# Paper 1: Wasserstein Distributionally Robust Optimization (DRO) Module

#### **Formulation**

The Wasserstein Distributionally Robust Optimization (DRO) module penalizes portfolio updates based on possible distribution shifts measured using the Wasserstein distance. The penalty is formulated as:

Penalty = 
$$\rho \|\boldsymbol{\mu}_p - \boldsymbol{\mu}_o\|_2$$

where:

- $\mu_p$ : mean vector of perturbed (stress) scenario returns.
- $\mu_o$ : historical mean return vector.
- $\rho$ : robustness parameter controlling the strength of the penalty.

### Why this Works

Including this penalty discourages overfitting to historical return distributions and hedges against model misspecification. It enforces stability by accounting for potential transport costs when distributions shift.

## Algorithm Steps

- 1. Compute historical mean vector  $\mu_o$ .
- 2. Generate perturbed means  $\mu_p$  by adding noise or stress perturbations.
- 3. Calculate Wasserstein penalty as  $\rho \| \boldsymbol{\mu}_p \boldsymbol{\mu}_o \|_2$ .
- 4. Update weights by scaling them with (1 Penalty).
- 5. Normalize updated weights to ensure they sum to 1.

# Code Equivalent

```
def dro_wasserstein_penalty(weights, perturbed_mean, original_mean, rho=0.1):
    distance = np.linalg.norm(perturbed_mean - original_mean)
    penalty = rho * distance
    return penalty

# Example usage:
penalty = dro_wasserstein_penalty(weights, perturbed_mean, mean_ret, rho=0.1)
adjusted_weights = weights * (1 - penalty)
adjusted_weights = np.maximum(adjusted_weights, 1e-5)
adjusted_weights /= np.sum(adjusted_weights)
```

### Summary

- Strengthens out-of-sample robustness.
- Proven in convex robust optimization frameworks.
- Simple yet powerful tool for defensive portfolio design.