☑ Training Strategy Refactor – To-Do List for Copilot Agent

Goal: **Keep the current separate training for Scenario 1 & 2**, but:

- Fix validation / evaluation so they work correctly with separate or unified logic.
- Add a **config toggle** to switch between:
  - `train_mode = "separate"` → two pipelines (S1 & S2)
  - `train_mode = "unified"` → single global pipeline

This file is designed as a step-by-step task list for an AI agent (e.g. Copilot).

---

1. Add a `train_mode` Toggle in Config

- ☐ Open the main configuration file (e.g. `config.yaml` or `config.py`).
- ☐ Add a new top-level key:
  - ☐ `train_mode: "separate"` # options: "separate", "unified"`
- ☐ If using nested config sections, ensure both training scripts can access:
  - ☐ `config.train_mode`
- ☐ Document the allowed values and behavior in comments or README.

Example YAML snippet:

```
training:
  train_mode: "separate"   # "separate" or "unified"
```

---

2. Centralize Scenario Definitions

- ☐ Create a central module (e.g. `src/config/scenarios.py`) to define:
  - ☐ Scenario 1:
    - ☐ horizon: months 0–23
    - ☐ available features: pre-LOE only
  - ☐ Scenario 2:
    - ☐ horizon: months 6–23
    - ☐ available features: pre-LOE + months 0–5 info
- ☐ Expose helper functions:
  - ☐ `get_scenario_definition(name: str) -> dict`
  - ☐ `is_month_in_scenario(month, scenario_name)`
- ☐ Update training code to import scenario definitions from this module instead of hard-coding ranges in multiple places.

---

3. Keep Current Separate Training Pipelines (S1 & S2)

- [ ] Identify the **current** S1 training script/function (e.g. `train_scenario1()`):
  - [ ] Confirm it:
    - [ ] Uses pre-LOE features only.
    - [ ] Trains on months 0–23.
    - [ ] Produces S1 predictions for months 0–23.
- [ ] Identify the **current** S2 training script/function (e.g. `train_scenario2()`):
  - [ ] Confirm it:
    - [ ] Uses pre-LOE + months 0–5 features.
    - [ ] Trains on months 6–23.
    - [ ] Produces S2 predictions for months 6–23.
- [ ] Wrap them in a higher-level function `train_separate(config)`:
  - [ ] Calls `train_scenario1(config)`.
  - [ ] Calls `train_scenario2(config)`.
  - [ ] Stores models, metrics, and artifacts in separate subfolders:
    - [ ] `models/scenario1/`
    - [ ] `models/scenario2/`
    - [ ] `metrics/scenario1/`
    - [ ] `metrics/scenario2/`

---

4. Implement a Unified Training Pipeline (Single Model)

- [ ] Add a new function `train_unified(config)`:
  - [ ] Builds a **single** multi-horizon model:
    - [ ] Training rows: all months 0–23 for all brands.
    - [ ] Features include `months_postgx` and time-window flags (early / mid / late).
  - [ ] Uses a unified feature pipeline that can support both:
    - [ ] Scenario 1 (no early post-LOE info at origin).
    - [ ] Scenario 2 (has early post-LOE summary at origin).
- [ ] Add a `scenario_flag` feature:
  - [ ] `scenario_flag = 0` for Scenario-1-like rows.
  - [ ] `scenario_flag = 1` for Scenario-2-like rows.
- [ ] In training, create two types of samples:
  - [ ] S1-style: origin at month 0, features limited to pre-LOE info.
  - [ ] S2-style: origin at month 6, features include summary of months 0–5.
- [ ] Concatenate S1-style and S2-style training rows into a single dataset.
- [ ] Train **one global model** that can handle both (`scenario_flag` differentiates regimes).

---

5. Connect `train_mode` to the Main Entry Point

- [ ] In the main training entry script (e.g. `train.py`):

    - [ ] Load config.
    - [ ] Read `config.training.train_mode` (or equivalent).
    - [ ] Use conditional logic:

```
if config.training.train_mode == "separate":
    train_separate(config)
elif config.training.train_mode == "unified":
    train_unified(config)
else:
    raise ValueError(f"Unknown train_mode:
{config.training.train_mode}")
```

- [ ] Ensure that all CLI entry points or notebooks honor this setting.

