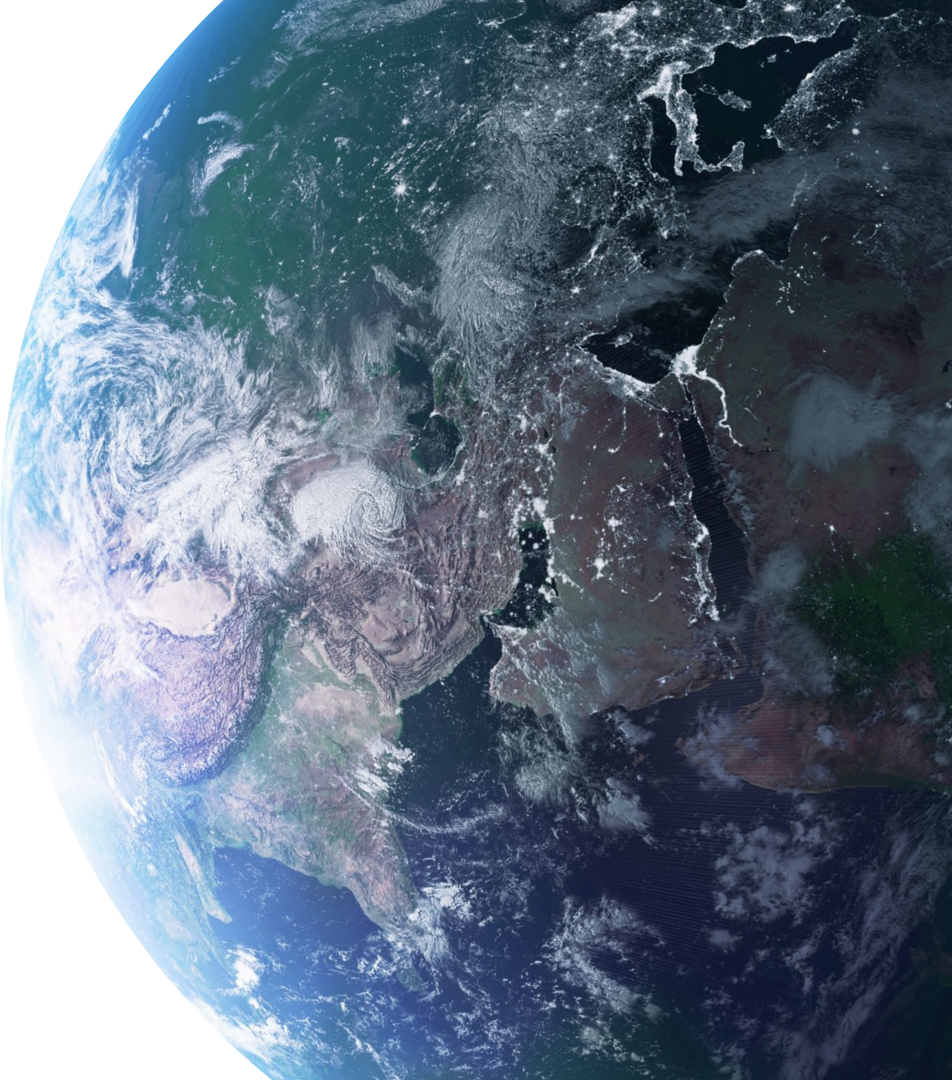
Tomorrow.io

Luke Conibear (luke.conibear@tomorrow.io), Ashley E. Payne, Allison Reed Harris, Tyler McCandless, Steven Brey, Kushal Keshavamurthy, Maxfield E. Green, Stylianos Flampouris, Luke Peffers

Operationalizing a Machine Learning Approach to Post-Processing High Resolution NWP Forecasts

February 2024

Session J15C - Towards Operationalizing AI/ML Weather Forecast and Decision Support Products II



Problem

Science

The content of the work

- Talk: Data, Ashley Payne, 29th January 2024, 2:00 PM, 337.
- Poster: Science, Ashley Payne, 29th January 2024, 3:00 PM, Hall E (100 Level).
- Paper (pre-print): An operational machine learning system that post-processes high-resolution, deterministic weather forecasts to produce short to medium-range probabilistic weather forecasts.

