRs for `timing` and `syntax` tests. *New* to the toolbox!
- `timing`
  - Broke this test up into several individual tests
  - Fit multiple use cases
- `syntax`
  - Created a simple test
  - Long journey to find the “rawest” data, try to parse the binary
  - CRC test was not worth it
    - Niche
    - Done by manufacturers
    - Not intelligible to intended audience
- `location` test(s) underway!
- Stage due end of August

Challenges: Identifying and avoiding rabbit holes

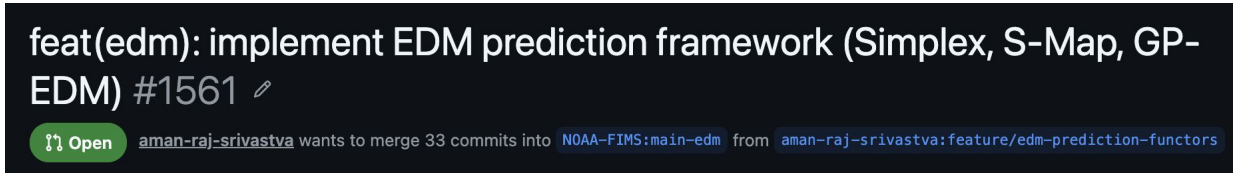


Great circle distances: Δ(Haversine, WGS84)



Comparing resolution of GSHHG (left) vs Natural Earth (right) coastlines for “on-land” flagging

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



Following the Delay Embedding infrastructure (#1528), implemented three EDM prediction methods:

1. Simplex Projection
Nearest-neighbor based nonlinear forecasting
Distance-weighted predictions
Configurable forecast horizon

2. S-Map
Locally weighted linear regression
Configurable localization parameter θ
Standard EDM S-Map prediction

3. GP-EDM
Gaussian Process prediction over delay embeddings
ARD/input-specific length-scale parameters

Also implemented:

Shared prediction infrastructure
Rcpp interfaces for all three methods
C++ GoogleTest coverage
R testthat coverage
Numerical/reference validation

Current status:

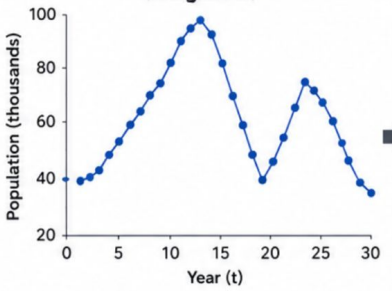
Based on reviews and feedback received from mentors
Prediction Framework PR #1561 is Under Review

Current focus: Refactoring and optimizing the implementation to better align with the FIMS architecture.

Project #3: Integrating EDM into FIMS: EDM Pipeline

1. Observed Time Series

Fish Population Over Time (in thousands)



Example Characteristics

- Nonlinear & non-stationary dynamics
- Potential cycles, regimes, and noise
- Goal: Predict future population from past observations

2. Delay Embedding ($E = 3, \tau = 1$)

Each row is an embedding vector constructed from past $E = 3$ values

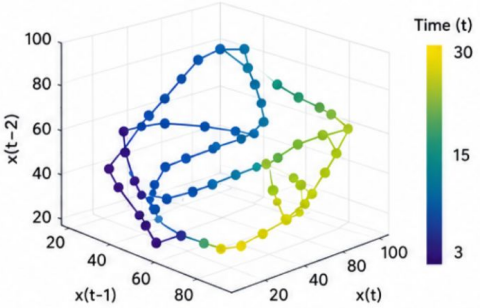
t	x(t)	x(t-1)	x(t-2)		Target x(t+1)
3	55.1	48.3	40.2	→	60.7
4	60.7	55.1	48.3	→	67.2
5	67.2	60.7	55.1	→	74.1
		⋮			⋮
27	41.7	47.8	53.6	→	38.5
28	38.5	41.7	47.8	→	36.9
29	36.9	38.5	41.7	→	39.6

Parameters

- Embedding dimension (E) = 3
- Time lag (τ) = 1
- Use past E values to predict the next value x(t+1)

3. Reconstructed State Space ($E = 3$)

Each point is [x(t), x(t−1), x(t−2)] in 3D space



Interpretation

- The trajectory reveals the underlying dynamics of the system
- EDM models operate on these embedded states

Current (Latest) Observations

x(29) = 36.9
x(28) = 38.5
x(27) = 41.7

Current State Vector (E=3)

[36.9, 38.5, 41.7]

Simplex Idea: "Find similar historical states."

