Running AI Weather Models: A Practical Guide

Adam Marchakitus
University of Chicago

Agenda

Core Concepts: Quick Definitions

- **VRAM (GPU memory)**
  - Video memory on the graphics card; distinct from system RAM
  - Holds model, weights, activations, and intermediate tensors
  - *Analogy:* factory floor space for machines and materials
- **Model**
  - Blueprint + learned parameters mapping inputs → outputs
  - Small on disk ≠ small in GPU memory (layers expand when loaded)
  - Different models → different VRAM requirements (e.g., AIFS ~24–30 GB; GenCast ~96 GB)
  - *Analogy:* a factory transforming raw materials into something useful

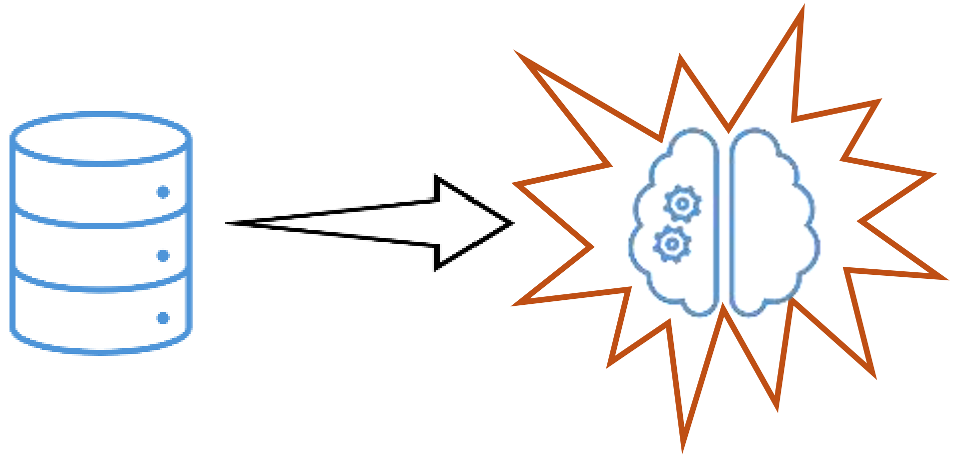
Trained Checkpoint vs Weights

- **Trained Checkpoint**
  - A **file** that encodes architecture + **weights** – tells the GPU what to do with the data
  - Common formats: .pkl .ckpt .npz
  - Downloaded from the developer who created/trained the model
  - Model families can have multiple checkpoints that are advantageous for different objectives (e.g., GraphCast: paper / operational / small)
  - Models can have multiple checkpoint files for different lead times
  - *Analogy:* the blueprint used to build the factory
- **Weights**
  - Learned numerical parameters use during inference
  - Often included within the checkpoint
  - Sometimes used colloquially to mean “inference-only checkpoint”
  - *Analogy:* the measurements and tolerances on the blueprint

Inference

- **Inference**
  - Convert initial conditions into a forecast
  - Model checkpoint is loaded and expanded into GPU memory, initial conditions are loaded and the GPU performs operations to obtain *T+1*
  - Often autoregressive (previous outputs feed the next step)
  - *Analogy:* switch the factory on and run production
- Why use GPUs for inference? Parallelization!
  - Inference is a series of matrix multiplications to transform the inputs
  - Matrix multiplication can be sped up through parallelization
  - GPUs excel at this due to parallelization from having many more cores
- Why not just use a CPU? It is possible in some cases but…
  - CPUs are great for serial tasks, but can be slower when it comes to parallelization
  - Multicore CPUs are expensive, and connection to RAM is slower than that between GPU and VRAM

Memory Reality Check



- **On-disk vs in-memory**
  - Example: *GenCast* ≈ 122 MB on disk → ~86 GB VRAM **per ensemble member**
  - **AIFS**: ~1 GB on disk ~24–30 GB VRAM
- Why is this the case?
  - Weights are compressed on disk, space required to store instructions is small
  - GPU memory is often pre-allocated to provide enough space for operations
  - Matrix multiplications result in higher dimensional data → Inference often requires intermediates that are stored temporarily in GPU memory
- **Takeaway:** budget for activations, precision, batch/lead length, ensemble size to avoid out-of-memory errors