Research

Manage data and software

- Data management
  - Requirements for data, backups, security, handling, sharing, etc.
- Software management
  - Version control, environments, etc.
  - Commonly find a Jupyter notebook in a GitHub repository with a conda environment.
- Software licensing
  - Licence type, funder requirements, etc.

Operations

Cost-effective deployment

- Code standards
- Tests
- Experiment tracking
- Documentation
- Containers
- Infrastructure
- Continuous integration
- Deployment (continuous)
- Monitoring
- Performance
- Cost

Phases

1. Initial ML exploration

Manual

- Code standards
- Tests
- Experiment tracking
- Documentation

Code standards

1. **Improve consistency, readability, and maintainability.**
2. **Focus time on decisions need to make.**
3. **Reduce unneeded complexity, manual steps, and technical debt.**

- Refactor to clean and simple code without new functionality.
    - e.g., from notebooks to modular scripts, remove dispensibles, meaningful naming, simplify.
    - 📗 Refactoring Guru (2023), *Refactoring.*
    - 📗 Martin (2008), *Clean Code.*
- Lint (static analysis) to find errors and bugs.
    - 🔧Ruff (VSCode extension)
- Format to make code easier to read and understand.
    - 🔧 Ruff Formatter
    - 📗 *Google Python Style Guide*
- Type checking: 🔧 mypy
- Metadata standards to ensure consistent for all weather data.
    - 📗 *Climate and Forecast (CF) Metadata Conventions*
- Templates to only expose decisions need to make.
    - 🔧 Cookiecutter
- Custom libraries for small, focused, and reused code.
    - 🔧 Private PyPI on JFrog

Tests

1. **Protect against bugs and enable fast feedback.**
2. **Improve maintainability and refactoring by testing output and not binding to implementation details.**

- Unit tests for individual components.
  - 🔧 pytest
    - Can use fixtures to share test data across different tests.
    - Can use parametrize to check multiple cases.
  - 📗 Khorikov (2020), *Unit Testing Principles, Practices, and Patterns.*
- Property-based tests to find edge cases.
  - 🔧 Hypothesis
- Mock tests with dummy objects.
  - e.g., network access, resource-intensive, hardware-limitations.
  - 🔧 unittest.mock
- Regression tests for any new bugs found.
- Integration tests to validate the system end-to-end.
- ML-specific tests pre, during and post training.
  - e.g., data (distributions, leakage, ranges, types, missing), model shape, weights update, loss reduces, overfit single batch, metrics above threshold.
  - 📗 Jordan (2020), *Effective testing for machine learning systems.*

Experiment tracking

1. **Reproducible record of experiments.**
2. **Enable a fast prototyping loop.**

- Measure and monitor metrics.
    - 🔧 W&B
- Collect user-defined arguments in configs.
    - 🔧 ml_collections (type safe, extendable).
    - Track lineage by versioning everything (e.g., data, software, system).
        - 📗 Semantic versioning.
- Pass user-defined configs to scripts as jobs.
    - 🔧 Abseil (type safe).
    - e.g., Flax (neural network library) config, script, variants.

