INTER IIT
TECH MEET 14.0

Client Centric WiFi RRM

INTER IIT TECH MEET 14.0

Detailed Design Document - Causal Inference Engine

Team 33

ARISTA

Contents

1 System Overview

The Causal Reward Engine is designed to validate Radio Resource Management (RRM) actions by computing causally-validated rewards for Reinforcement Learning (RL) agents. Unlike traditional correlation-based metrics, this system employs rigorous causal inference methods to isolate the true effect of RRM interventions on network Quality of Experience (QoE).

1.1 Key Objectives

- **Replace GNN predictions** with real causal rewards during deployment
- **Validate counterfactual reasoning** by comparing treatment vs control groups
- **Handle small sample sizes** through model persistence and progressive learning
- **Maintain RL compatibility** by matching existing reward structures
- **Provide statistical guarantees** through significance testing and confidence intervals

1.2 Architecture Overview

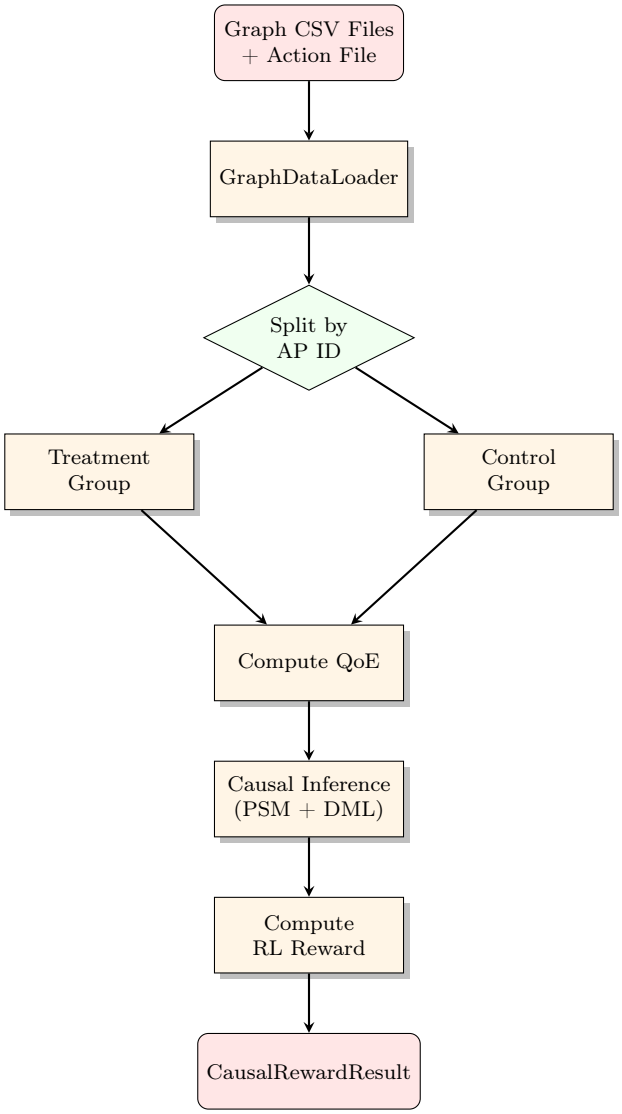

Figure 1: Causal Reward Engine Architecture

2 Data Structures

2.1 Configuration Structure

The **CausalConfig** class encapsulates all configuration parameters for the causal inference engine.

Table 1: CausalConfig Field Specifications

Field Name	Type	Default	Description
`graph_data_dir`	str	-	Directory containing graph CSV files
`action_file_path`	str	-	Path to action file (apply/skip labels)
`min_graphs_required`	int	10	Minimum graphs for valid analysis
`max_graphs_to_load`	int	20	Maximum graphs to process
`model_cache_dir`	str	-	Directory for model persistence
`enable_model_persistence`	bool	True	Enable progressive learning
`min_samples_for_training`	int	30	Threshold for model training
`psm_caliper`	float	0.15	Maximum propensity score distance
`psm_k_neighbors`	int	3	Number of nearest neighbors for matching
`significance_level`	float	0.05	Statistical significance threshold
`min_sample_size_per_group`	int	5	Minimum samples per group
`w_penalty`	float	3.0	Weight for QoE decrease (RL agent)
`w_uplift`	float	1.0	Weight for QoE increase (RL agent)
`w_state`	float	0.5	State bias weight (RL agent)
`reward_clip_min`	float	-10.0	Minimum reward value
`reward_clip_max`	float	10.0	Maximum reward value
`retry_penalty_weight`	float	10.0	QoE penalty for retries
`latency_penalty_weight`	float	0.1	QoE penalty for latency