Computing Environment: GPU Hardware

- **NVIDIA GPUs** are most widely supported
- **VRAM** is the #1 limiter for inference
- Common options:
  - **A100** (40/80 GB) — still useful, but being phased out
  - **L40** (48GB) — modern A-series replacement for many inference tasks
  - **H100** (80/96 GB) — suitable for large models like GenCast
  - **H200** (141 GB) — extra headroom for memory-hungry workloads

Widely Used AI Weather Models

Run Locally

- FourCastNet
- SNFO (FourCastNetv2)
- AIFS
- FuXi
- Pangu
- NeuralGCM
- GraphCast (small, 1º)
- GenCast (mini, 1º)

Requires HPC Hardware

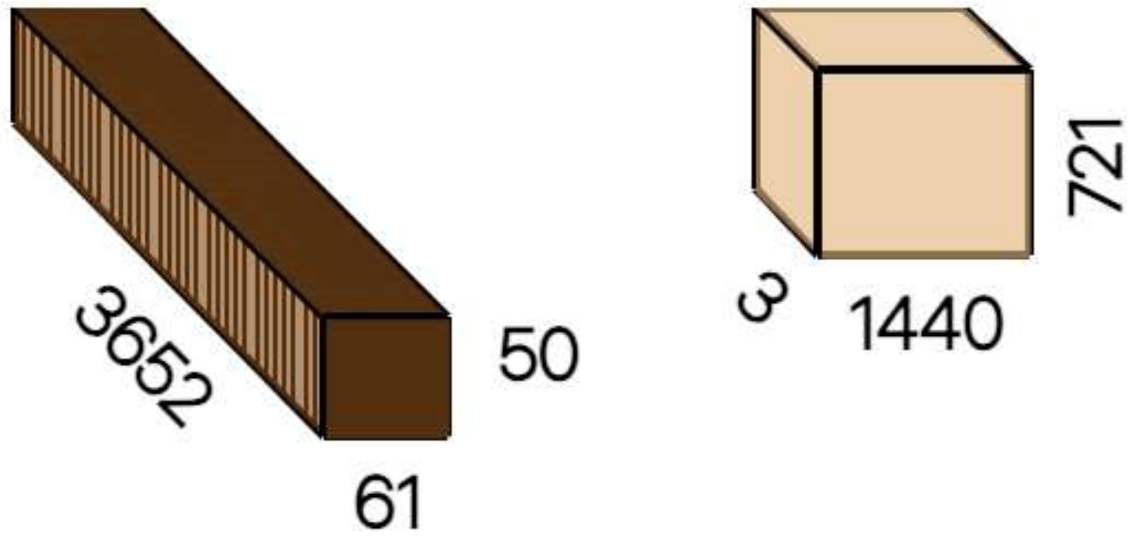
- GenCast
- GraphCast (37 pressure levels)

Computing Environment: Software

- **GPU Drivers** — OS ↔ GPU communication
- **CUDA** — NVIDIA toolkit/runtime enabling GPU tensor ops; must match driver/framework
- **Python** — control layer; heavy math runs in compiled C/C++/CUDA backends
- **Conda** — per-model Python env to avoid package conflicts (CUDA can be in-env)
- **ML Packages** — PyTorch, JAX; domain libs (e.g., ECMWF Anemoi interface)
- Data manipulation — Xarray, Dask

Data Foundations: Shapes & Dimensions

- **Gridded, multi-dimensional data** (often ~0.25° resolution)
- **Inputs (typical 4D):** time × latitude × longitude × level
- **Outputs (up to 6D):** init_time × lead_time × latitude × longitude × level × member



Data Formats

Zarr

- **What it is:** chunked, cloud-native array store
- **Pros**
  - Parallel read/write; fetch only needed chunks/metadata
  - Efficient chunking; append without rewriting all data
- **Cons**
  - Many files → can hit HPC file limits; slow bulk copy/delete