Find E+1 nearest neighbors
↓
Weight by distance
↓
Weighted future values
↓
Prediction (Point prediction only)

S-Map Idea: "Fit a local linear model around the current state."

Calculate distances
↓
θ controls how local the weighting is
↓
Weighted linear regression
↓
Prediction (Point prediction + local coefficients)

GP-EDM Idea: "Model similarity probabilistically."

ARD kernel compares embedded states
↓
Covariance matrix
↓
GP posterior
↓
Prediction (Probabilistic prediction + uncertainty

Project #3: Integrating EDM into FIMS

[WIP] feat(edm): integrate EDM prediction framework into FIMS #1620

Open aman-raj-srivastva wants to merge 39 commits into `NOAA-FIMS:main-edm` from `aman-raj-srivastva:feature/edm-fims-integration`

Goal: Integrate the standalone EDM predictors from PR #1561 into the FIMS statistical modeling and TMB pipeline.

EDMModel<Type> module (commited) — provides the FIMS-level wrapper that owns and runs the configured Simplex, S-Map, or GP-EDM predictor.

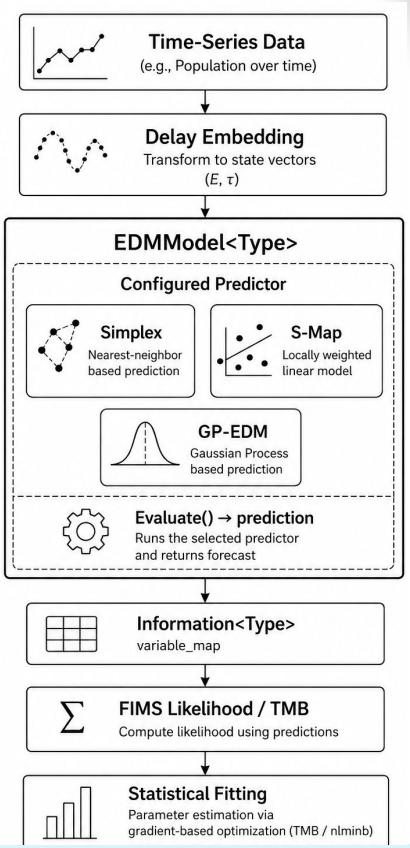
Information<Type> integration (commited) — registers EDM models and exposes their prediction outputs through variable_map.

Model::Evaluate() integration (commited) — executes EDM predictions as part of the FIMS model evaluation workflow.

