# Designing EAGLE

## A Portable and Reproducible Framework for AI/ML Weather Prediction

Contributors: Paul Madden[1,2], Emily Carpenter[1,2], Mariah Pope[3,4], Anil Kumar[3,4], Kristopher Booker[3,4], D. Alex Burrows[3,5]

1 Cooperative Institute for Research In Environmental Sciences, University of Colorado
2 NOAA Global Systems Laboratory
3 NOAA Earth Prediction and Innovation Center
4 Tomorrow.io
5 RedLine Performance Solutions

# What is EAGLE?

- Experimental AI Global and Limited-area Ensemble forecast system
- "A joint effort between NOAA Research Laboratories, EPIC, and NWS to provide the ability to rapidly test, develop, and demonstrate Artificial Intelligence (AI) models for global and limited area ensemble forecasting"
- Integrates various community software packages: ufs2arco and eagle-tools for data transformation, Anemoi for training and inference, wxvx for verification, and uwtools (Unified Workflow Tools) for configuration management and execution

UIFCW26
A UFS Collaboration Powered by EPIC

# Design Questions

- How to get all these software packages to coexist?
  - All are fortunately Python packages, but some from conda and some from pip
- How to install them at all?
  - Is this even EAGLE's responsibility?
- How to configure so many components sanely?
  - And without having to manually sync values
- How to execute components in pipeline steps?
  - Maybe locally, maybe via a batch system

# Build System

- Decision: Provisioning a software environment *is* EAGLE's responsibility
  - Developers, end users, and continuous integration (CI) all take the same, reproducible steps via automation
- A **`setup`** script installs Miniforge (conda), and creates five distinct environments: **`base`**, **`data`**, **`anemoi`**, **`wxvx`**, **`visualization`**
  - This allows different, potentially conflicting package versions in different environments
  - Primary (and some transitive dependencies) captured as YAML files under **`envs/`**
  - conda can install pip packages when needed

UIFCW26
A UFS Collaboration Powered by EPIC

# Build System: Example YAML

```
channels:
  - nodefaults
  - conda-forge
  - oar-gsl
  - ufs-community
dependencies:
  - pip
  - python 3.13
  - requests 2.34.*
  - uwtools 2.16.*
  - wxvx >=0.7.3
  - xesmf 0.8.*
  - zarr 2.18.*
  - pip:
      - eagle-tools==0.8.1
```

- For the `wxvx` conda environment
- Uses three conda channels, but avoids license-restricted `defaults`
- Pins primary packages
- Pins select transitive packages that require specific versions or that would be re-installed by pip
- Installs `eagle-tools` from PyPI, as it is not published as a conda package

# Configuration System

- EAGLE uses **`uwtools`** for configuration management
- YAML configs under **`config/`** can be composed per use case:

  **`base+nested+ursa`** vs **`base+global+ursa`**

```
# from base.yaml
app:
  base: /path/to/eagle
  experiment_name: default
```

```
# from ursa.yaml
app:
  experiment_name: nested
  partitions:
    netaccess: u1-service
```

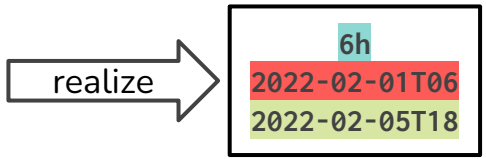
compose

```
# final eagle.yaml
app:
  base: /path/to/eagle
  experiment_name: nested
  partitions:
    netaccess: u1-service
```

# Configuration System: Jinja2

- Configs can use Jinja2 expressions to reference other values, compute new ones, include environment variables, etc.

```yaml
# from base.yaml
app:
  time:
    freq: '{{ (app.time.step.total_seconds() / 3600) | int }}h'
    start: !datetime 2022-02-01T00
    step: !timedelta 6
    stop: !datetime 2022-02-05T12
ufs2arco:
  source:
    analysis:
      t0:
        start: '{{ (app.time.start + app.time.step).strftime("%Y-%m-%dT%H") }}'
        end: '{{ (app.time.stop + app.time.step).strftime("%Y-%m-%dT%H") }}'
```



# Configuration system: JSON Schema

- Validation via JSON Schema ranges from missing values:

```
[2026-07-14T19:06:43]    ERROR 1 schema-validation error found in grids_and_meshes config
[2026-07-14T19:06:43]    ERROR Error at grids_and_meshes:
[2026-07-14T19:06:43]    ERROR   'global_grid_resolution_deg' is a required property
[2026-07-14T19:06:43]    ERROR YAML validation errors
```

- To more nuanced value checks:

```
[2026-07-14T19:11:18]    ERROR 1 schema-validation error found in grids_and_meshes config
[2026-07-14T19:11:18]    ERROR Error at grids_and_meshes.conus_grid_resolution_km:
[2026-07-14T19:11:18]    ERROR   7 is not a multiple of 3
[2026-07-14T19:11:18]    ERROR YAML validation errors
```