NetCDF

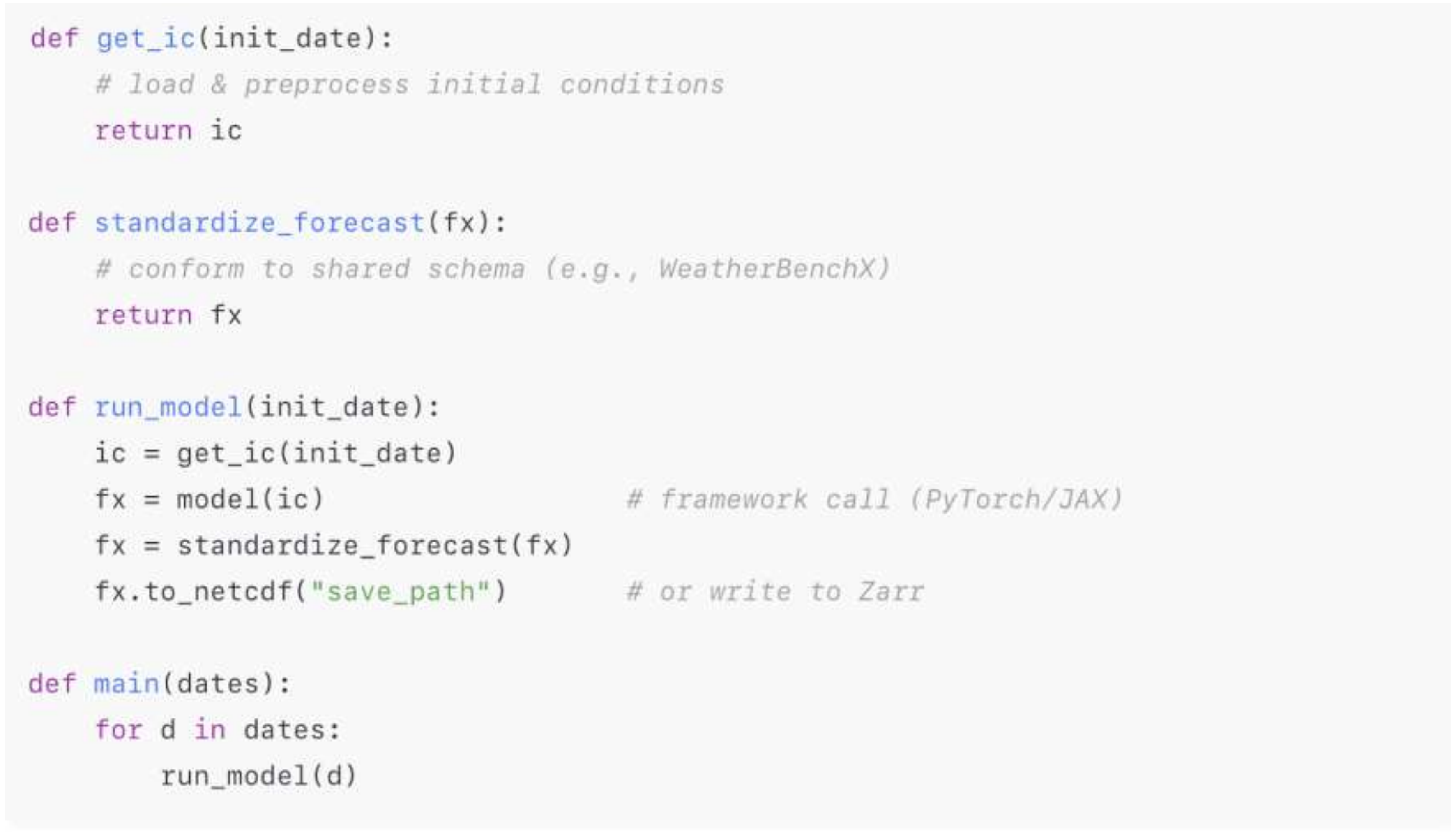
- **What it is:** self-contained scientific data file (often single file/dataset)
- **Pros**
  - Ubiquitous in atmospheric science; portable single file
- **Cons**
  - Limited native parallelism
  - Appends often require rewriting
  - Can be slower at scale