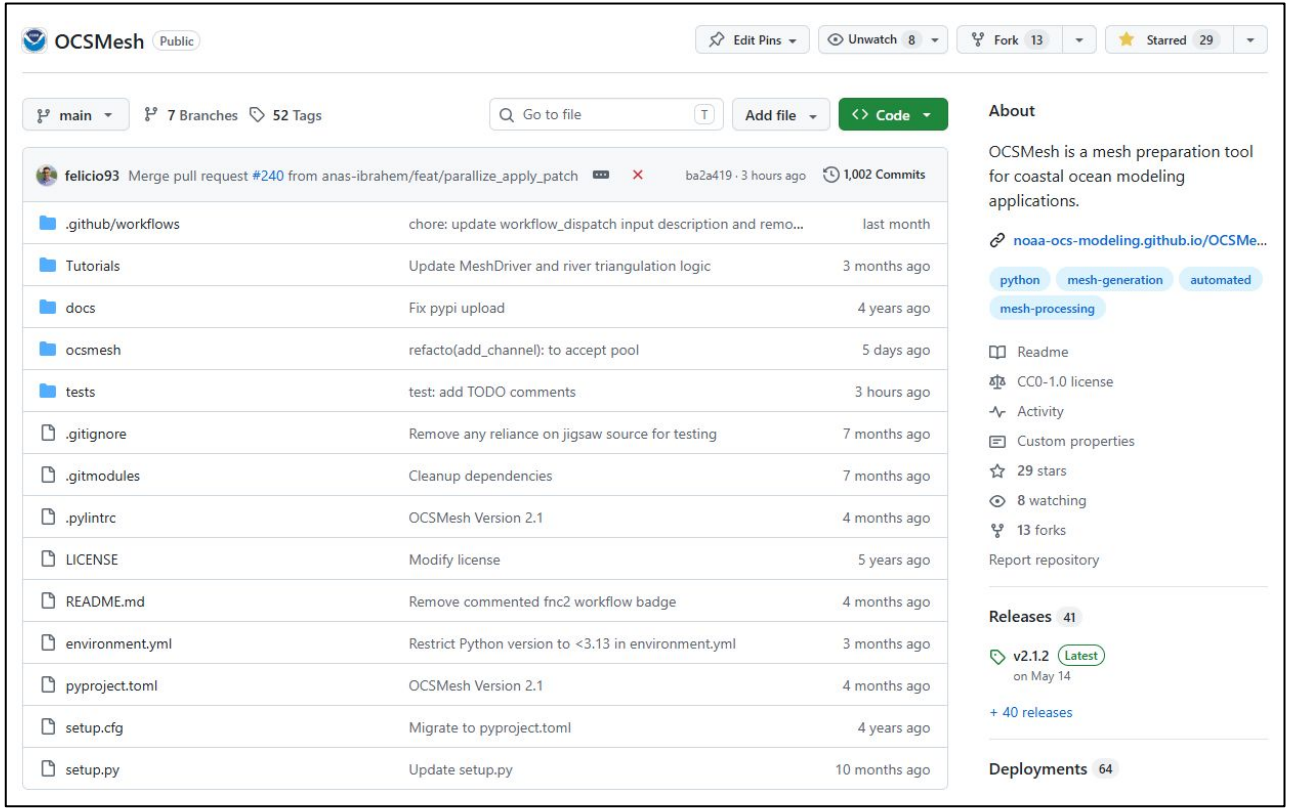
Rcpp & end-to-end integration (next) — connect the R-facing EDM interfaces with the TMB model and complete integration testing.

Current priority — iaddressing mentor review on PR #1561 before completing PR #1620.

Standalone EDM algorithms → reusable FIMS model components → likelihood-based statistical fitting



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

Merged on Jun 30th

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

Merged on Jul 1st

Scalable OCSMesh: Parallelization and Spatial Partitioning

Simple Runs (Local) After Phase 1

Benchmark Test Case	Raster Setup	Serial Time	Parallel Time	Speedup	Validation
Test Case 1	4 Rasters (20% Overlap)	16.14 s	10.02 s	**1.61x**	**PASS**
Test Case 2	4 Rasters (100% Overlap)	30.02 s	11.98 s	**2.51x**	**PASS**

Pre Phase 2: PR #245 vs PR #248 Comparison

Factor	PR #245 (Static Scatter/Gather)	PR #248 (Dynamic Send/Recv)
Architecture	Worker loop + bcast	MPITaskRunner + point-to-point send/recv
Rank 0	Coordinate AND compute	Coordinate, NO compute
Deadlock risk	High - bcast in control loop, one missing collective hands entire job	None - explicit send/recv, no collectives. One rank crashes only that one fails.
Load balancing	Static round-robin, task are distributed a priori (one rank can have 2 large tiles and the other 2 light ones)	Dynamic (idle worker pulls next task)
User-facing API	Requires if rank==0/else:mpi_worker_loop()	None
For our small cluster test	~35% faster	Slower (rank 0 overhead)
Large cluster scaling	Same	Expected to be better (coordinator cost)
Code complexity	Lower	Higher

Phase 2 - Due Aug 1st

- 2.0 - Create IOOS Sandbox (MPI Instance) **Completed**
- 2.1 - Initialization:
  - MPITaskRunner class with dynamic send/recv task distribution **Completed**
- 2.2 Initial MPI Implementation for hfun refinements:
  - Currently implemented with multiprocessing.Poll not MPI. We still need to:
    - Create a execution_mode == 'mpi' branch using MPITaskRunner.dispatch()

  Not Started - Due Aug 1st
- 2.3 MPI Validation:
  - Develop a full test suite for the MPI implementation.
  - Test on an actual HPC cluster using multi-node jobs.

