`epsml` - ML Monorepo Proposal

Sep 29, 2025

Matt, Andrej, & Arjun

TL;DR: Our current codebase structure is causing compatibility issues across training and model serving. Let’s migrate to a monorepo (with subpackages), and handle all dependencies using uv and python packaging.

Current codebase structure

`github.com/EpsilonLabsInc`
→ `epsclassifiers`
→ `epsutils`
→ `epsdatasets`
→ `2d-image-encoders`
→ `reports-pipeline`
→ `ml-inference`
→ `dinov3`
→ ...

Problems:

1. **No python packaging** - In order to use code from one of the repos, you must clone the repo and add its local path to your `PYTHONPATH` environment variable. This is difficult to scale - spinning up a new VM or onboarding a new eng is painful.

2. **Outdated dependencies.** Not all repos have up-to-date `requirements.txt` files. It’s not straightforward to simply create a virtual env and run `pip install -r requirements.txt`. Similarly, some `requirements.txt` have conflicts across repos, and we have no way of easily identifying/resolving this.

3. **Large builds due to broad imports** - Production code depends on epsclassifiers, which in turn depends on epsutils. Epsutils is a massive repo

with many dependencies. In order to run classifiers in production, we would need to include the entire (potentially conflict-prone) epsutils.

Proposed codebase structure

```
github.com/EpsilonLabsInc
      → epsml
            → pyproject.toml            # Workspace root [details]
            → uv.lock                   # Single lockfile for entire repo
            → .venv/                    # Single virtual env
            → packages/
                  → epsclassifiers
                        → pyproject.toml
                        → src/epsclassifiers/
                  → 2d-image-encoders
                        → pyproject.toml
                        → src/2d-image-encoders/
                  → epsdatasets
                        → pyproject.toml
                        → src/epsdatasets/
                  → epsanalytics
                        → pyproject.toml
                        → src/epsanalytics/
                  → epstraining         # New, branched from epsutils
                        → ...
                  → epslabels           # ^ Same
                  → epsdicom            # ^ Same
                  → …                   # Any future package goes here
      → ~~epsutils                        # Deprecated, delete eventually~~
      → reports-pipeline
      → ml-inference                    # References
      → dinov3
            → pyproject.toml            # External repos as packages.
            → uv.lock
      → InternVL-3x
            → pyproject.toml            # External repos as packages.
            → uv.lock
```

Internally, uv will ensure a single set of dependencies across all packages via a single lockfile, ensuring that we have no conflicts. Any changes within the monorepo will be reflected locally whenever developing internally - no rebuilding / repackaging would be necessary for other epsml packages [example].

External codebases (outside the monorepo) can add dependencies to the subpackages directly, reducing the need to add unnecessary dependencies. They will need to reference a monorepo package using an explicit tag (e.g. v0.1.0) or commit hash [example].

How to setup/use the monorepo:

Shell

```shell
# Clone monorepo
git clone https://github.com/EpsilonLabsInc/epsml
cd epsml

# Install uv
curl -LsSf https://astral.sh/uv/install.sh | sh

# Sync dependencies with lockfile (from git).
uv sync

# Run some code...
source .venv/bin/activate
cd packages/epsclassifiers
python src/epcclassifiers/run_training.py --dinov3   # example
```

Advantages:

1. **Simple setup** - To set up a new machine / engineer, simply clone epsml, install uv, and run uv sync. Then, use the virtual environment to run your jobs.

2. **Easy dependencies for external code** - External repos (e.g. production inference code) will simply need to add a dependency of relevant packages within the monorepo, and use the monorepo lockfile to ensure a perfectly reproducible setup.

3. **Single lockfile guarantees no dependency conflicts** - uv will coordinate dependencies across all subpackages within the epsml/packages/ directory. It will choose a single set of dependency versions to ensure no cross-package

compatibility issues.

4. **Consistent standards across all ML code** - We can define Github Actions for all of epsilon_ml that will run: dependency testing, pytesting, linting & formatting.