Running a Model: Step-by-Step

1. Create a **dedicated conda environment**
2. Install **inference packages** per model docs (match CUDA/driver/framework)
3. Obtain the **checkpoint** (+ configs)
4. Write a **reproducible code** to preprocess → run → standardize → save

Code Schema (Modular)

Base all arguments on initiation date



```python
def get_ic(init_date):
    # load & preprocess initial conditions
    return ic

def standardize_forecast(fx):
    # conform to shared schema (e.g., WeatherBenchX)
    return fx

def run_model(init_date):
    ic = get_ic(init_date)
    fx = model(ic)                  # framework call (PyTorch/JAX)
    fx = standardize_forecast(fx)
    fx.to_netcdf("save_path")       # or write to Zarr

def main(dates):
    for d in dates:
        run_model(d)
```

Solutions to Common Challenges

Make Life Easier: Standardize Outputs

- Models output varying shapes/metadata → **normalize** to WeatherBench conventions:
  - (time, prediction_timedelta, latitude, longitude) (+ level, member when needed)
- Benefits
  - Consistent qualitative checks
  - Plug-and-play WeatherBenchX metrics

Systematic Evaluation

- Use **WeatherBenchX** for base metrics
  - Avoid hand-rolled metric errors
  - Compare across many initialization dates and models
- Automate across inits/members for robust, repeatable scoring

Avoid the queue: use ARCO ERA5 instead of CDS ERA5

- xr.open_zarr('gs://gcp-public-data-arco-era5/ar/full_37-1h-0p25deg-chunk-1.zarr-v3’)
- Source cloud-based zarr datasets wherever possible (internet permitting)

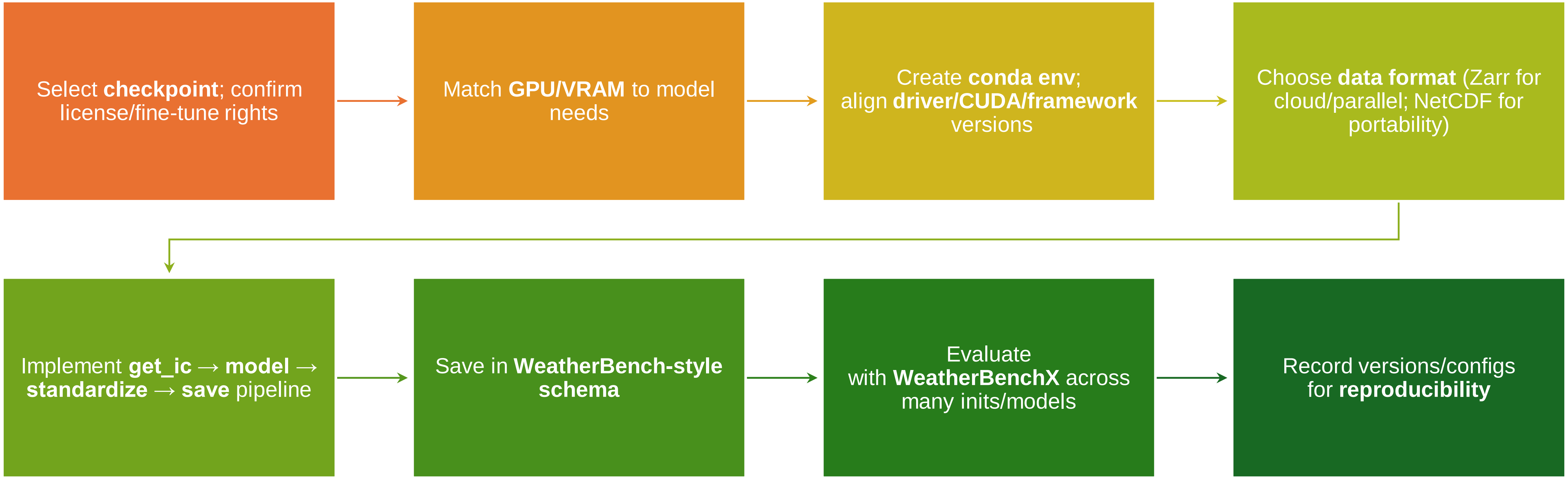
Accessibility & Licensing

- **Weights are required** to run inference at the very least
- If weights aren’t public → **retraining** is the only path (needs training code + compute)
- Retrained models may differ from published results