  Not Started - Due Aug 1st
- Nice to have:
  - Pre-balance tasks before dispatch
    - Estimate per-tile compute cost (based on raster resolution/grid size) and assign heavier tiles to faster workers or distribute them to balance wall-clock time (even a simple sort tiles by grid size descending should work).
  - **MPI for the "fast" method path.**

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

- Is this even needed now that we are using dynamic send/recv? Yes (probably)! For very large applications.

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

The problem:

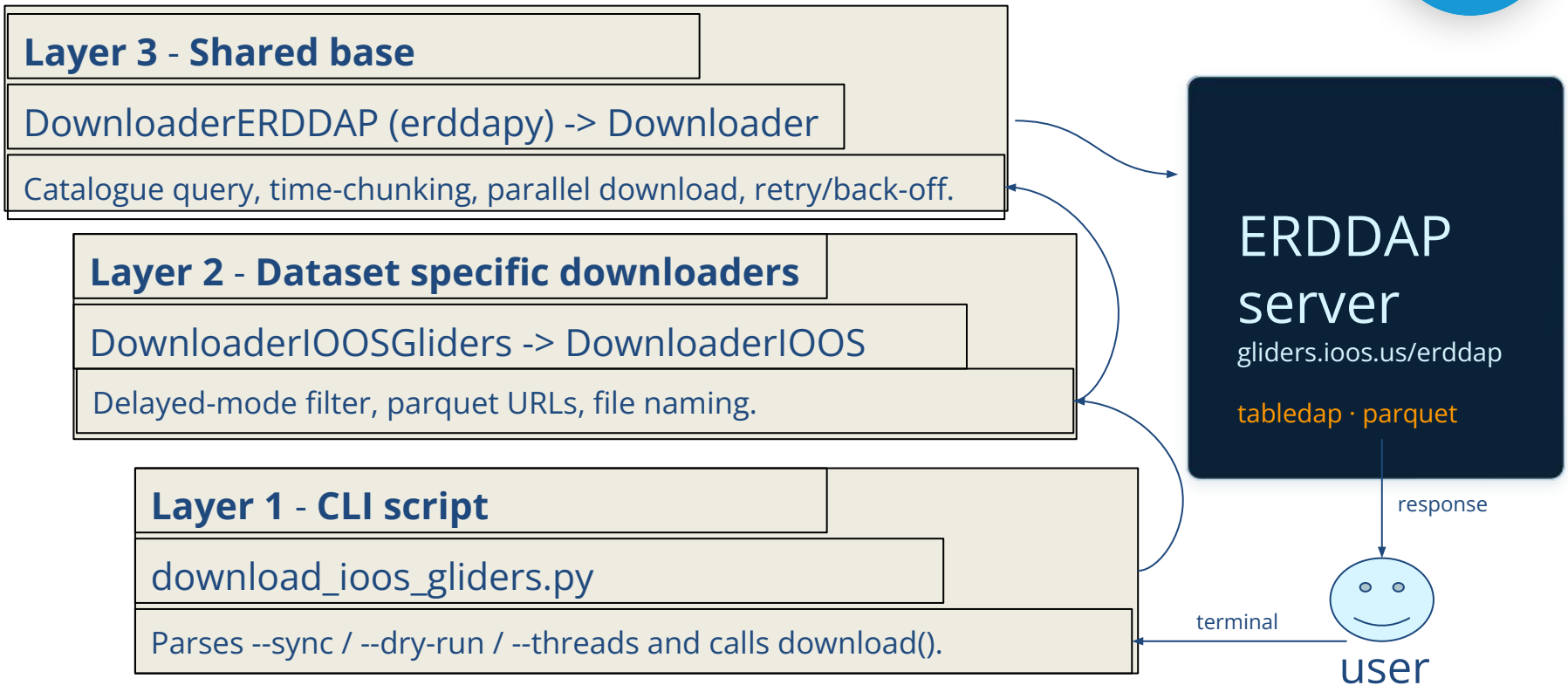
- ❏ Static downloaders can't track a dynamic server.
- ❏ gliders.ioos.us/erddap hosts thousands of glider data deployments. Existing datasets get updated time to time.
- ❏ We need to have a syncing mechanism along with the basic download option.