Appendix

A. Example monorepo pyproject.toml files

Example root pyproject.toml file:

None

```
# epsml/pyproject.toml

[tool.uv.workspace]
members = [
    "packages/*",
]
```

Example subpackage pyproject.toml files:

None

```
# epsml/packages/epsdatasets/pyproject.toml
dependencies = ["pandas>=2.0"]

# epsml/packages/epstraining/pyproject.toml
dependencies = ["torch>=2.0"]

# epsml/packages/epsclassifiers/pyproject.toml
dependencies = [
    "epsdatasets",
    "epstraining",
    "torch>=2.1"
]
```

B. Example internal and external usage

Example 1: Testing changes within the monorepo (fixing a bug in epsdatasets and testing it in epsclassifiers)

Shell

```
# You're working in the epsml monorepo
cd ~/epsml/

# 1. Fix bug in epsdatasets
vim packages/epsdatasets/src/epsdatasets/loader.py
# Change line 42: return data.dropna()  # Fixed the bug!

# 2. Update epsclassifiers to use the fix
vim packages/epsclassifiers/src/epsclassifiers/training.py
# Add: data = loader.load_data()  # Uses the fixed version

# 3. Test immediately - no commits or tags needed
cd packages/epsclassifiers/
uv run python src/epsclassifiers/training.py
# ✅ Works! The bug fix is immediately visible

# 4. Run tests
uv run pytest
# ✅ All tests pass

# 5. Only NOW do you commit (once you're happy)
git add packages/epsdatasets packages/epsclassifiers
git commit -m "Fix data loading bug and update classifier"
git push

# 6. Tag a release when ready for external consumers
git tag v0.3.0
git push --tags
```

Example 2: Working outside the monorepo.

Shell

```
# You're working in ml-inference (external repo)
cd ~/ml-inference/
```

```
# Current state: ml-inference depends on epsclassifiers
cat pyproject.toml
```

```
None
# ml-inference/pyproject.toml
[project]
dependencies = ["epsclassifiers"]

[tool.uv.sources]
epsclassifiers = {
    git = "https://github.com/EpsilonLabsInc/epsml",
    subdirectory = "packages/epsclassifiers",
    tag = "v0.3.0"  # ← Updated to the new version
}
```

C. Eng notes

Arjun eng TODO:

- Git patch to move repo while maintaining the git history: https://plankenau.com/blog/post/copy-commits-separate-git-repos
- TODO: test dependencies work fine for the new setup (e.g. run inference on old setup vs new setup and ensure same outputs)
- Setup github actions for CI: run tests, run linter/formatter, run type checking, show results in PR

Order of operations

1. ~~[Matt/Andrej/Arjun] Confirm plan~~

2. ~~[Arjun] Create epsml repo~~

3. ~~[All] Ask everyone to submit existing code to their repos (main branch)~~

— ^ Sept 30 ^ —

4. ~~[Andrej] Follow blog post (to keep git history), copy in all source repos: epsclassifiers, epsdataset, epsutils, 2d-image-encoders, analytics~~

    a. ~~Then commit, without any restructuring / deleting~~

5. ~~[Andrej] Restructure - move epsutils subdirs into their own packages, delete old code, etc etc~~
    a. ~~New structure: packages/epsclassifiers/src/epsclassifiers/…~~
    b. ~~Include requirements.txt for easy uv dependencies later~~
    c. ~~Cleanup~~
    d. ~~Then commit~~

— ^ Oct 1 ^ —

6. [Matt/Arjun] Address dependencies with uv
    a. Look at requirements.txt, probably most are up-to-date but may be

7. [Arjun/Andrej] Test example classification training with DINOv3, ensure same/similar results as before

— ^ Oct 2-5 ^ —

8. [All] DICOM code conflicts

9. [Arjun] Run code formatter over all files
    a. [black python linter] or [ruff python linter]

— ^ Oct 6-7 ^ —

10. Final step, add all remaining repos when we get time
    a. 3d-image-encoders
    b. …

11. Set up Github Actions: dependency testing, pytesting, linting & formatting.