Case Studies

Pangu & FuXi	Checkpoints in **ONNX** → inference-only; further training requires full retrain and public training code (often unavailable)
FuXi-S2S	Weights “available upon request”; not generally accessible; requests not guaranteed
NeuralGCM	Public inference/training/fine-tuning code + pretrained checkpoints Documentation is limited → higher lift to build robust pipelines

One-Page Checklist


Select **checkpoint**; confirm license/fine-tune rights
Match **GPU/VRAM** to model needs
Create **conda env**; align **driver/CUDA/framework** versions
Choose **data format** (Zarr for cloud/parallel; NetCDF for portability)
Implement **get_ic → model → standardize → save** pipeline
Save in **WeatherBench-style schema**
Evaluate with **WeatherBenchX** across many inits/models
Record versions/configs for **reproducibility**

Wrap-Up

Understand	Understand the pieces → provision the appropriate hardware/software
Standardiz e	Standardize outputs → unlock fair, automated evaluation
Check	Check licensing/availability early → determines feasibility

Questions to Ask

- Which lead times matter most to our use-case (nowcasting, 0-72h, medium range, S2S), and which AI models align with those?
- For our pipelines (GRIB2/NetCDF/Zarr), which format is most practical for ingest + analysis + downstream services?
- How many initialization dates and seasons do we need before we’d trust a model for a limited operational trial?
- How do we evaluate AI postprocessing vs. raw AI fields–do we verify at observation points, gridded analyses, or both?
- What upskilling do forecasters and HPC/DevOps staff need (conda/CUDA basics, interpretation of AI uncertainty)?

Set Up Demo 2: Running AIFS Locally

1. Open VSCode
2. In the top left, click “File” □ ”Open Folder”
3. Navigate into the “training-program” directory and open the directory
4. Navigate into the ”demo_2” directory, you should see directories that say “support” and ”weights”
5. Then click the “select folder” button to open the “demo_2” folder in VSCode
6. Wait 5 seconds, then click the “Reopen in Container” pop-up
7. Once the container is open, click “aifs_final.ipynb” to open the notebook
8. Run the first cell (you may have to select the “ai_models” environment from the dropdown menu when prompted)
9. Read through the notebook and run the cells as you go
10. When you get to section 5, use any date within 2023 or 2024, and then run the cell to generate the forecast

Demo 2: AIFS Running Locally
Discussion Questions

- How fast does it take to generate one forecast?
- Which regions does AIFS appear to do well for temperature? For precipitation?
- Qualitatively, how does the forecast skill change with lead time?
  - Are there any regional differences with this change?
- If you compare with your neighbor, which date did they choose, and do you notice any qualitative differences in forecast skill at the same lead time?

How to pull from GitHub

1. Open VSCode
2. In the top left, click “File” □ ”Open Folder”
3. Navigate into the “training-program” directory and open the directory
4. Then click the “select folder” button to open the “training-program” folder in VSCode
5. In the lower left corner, ensure the text reads “main” or “main*”
6. Next to main, click the circular arrows
7. If there are any pop-up prompts, confirm you want to sync the repo
   a. If you are unsure about the prompt, feel free to inquire with the team
8. This will synchronize the repository and ensure everything is up to date
9. After this is completed, we can proceed to demo4

Set Up Demo 4: Using WeatherBenchX

1. Open VSCode
2. In the top left, click “File” □ ”Open Folder”
3. Navigate into the “training-program” directory and open the directory
4. Navigate into the ”demo_4_5” directory
5. Then click the “Select Folder” button to open the “demo_4_5 ” folder in VSCode
6. Click the “Reopen in Container” pop-up
7. Once the container is open, click “demo4_WBX.ipynb” to open the notebook
8. Run the first cell (you may have to select the “benchmarks_env” environment from the dropdown menu when prompted)
9. Read through the notebook and run the cells as you go