---

6. Fix / Unify Validation & Metric Code

6.1 Centralize Metric Computation

- [ ] Create a module `src/metrics/loe_metrics.py` with functions:
    - [ ] `compute_metric_scenario1(y_true, y_pred, avg_j, buckets)`
    - [ ] `compute_metric_scenario2(y_true, y_pred, avg_j, buckets)`
- [ ] Move any scattered metric code into these functions.
- [ ] Ensure the functions:
    - [ ] Use the official formulas for PE (S1 and S2).
    - [ ] Apply Bucket 1 and Bucket 2 weights correctly.
- [ ] Add unit tests for both metrics using small synthetic examples.

6.2 Validation for Separate Training

- [ ] In **separate mode** (`train_mode = "separate"`):
    - [ ] For Scenario 1:
        - [ ] Perform CV using only S1 pipeline and S1 metric.
    - [ ] For Scenario 2:
        - [ ] Perform CV using only S2 pipeline and S2 metric.
- [ ] Ensure validation outputs are stored as:
    - [ ] `metrics/val_scenario1.json`
    - [ ] `metrics/val_scenario2.json`

6.3 Validation for Unified Training

- [ ] In **unified mode** (`train_mode = "unified"`):
    - [ ] Use **GroupKFold by brand** for CV.
    - [ ] For each fold:
        - [ ] Generate predictions for all brands/months 0–23.

        - [ ] Compute:
            - [ ] S1 metric on that fold.
            - [ ] S2 metric on that fold.
    - [ ] Aggregate metrics across folds:
        - [ ] `mean_PE_s1`
        - [ ] `mean_PE_s2`
        - [ ] A combined score, e.g. `0.5 * PE_s1 + 0.5 * PE_s2`.
- [ ] Save unified validation outputs as:
    - [ ] `metrics/val_unified.json` (including both S1 and S2 metrics).

---

7. Ensure Feature Pipelines Behave Under Both Modes

- [ ] Identify the shared **feature engineering** module (e.g. `src/features/build_features.py`).
- [ ] Add support for:
    - [ ] `scenario_flag` feature.
    - [ ] Optional early post-LOE summary features (mean 0–5, slope 0–5, last value at 5, etc.).
- [ ] In **separate mode**:
    - [ ] For Scenario 1:
        - [ ] Disable early-post features (fill with 0 or NaN).
    - [ ] For Scenario 2:
        - [ ] Enable early-post features.
- [ ] In **unified mode**:
    - [ ] Always build both S1-style and S2-style rows.
    - [ ] Ensure:
        - [ ] S1-style rows have `scenario_flag = 0`, early-post features disabled.
        - [ ] S2-style rows have `scenario_flag = 1`, early-post features computed.

---

8. Model Saving & Loading Conventions

- [ ] For **separate mode**:
    - [ ] Save models as:
        - [ ] `models/scenario1/model.pkl`
        - [ ] `models/scenario2/model.pkl`
- [ ] For **unified mode**:
    - [ ] Save model as:
        - [ ] `models/unified/model.pkl`
- [ ] Adjust prediction scripts to:
    - [ ] Load the correct model depending on `train_mode`.
    - [ ] Produce predictions sliced appropriately:
        - [ ] S1: months 0–23.
        - [ ] S2: months 6–23.

---

9. CLI / Notebook Documentation

- [ ] Update `README.md` or `docs/training.md` to explain:

    - [ ] The purpose of `train_mode`.
    - [ ] Differences between `separate` and `unified` modes.
    - [ ] How to run each mode, e.g.:
        - [ ] `python train.py --train_mode separate`
        - [ ] `python train.py --train_mode unified`
- [ ] Provide example commands for:
    - [ ] Training.
    - [ ] Validation only.
    - [ ] Generating S1 and S2 submissions from each mode.

---

10. Sanity Checks

- [ ] In both `separate` and `unified` modes:
    - [ ] Confirm that S1 predictions always have months 0–23 per brand.
    - [ ] Confirm that S2 predictions always have months 6–23 per brand.
    - [ ] Run a small smoke test with 2–3 brands to ensure the pipelines don't mix up horizons.
- [ ] Verify that validation metrics are consistent between:
    - [ ] Legacy (old) separate pipelines.
    - [ ] New refactored separate pipelines.
- [ ] Confirm that unified mode produces comparable or better metrics when hyperparameters are tuned.