Arman Variant – Gaps vs Main_project

Notes from reviewing `novartis_datathon_2025-Arman` compared with `Main_project`.

Configuration & Orchestration

- Uses YAML-driven configs (`configs/data.yaml` , `features.yaml` , `run_defaults.yaml` , `model_*.yaml`) with a loader/merger layer (`src/utils.py` , `src/config_sweep.py`) and sweep expansion for grid/hyperparam runs; main_project hardcodes settings in `src/config.py` .
- CLI training pipeline (`src/train.py`) with argparse, experiment tracking hooks (MLflow/W&B), Optuna-based sweeps, run metadata hashing, checkpointing, and process-parallel sweeps; main_project scripts call Python functions directly without a CLI or tracking hooks.
- Reproducibility helpers: git hash capture, config snapshots, deterministic seeds, `reproduce.sh` ; main_project lacks these wrappers.

Data, Features, External Context

- Data module (`src/data.py`) validates schema/ranges, enforces no duplicates, caches panels, and normalizes meta columns; main_project data loading is simpler and lacks schema validation/caching.
- Feature pipeline (`src/features.py`) is scenario-aware with cutoff enforcement and optional blocks: target encoding, interaction pairs, sequence features, visibility/collaboration flags, seasonal detection, future generics as exogenous inputs; main_project covers core lags/rolling/competition/time but not these toggles or leakage guardrails.
- External/context ingestion stubs (`src/external_data.py` , `src/visibility_sources.py`) plus scenario simulation utilities (`src/scenario_analysis.py`) are present; main_project has no external-context hooks or shock simulators.
- Graph utilities (`src/graph_utils.py`) to build drug graphs/embeddings and `src/sequence_builder.py` for sequence datasets and scalers used by deep/graph models; main_project has no graph or sequence builders.

Models

- Additional model families beyond boosting/hybrid/arihow: CatBoost (`src/models/cat_model.py`), linear models (ridge/lasso/elasticnet/huber), PyTorch MLP (`src/models/nn.py`), CNN-LSTM (`src/models/cnn_lstm.py`), KG-GCN-LSTM (`src/models/kg_gcn_lstm.py` + `src/models/gcn_layers.py`), and an LSTM config (`configs/model_lstm.yaml`); main_project only ships baseline/boosting/hybrid/ARHOW.
- Model registry & BaseModel interface (`src/models/__init__.py` , `src/models/base.py`) standardize `fit/predict/save/load` and alias mapping; main_project uses ad-hoc classes without a common factory.
- Ensembles include averaging/weighted/stacking/blending (`src/models/ensemble.py`) with config-driven membership; main_project only implements a simple blender.

Evaluation, Validation, Inference

- Validation utilities (`src/validation.py`) provide series-level splits (GroupKFold/Stratified), per-fold metadata, and OOF handling; main_project splits inside training scripts without dedicated module.
- Metric module (`src/evaluate.py`) mirrors official Metric1/Metric2 with per-series logging, aux-file creation, and unified metric records; main_project metrics live in `src/metric_calculation.py` without per-series logging helpers.
- Inference script (`src/inference.py`) loads saved model artifacts, rebuilds features, and writes submissions; main_project couples inference to training scripts.

Testing & Project Hygiene

- Test suite (`tests/test_smoke.py` , `pytest.ini`) checks imports, config alignment, data loading, feature leakage guards, and model training; main_project has no automated tests.
- Environment scaffolding (`env/requirements.txt` , `env/environment.yml`), `CONTRIBUTING.md` , and licensing/README badges are maintained; main_project lacks env files and contribution docs.

Net: Missing in Main_project

- YAML config system + sweep runner
- External/visibility/collaboration features and scenario simulations
- Graph/sequence builders and deep models (CNN-LSTM, KG-GCN-LSTM) plus CatBoost/linear/NN models
- Experiment tracking hooks, CLI training/inference, and Optuna sweeps
- Dedicated validation/evaluation modules and test suite