Documentation

1. **Explanations not understandable from the code e.g., why, architecture.**
2. **Communication and collaboration between developers to maintain, reuse, and extend systems.**
3. **Onboard new developers.**

- 📗 Comments
- Docstrings
  - 📗 Google style
- Other e.g., processes, setups, guides, etc.
  - 🔧 Confluence

Phases

1. Initial ML exploration

Manual

- Code standards
- Tests
- Experiment tracking
- Documentation

2. Repeatable and reliable

Automated

- Phase 1
- Containers
- Infrastructure
- Continuous integration
- Deployment (continuous)
- Monitoring

Containers

1. **Increase consistency, reproducibility, and portability of isolated environments.**
2. **Speed up onboarding, development, and deployment by removing environment issues (e.g., "*it works on my laptop*", Apple chips, conda issues).**

- Small images and fast builds.
  - 🔧 Docker
  - 📗 Turner-Trauring (2023), *Python on Docker Production Handbook.*
- Develop in the same environment as used in continuous integration and production.
  - 🔧 DevContainers (VSCode)
- Customise pre-built images from ML platform.
  - Optimised for ML platform with drivers, CUDA, etc.

Infrastructure

1. **Provide hardware and software to develop and deploy reliably and efficiently.**

- Infrastructure-as-code to version control, automate, and scale reproducible resources.
    - 🔧 Terraform
- Fully-managed ML platform
    - e.g., Azure ML (Microsoft), SageMaker (AWS), Vertex AI (GCP).
    - SDKs: native (optimised for platform, simpler) and open-source (portable).
    - Registries: containers, models, metadata.
    - Compute: development (e.g., notebooks, instances) and jobs (e.g., clusters, accelerators).
        - Start small and incrementally adjust.
        - Memory-optimised.
        - 📗 Prasanna (2020), *Choosing the right GPU for deep learning on AWS.*
            - Considerations mostly transfer to other clouds
            - Memory (GPU, bandwidth).
            - Profile utilisation.

Continuous integration

1. **Automate merging and checking of new code changes.**

- Automatically run checks and tests e.g., formatting, linting, type checking.
  - 🔧 GitHub Actions
  - 🔧 pre-commit
- Local replication
  - 🔧 Makefiles
  - 🔧 act

Deployment (continuous)

1. **Enable deployment strategies (e.g., manual, automatic, during pull-request, A/B), pipelines, and frequent release iterations (including rollbacks).**

- Consistent solution across company
  - Our system includes PubSub (notifications), Kubernetes (compute), and cloud storage.
    - e.g., new data (automatic notification), on demand (manual send in notification), schedule (cron job).
    - Continuous testing (CT) in production (performance degradation, concept drift, data schema).
    - Combine jobs into pipeline e.g., preprocessing and inference.
    - 📗 Anderson, *Kubernetes Deconstructed.*

Monitoring

1. **Enable quick identification and resolution of errors.**
2. **Measure and monitor key performance indicators (KPIs).**

- Measure, monitor, and summarise errors.
  - 🔧 DataDog Metrics, Monitors, and Dashboards.
- Alert, track, and respond to issues.
  - 🔧 PagerDuty Services, Integrations, and Schedules.
- Document the on-call process.
- Review (post-mortem) service interruptions.
- Consider different KPIs e.g., scientific (metrics), engineering (latency, cost), operational (availability).

Phases

1. Initial ML exploration

Manual

- Code standards
- Tests
- Experiment tracking
- Documentation

2. Repeatable and reliable

Automated

- Phase 1
- Containers
- Infrastructure
- Continuous integration
- Deployment (continuous)
- Monitoring

3. Scalable

Cost-effective

- Phase 2
- Performance
- Cost

- 📗 *Best Practices for ML Engineering, Google.*
- 📗 *Breck et al., (2017), The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction.*
- 📗 *Full Stack Data Science*
- 📗 *Huyen (2022), Designing Machine Learning Systems, O'Reilly Media, Inc.*
- 📗 *Machine Learning Engineering for Production (MLOps) Specialization, Coursera, DeepLearning.AI*
- 📗 *Sculley et al., (2014), Machine Learning: The High-Interest Credit Card of Technical Debt.*
- 📗 *Sculley, et al., (2015), Hidden Technical Debt in Machine Learning Systems.*
- 📗 *Godbole et al., (2023), Deep Learning Tuning Playbook.*

