SFU SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

A MACHINE LEARNING RESEARCH TOOLKIT

Anderson de Andrade

School of Engineering Science
Simon Fraser University
Burnaby, BC, Canada

August 16, 2025

OVERVIEW

- Rationale
- Philosophy
- Architecture breakdown
- Component dive-in:
    1. Code
    2. Artifacts
    3. Deployment

RATIONALE

1. Before working on theory, might be best to produce empirical results - requires **development speed** and **correctness**
2. In absence of theoretical proof, **correctness** of empirical results takes precedence
3. Dynamic allocation of computing resources essential for **quick turnaround**
4. Recording, labeling and management of experiment artifacts is essential for result **validity** and **reproducibility**
5. Most of your research is **extensible**, so should your code

PHILOSOPHY

1. Simple over easy[1]
   - One role, one task, one concept, one dimension
   - But **not**: one instance, one operation, one function, one script
   - Often means making more things, not fewer
   - Does **not** mean: convenient, available, familiar, at hand, succinctly described
   - *"Simplicity is prerequisite for reliability"* - Edsger Dijkstra
   - *"Simplicity is the ultimate sophistication"* - Leonardo da Vinci

[1] Rich Hickey. Simple Made Easy. StrangeLoop 2011

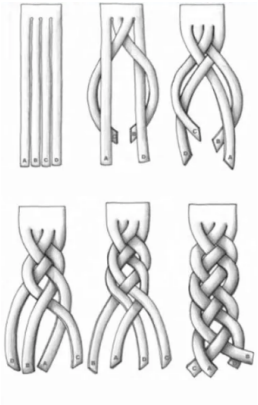
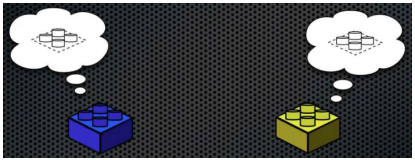


Figure 1: Which one would you rather work with?[2]

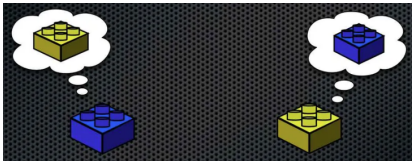
[2] Rich Hickey. Simple Made Easy. StrangeLoop 2011

2. Decoupling of concerns is imperative
    - Ease of understanding
    - Ease of change
    - Flexibility: pick what you need/like
    - Modularity and organization do not imply it but enable it
    - Libraries over frameworks: remain in control
    - ~~Convention over configuration~~ Explicit is better than implicit

PHILOSOPHY



(a) Only need to know what (interface)



(b) Need to also know how[a]

[a] Rich Hickey. Simple Made Easy. StrangeLoop 2011

3. Justified abstractions
    - To draw away, not to hide
    - About separating the **what** and **how**, also who, when, and where
    - ~~Don't repeat yourself~~ Avoid hasty abstractions
    - Keep current research your priority
    - How much time are we saving?
    - Opinion will decrease over time: perspective, not skill

PHILOSOPHY

- Extend philosophy to the tools/libraries you use: choose accordingly
- Better to reimplement/port than bring hairball to codebase
- Be able to reason about entire system

ARCHITECTURE BREAKDOWN

Type	Constructs	Tools[3]
Code	Versioning	GitHub
	Project template	Cookiecutter
	Dependencies, environment, packaging	Poetry, PyEnv
Code	Model development	PyTorch, Numba
	Training & inference boilerplate	PyTorch Lightning
	Dataset development	PyTorch Data Utilities
	Data augmentations	TorchVision, SFU Torch Library
	Argument parsing	Defopt

[3] All tools instrumented in project template

ARCHITECTURE BREAKDOWN

Type	Constructs	Tools[4]
Code	Formatting	ruff
	Type checking	basedpyright
	Tests	PyTest
Artifacts	Dataset storage	MinIO, SFU Torch Library
	Experiment artifacts	MLFlow Tracking, SFU Torch Library
	Notifications	Slack, SFU Torch Library
Deployment	Containers	Docker Images, MLFlow Projects
	Job scheduling	Slurm, Apptainer & Kubernetes
	Credentials, configuration	Ansible, Direnv

[4] All tools instrumented in project template

DIVE-IN: VERSIONING (CODE)

- Uncluttered: **avoid** commented blocks, extra files, unused implementations
- Rely on **meaningful** code versioning
- Experiments run on specific version
- OK to rewrite history outside main branch: `git-aap`
- Codebase is not your agenda: LogSeq, Obsidian
- Cannot memorize the CLI? Lazygit, VSCode, GitHub Desktop

DIVE-IN: PROJECT TEMPLATE (CODE)

- Provides:
  - Minimal project configuration using collected information
  - *Make* tasks for deployment, linting, and testing
  - Job scheduling scripts
  - Small project scaffold
- Duplication allows project independence
- Run `make project`

DIVE-IN: DEPENDENCIES, ENVIRONMENT, PACKAGING (CODE)

- *Poetry lock* enables reproducibility
- *PyEnv* manages different *Python* versions, installed under user
- Include all dependencies that you explicitly import
- Be specific about version unless used mainly by dependencies
- Treat *Python* version as a dependency
- Initialize environment `make init`
- Activate environment `$(poetry env activate)`
- Run command within environment `poetry run`
- Build package `make build`

DIVE-IN: MODEL DEVELOPMENT (CODE)