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

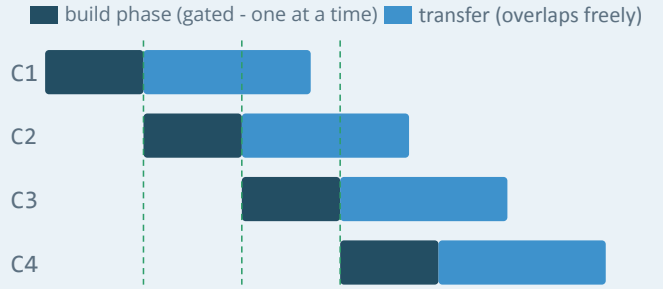
Download 24-hour windows, then merge locally into one parquet. If a window still returns 413(large payload), step the schedule down: 24h -> 12h -> 6h.

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

Decisions from collaborative meeting with IOOS GDAC Team regarding error handling (Project #5)

- The solution implemented so far is already well optimized with respect to the existing erddap server state.

- Enrico(mentor) suggested not to change anything right now as it all seems good and working.

- The issues we faced so far are some of the expected and known behavior from the erddap server, so its a limitation on erddap server rather than our implemented solution.

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.
  - There were around 3500 unique variables across the datasets. Not every variable was needed for crocolake. So we created a curated list of desired variables from the available variable.
  - https://github.com/boom-lab/crocolaketools/pull/50 (Merged).
- Mitigation of Crocolakeloader dependency from crocolaketools.
  - https://github.com/boom-lab/crocolaketools/pull/53 (Merged)
  - https://github.com/boom-lab/crocolakeloader/pull/10 (Merged)
- Created a CI pipeline for syncing Crocolaketools and CrocolakeLoader. (Needs org permission)
- Animal Telemetry stretch goal. (skipped for now, as the available variables don't match crocolake convention)
- Enhanced the erddap backend on CrocoLakeTools with custom constraint controlling.
  - https://github.com/boom-lab/crocolaketools/pull/59 (Review in Progress)
- As the main scope of the GSoC project is almost covered we planned to work on IOOS data converter scripts for the remaining time. (working on it right now)

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

Project Milestones

Data structure infrastructures for partitioning the population.

Code submitted and reviewed in PR#1587.