Performance

1. **Optimise code *after* it is correct, tested, documented, and profiled.**
2. **Efficient resource use and time to solution.**

- 📗 Profiling analyses your code.
  - Time: 🔧 %timeit (IPython magic command), SnakeViz (visualises cProfile output).
  - Memory: 🔧 memray (live and visual).
- 📗 Data structures, algorithms, and libraries.
  - e.g., built-ins, standard/optimised libraries, data types, data precision, minimise data movement.
  - Efficient access to traditional file types (e.g., NetCDF, GRIB2) on the cloud: 🔧 kerchunk.
  - Compress to information content: 🔧 xbitinfo (xbitinfo-python).
  - Libraries that work well together: 🔧 JAX (high-performance numerical computing), Flax (neural networks), Scenic (computer vision), use as data model within xarray.
- 📗 Vectorisation (broadcasting): 🔧 JAX.
- 📗 Compile using JITs (Just-In-Time): 🔧 Numba, JAX.
- 📗 Parallelise large problems into many smaller ones and solves them simultaneously.
  - 🔧 Joblib, Dask, Kubernetes replicas (horizontal pod autoscaling, KEDA), JAX.
  - 📗 Distributed ML training: 🔧 Horovod.
- 📗 Accelerators: 🔧 Numba, JAX.
- 📗 ML data pipelines: 🔧 tf.data.

Cost

1. **Manage and optimise costs.**

- Create cost budgets, alerts, attributions, and reports.
  - Tag resources.
  - 🔧 DoIT
- Tips to reduce costs.
  - Profile and monitor what actually use.
  - Stop/scale down instances/clusters when not in use (automatically if possible).
  - Start small on a sample of data, and scale out when ready.
  - For clusters, set the minimum node count to 0.
  - Spot (preemptible) instances.
  - Checkpointing.
  - Add data retention and deletion policies.
  - Careful of data backups, excessive logging, cross-regional resources.

Results

1. Initial ML exploration

Manual

- Proof-of-concept complete (i.e., Jupyter notebooks, sample data, CPUs, split from in-house deployment system).
- Created operational prototype
  - 📗 Refactoring, CF conventions, Google style
  - 🔧Ruff, Ruff Formatter, mypy, Cookiecutter, JFrog, pytest, unittest.mock, W&B, ml_collections, Abseil

2. Repeatable and reliable

Automated

- Increased to full data set.
- Integrated with in-house deployment system including automated pipelines.
  - 🔧 Fully-managed ML platform, Docker, pre-built images, DevContainers, Terraform, GitHub Actions, pre-commit, Makefiles, PubSub, Kubernetes, cloud storage, DataDog, PagerDuty, Confluence

3. Scalable

Cost-effective

- Reduced time, cost, and file size by 90+%.
  - 🔧 SnakeViz, memray, bottleneck, polars, tf.data, mixed precision, accelerators, JIT, Joblib, KEDA, xbitinfo-python, lifecycle configs, spot training, DoIT
- Training ~1.5 hours for CONUS (~9.5 hours for Global) on 1 NVIDIA T4.
- Scalable
  - Data, projects, training.

Summary

- **Talk: Data, Ashley Payne, 29th January 2024, 2:00 PM, 337.**
- **Poster: Science, Ashley Payne, 29th January 2024, 3:00 PM, Hall E (100 Level).**
- **Paper (pre-print): Operational ML post-processes system to create probabilistic forecasts.**

- Towards operationalising ML weather products
    - Go incrementally through phases:
        - Phase 1: Initial ML exploration (manual)
        - Phase 2: Repeatable and reliable (automated)
        - Phase 3: Scalable (cost-effective)
    - Starting small, simplifying where can, only adding what is required, only exposing decisions need to make, and if in doubt *follow good software engineering practices.*

Luke Conibear (luke.conibear@tomorrow.io)