- *PyTorch's* `Module` interface enables decoupling and abstractions
- Express core logic in **simple** functions
- Group functions with interface implementations
- Keep interfaces pure
- Enable *Accelerated Linear Algebra* (XLA) with `torch.compile()`
- Custom operators with `Function`[5] and `torch.library`
- Low-level implementations with CUDA support using *Numba*[6]

[5] torch.autograd.Function
[6] See numba.pydata.org

DIVE-IN: TRAINING & INFERENCE BOILERPLATE (CODE)

- *PyTorch Lightning*[7] is a minimalist **framework**
- Provides a high-level interface for *PyTorch* models: `LightningModule`
- `Trainer` provides:
  - Gradient management
  - Dataloader management: device placement
  - Metric accumulation: supports torchmetrics[8]
  - **Callbacks**
- Callbacks are main way to manage functionality

[7] See lightning.ai/docs/pytorch/stable
[8] See lightning.ai/docs/torchmetrics/stable

DIVE-IN: DATASET DEVELOPMENT (CODE)

- *PyTorch's* `Dataset`[9] and `IterableDataset` interfaces are simple
- Manipulations: stacking, concatenation, chaining, subsets
- Different sampling methods, customizable via `Sampler`
- `FileFetcher`[10] interface implemented for ZIP, TAR and file systems
- `DataLoader`: multiprocess data loading, prefetching and memory pinning
- Control over batch collation with overloaded defaults

[9] `torch.utils.data`
[10] `sfu_torch_lib.file_fetcher`

DIVE-IN: DATA AUGMENTATIONS (CODE)

- *TorchVision's* `Compose`[11] for transformation pipelines
- Use tensors instead of *PIL* images
- Use `torch.uint8` dtype
- Use *SFU Torch Lib's* `ComposeTree`[12] for fine-grained group transformations

[11] `torchvision.transforms.v2`
[12] `sfu_torch_lib.processing`

DIVE-IN: ARGUMENT PARSING (CODE)

- `defopt`:[13] lightweight, minmal effort
- Call functions without having to recreate arguments object
- Allows configuration to be explicit
- Only documented functions can become abstractions

[13] See defopt.readthedocs.io

DIVE-IN: FORMATTING (CODE)

- Readability and consistency across projects
- `ruff`[14] written in Rust (fast)
- 800+ **configurable** rules
- `pyproject.toml` support
- Automatic error correction
- Run `make format`
- Keep consistency between editor and CLI

[14] See github.com/astral-sh/ruff

DIVE-IN: TYPE CHECKING (CODE)

- HPC is a strong case for static typing
- `basedpyright`[15] forks `pyright`
- Adds `pylance`[16] for completion, quick info, members
- `pyproject.toml` support
- Assertions enforce type
- Avoid reusing variables, simple
- `# type: ignore`
- Run `make type-check`
- Keep consistency between editor and CLI

[15] See docs.basedpyright.com

[16] See marketplace.visualstudio.com/items?itemName=ms-python.vscode-pylance

DIVE-IN: TESTING (CODE)

- No more than guardrails
- HPC has a strong case for it
- Unit test vs. functional (integration) test
- Useful assertions in `torch.testing`[17] and `numpy.testing`[18]
- Run `make test`

[17] pytorch.org/docs/stable/testing.html
[18] numpy.org/doc/stable/reference/routines.testing.html

DIVE-IN: DATASET STORAGE (ARTIFACTS)

- Centralized, decoupled and ubiquitous
- Available via Internet
- S3-compatible for extensive API support[19]
- Two-level caching via `sfu_torch_lib.io`[20]
- `aws s3` for management
- GUI `minio.multimedialabsfu.ca`

[19] On Python, use boto3
[20] See sfu_torch_lib.io

DIVE-IN: EXPERIMENT ARTIFACTS (ARTIFACTS)

- MLFLow Tracking focuses on experimentation
- Parameters are considered immutable
- Datasets are often immutable, only URL as parameter
- Runs linked to Git version
- Instrument via logger, checkpoint callback and entry function annotator
- GUI `mlflow.multimedialabsfu.ca`
- `mlflow-skinny`[21] API allows to retrieve data programmatically
- Uses our Minio cluster for model storage

[21] See pypi.org/project/mlflow-skinny

DIVE-IN: NOTIFICATIONS (ARTIFACTS)

- Slack notification on script: start, end and error
- Collects arguments from function call
- Collects and reports exception messages
- Implemented via entry function annotator[22]

[22] See sfu_torch_lib.slack

DIVE-IN: CONTAINERS (DEPLOYMENT)

- Docker images are OCI-compliant
- Compatible with: Apptainer, Kubernetes
- Embeds project's Python and dependencies
- MLFLow Project[23] for entry points
- `MLProject` included in scaffold
- Build `make training-image`
- Shared repository on DockerHub[24] with project name as label

[23] See mlflow.org/docs/latest/projects
[24] See hub.docker.com/repositories/multimedialabsfu

DIVE-IN: JOB SCHEDULING (DEPLOYMENT)

- Project commands[25] support: Slurm, Kubernetes and Apptainer
- `slurm-{launch,status,log,stop,cancel}`
- Slurm scripts support periodic restarts
- Support for argument sets
- Relaunch with `slurm-launch --run-id`

[25] See repository

DIVE-IN: CREDENTIALS & CONFIGURATION (DEPLOYMENT)

- Credentials stored in encrypted file
- Decryption key stored in OS keychain
- All credentials and configuration as environment variables
- Loaded in shell when changing to project directory: `direnv allow`
- Available in job's environment

SET UP

- Fork github.com/multimedialabsfu/research
- Follow instructions on `README.md` to install required software
- Update cookiecutter before creating a new project