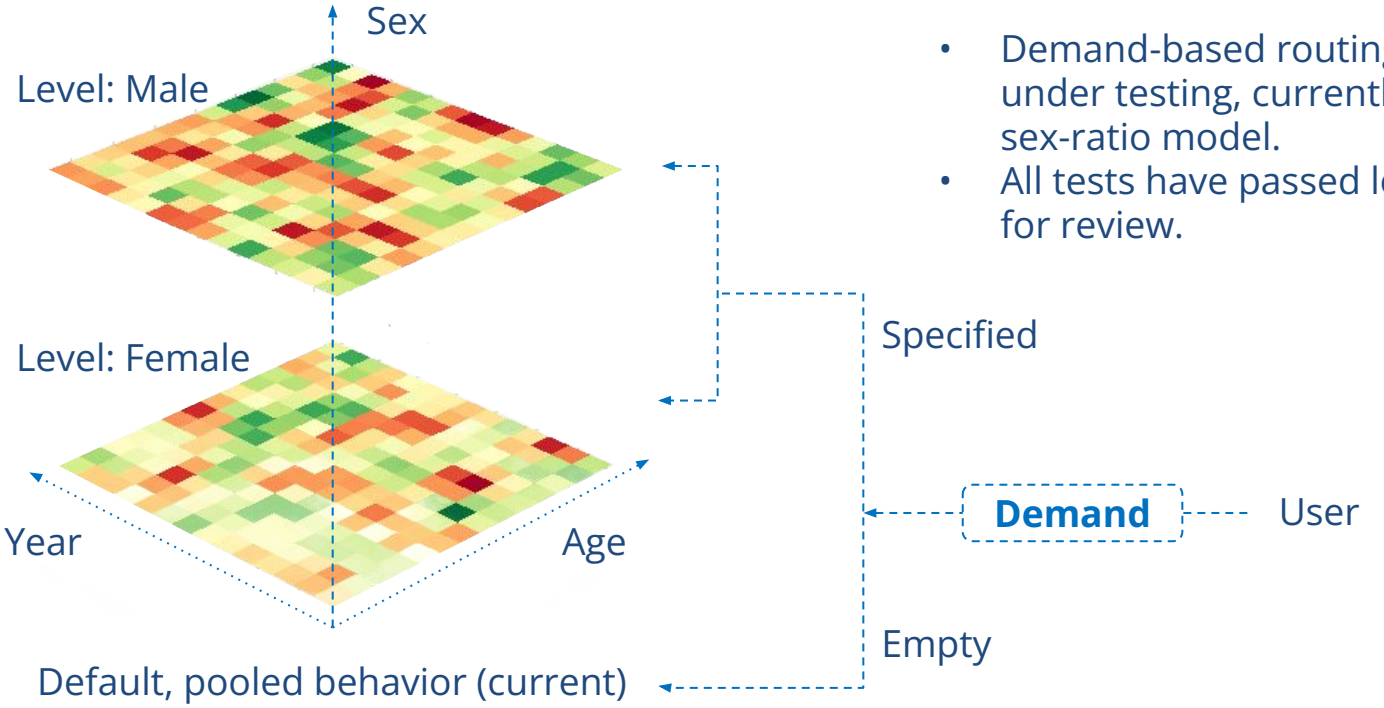
Demand-based computation, determine which partitions to compute using what model.

Code changes are completed locally, doing some testing before submitting for mentors' review.

Implement model options for computing partitioned values.

Sex-ratio at age model merged. Ongoing discussion for explicit two-sex model in issue #1625.

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



In Progress

- Demand-based routing is completed and under testing, currently wired to the sex-ratio model.
- All tests have passed locally, soon ready for review.

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

User-supplied, initial sex-ratio

Total mortality is the sum of natural (M) and fishing (F) mortality

$$N^{\text{fem}}_{a,y} = p^{\text{fem}}_{a-1,y-1} N_{a-1,y-1} e^{-Z^{\text{fem}}_{a-1,y-1}}$$

$$N^{\text{male}}_{a,y} = \left(1 - p^{\text{fem}}_{a-1,y-1}\right) N_{a-1,y-1} e^{-Z^{\text{male}}_{a-1,y-1}}$$

$$N_{a,y} = N^{\text{fem}}_{a,y} + N^{\text{male}}_{a,y}$$

$$p^{\text{fem}}_{a,y} = \frac{N^{\text{fem}}_{a,y}}{N_{a,y}}$$

$$Z^{fem}_{a,y} = M^{fem}_{a,y} + F^{fem}_{a,y}$$

$$F_{f,a,y} = F_{f,y} f_y S_f(a)$$

Fishing mortality is determined by selectivity (S), which can be sex-specific

Abundance (age, year) then becomes sex-specific

Dynamically update p_fem at the next step using female abundance

Open Introduce the explicit two-sex model for the FIMS catch-at-age population dynamics #1625