- Schemas can also validate types (**`string`**, **`integer`**), number of list elements, and much more

UIFCW26
A UFS Collaboration Powered by EPIC

# Execution System

- EAGLE uses **`uwtools`** for component execution
- **`uwtools`** ships with 24 NWP component drivers, but also supports *external drivers* defined by applications
- Drivers share required (**`executable`**, **`rundir`**) and optional (**`batchargs`** for job-scheduler interaction) config parameters
  - Drivers can define custom parameters, too
- Benefits to implementing drivers:
  - Consistent logging, CLI, runscript, output files
  - Insertion of batch directives (**`#SBATCH`**) in runscripts
  - Schema validation of YAML configurations
- EAGLE drivers are auto-formatted, linted, typechecked and 100% unit tested

# Execution System: Config Excerpt

```
inference:
  anemoi:
    # Anemoi-specific config...
  execution:
    batchargs:
      --gres: gpu:h100:1
      nodes: 1
      partition: u1-h100
      queue: gpuwf
      export: false
      memory: 16G
      rundir: /scratch4/BMC/gsd-hpcs/Paul.Madden/EAGLE/run/default/inference
      walltime: 00:10:00
    envcmds:
    - source /etc/profile.d/profile-slurm.sh
    - source /scratch4/BMC/gsd-hpcs/Paul.Madden/EAGLE/conda/etc/profile.d/conda.sh
    - conda activate anemoi
    - set -ex
    executable: srun --export=ALL eagle-tools inference inference.yaml
  rundir: /scratch4/BMC/gsd-hpcs/Paul.Madden/EAGLE/run/default/inference
  validate: true
```

# Execution System: Runscript

```
$ cat /scratch4/BMC/gsd-hpcs/Paul.Madden/EAGLE/run/default/inference/runscript.inference
#!/bin/bash

#SBATCH --account=epic
#SBATCH --chdir=/scratch4/BMC/gsd-hpcs/Paul.Madden/EAGLE/run/default/inference
#SBATCH --export=NONE
#SBATCH --gres=gpu:h100:1
#SBATCH --mem=16G
#SBATCH --nodes=1
#SBATCH --output=runscript.inference.out
#SBATCH --partition=u1-h100
#SBATCH --qos=gpuwf
#SBATCH --time=00:10:00

source /etc/profile.d/profile-slurm.sh
source /scratch4/BMC/gsd-hpcs/Paul.Madden/EAGLE/conda/etc/profile.d/conda.sh
conda activate anemoi
set -ex

time srun --export=ALL eagle-tools inference inference.yaml
test $? -eq 0 && touch runscript.inference.done
```

UIFCW26
A UFS Collaboration Powered by EPIC

# Execution System: Example Invocation

```
+ uw execute --config-file eagle.yaml --module eagle/inference/inference.py --classname Inference --task run --batch
[2026-07-15T04:08:03]     INFO Schema validation succeeded for inference config
...
[2026-07-15T04:08:08]     INFO inference runscript.inference: Executing
[2026-07-15T04:08:08]     INFO inference runscript.inference: Ready
[2026-07-15T04:08:08]     INFO inference inference config: Executing
[2026-07-15T04:08:08]     INFO inference inference config: Ready
[2026-07-15T04:08:08]     INFO inference provisioned run directory: Ready
[2026-07-15T04:08:08]     INFO inference run via batch submission: Executing
[2026-07-15T04:08:08]     INFO Running in /scratch4/BMC/gsd-hpcs/Paul.Madden/EAGLE/run/default/inference:
[2026-07-15T04:08:08]     INFO   set -o pipefail && sbatch runscript.inference 2>&1 | tee runscript.inference.submit
[2026-07-15T04:08:08]     INFO inference run via batch submission: Ready
[2026-07-15T04:08:08]     INFO inference run: Ready
```

UIFCW26
A UFS Collaboration Powered by EPIC

# Task Runner

- Use `make` as a task runner, for example:
  - Create software environment

    ```
    make env cudascript=ursa
    ```
  - Create comprehensive configuration file

    ```
    make config compose=base:nested:ursa >eagle.yaml
    ```
  - Run inference

    ```
    make inference config=eagle.yaml
    ```
- Some targets perform a pipeline step, and can be called directly by users, by workflows (e.g. ecFlow, Rocoto), by continuous integration, etc.
- Some targets run code-quality checks
- Each target activates the appropriate conda environment

# Summary

- We are managing EAGLE's complexity via:
  - A common, automated, git-versioned build system
  - Composable, schema-checked YAML configurations
  - Component drivers for pipeline execution
  - Robust code-quality checks
- Developers run a full suite of code-quality tools locally, then CI runs the same tests in GitHub *and* a full end-to-end Nested-EAGLE pipeline on Ursa
- Both global and nested configurations are available for both retrospective and near-real-time use cases.
- A Rocoto workflow to drive the Nested-EAGLE pipeline is under development

# Thanks!