Parent: #1582

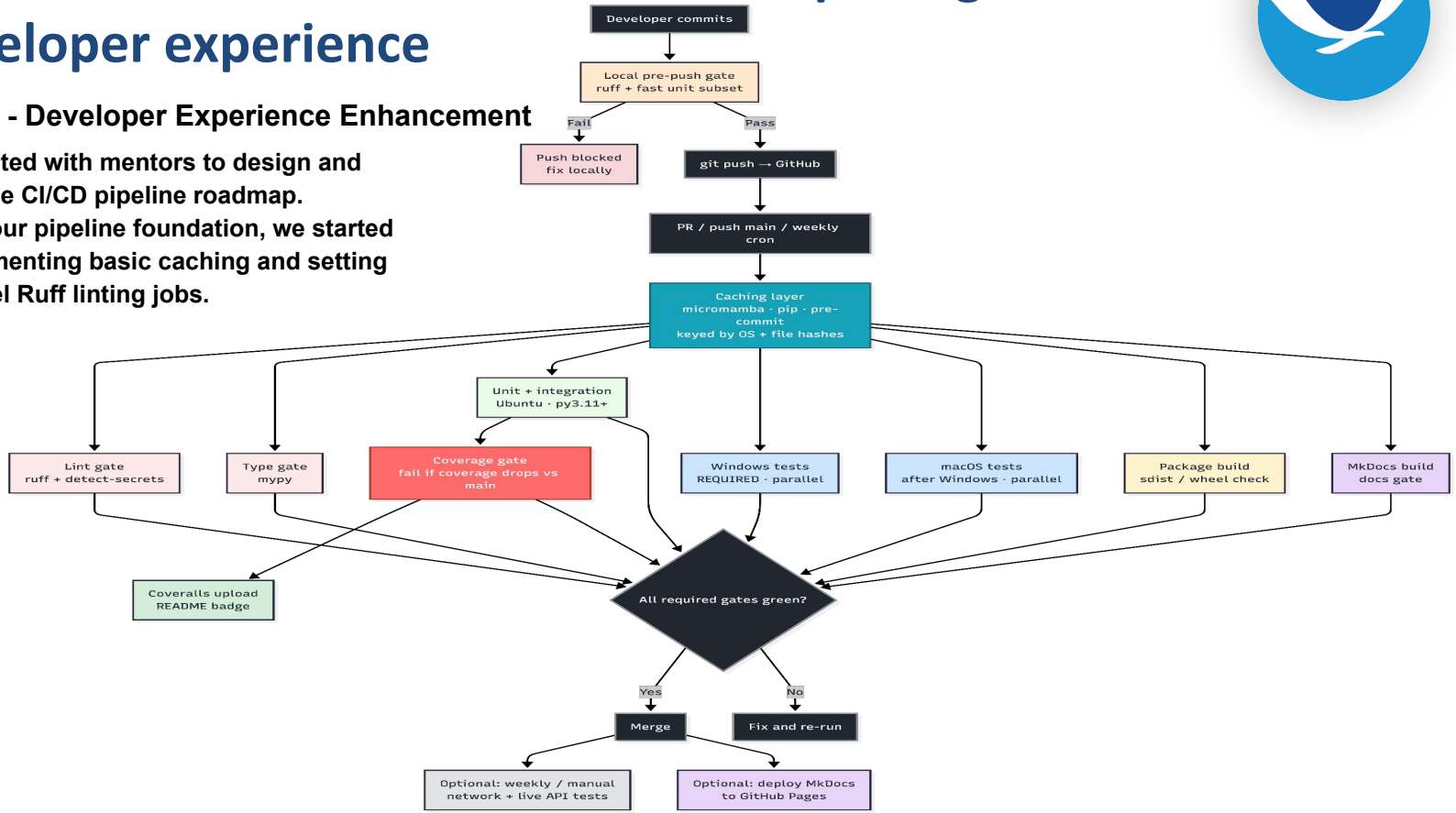
Project #7: Enhance NOS skill assessment package's user and developer experience

1) **Phase 1 - User Experience Enhancement**

Status: Completed, Merged

- We stopped bad runs before they start: date checks, OFS-specific whichcast rules (like LOOFS2), config picking from the UI, file path dropdowns.
- Shared helpers, validators, and a typed params model moved into gui_helpers.py, with tests, so the GUI code wasn't one giant file anymore.
- Consistent theme, spinboxes instead of sliders, tooltips, red highlights on bad fields, and smoother scrolling across platforms.
- One launcher, five tools (1D, 2D, ice, obs, model), shared last-used settings, live validation, a "are you sure?" summary, progress while it runs, recent paths, and reset-to-defaults.

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

2. Phase 2 - Developer Experience Enhancement

- Collaborated with mentors to design and finalize the CI/CD pipeline roadmap.
- To build our pipeline foundation, we started by implementing basic caching and setting up parallel Ruff linting jobs.