2.2 Result Structure

The **CausalRewardResult** class contains the complete output of causal reward computation, organized into four categories.

2.2.1 Primary Outputs

Table 2: CausalRewardResult - Primary Outputs

Field Name	Type	Description
`causal_reward`	float	Final reward for RL agent (clipped to $[-10, 10]$)
`causal_effect`	float	Estimated treatment effect on QoE

2.2.2 Validation Metrics

Table 3: CausalRewardResult - Validation Metrics

Field Name	Type	Description
`is_significant`	bool	Statistical significance ($p < 0.05$)
`p_value`	float	Two-tailed p-value from t-test
`confidence_interval`	(float, float)	95% CI for effect

2.2.3 Method-Specific Estimates

Table 4: CausalRewardResult - Method-Specific Estimates

Field Name	Type	Description
`psm_effect`	Optional[float]	Effect from PSM method
`dml_effect`	Optional[float]	Effect from DML method

2.2.4 Data Quality Metrics

Table 5: CausalRewardResult - Data Quality Metrics

Field Name	Type	Description
`n_treatment`	int	Number of treatment units
`n_control`	int	Number of control units
`balance_score`	float	Covariate balance quality $\in [0, 1]$
`qoe_treatment_mean`	float	Average QoE in treatment group
`qoe_control_mean`	float	Average QoE in control group
`observed_delta`	float	Raw difference (before causal adjustment)

2.2.5 Model Information

Table 6: CausalRewardResult - Model Information

Field Name	Type	Description
`used_cached_models`	bool	Whether cached models were used
`model_sample_count`	int	Number of samples in cached models

3 Input Data Specification

3.1 Action File Format

The action file determines treatment/control group assignment.

Table 7: Action File Schema (`approved_changes.csv`)

Column	Type	Description
`ap_id`	int	Unique Access Point identifier
`channel_width`	int	Channel width in MHz (e.g., 20, 40, 80)
`tx_power`	float	Transmit power in dBm
`obss_pd`	float	OBSS-PD threshold in dBm
`action`	str	`'apply'` (treatment) or `'skip'` (control)

Example:

Listing 1: Sample Action File

```
ap_id,channel_width,tx_power,obss_pd,action
4,80,3,-52,apply
6,20,18,-52,apply
2,80,3,-52,apply
5,20,23,-82,skip
1,20,23,-82,skip
```

3.2 Graph Snapshot Files

Graph snapshots capture network state at multiple timestamps.

3.2.1 Directory Structure

Listing 2: Expected Directory Layout

```
causal_data/graphs/
    interference_graph_nodes/
        interference_graph_nodes_t60.0s.csv
        interference_graph_nodes_t150.0s.csv
        ...
    interference_graph_edges/
        interference_graph_edges_t60.0s.csv
        interference_graph_edges_t150.0s.csv
        ...
```

3.2.2 Node Features Schema

Table 8: Node CSV Schema (AP-level features)

Column	Type	Description
`ap_id`	int	Access Point identifier
`channel`	int	Operating channel number
`power_dbm`	float	Transmit power in dBm
`width_mhz`	int	Channel width in MHz
`obss_pd_dbm`	float	OBSS-PD threshold
`client_count`	int	Number of associated clients
`airtime_util`	float	Airtime utilization $\in [0, 1]$
`tx_bytes`	int	Transmitted bytes
`rx_bytes`	int	Received bytes
`avg_throughput_mbps`	float	Average throughput in Mbps
`avg_client_rssi`	float	Average client RSSI in dBm
`edge_client_count`	int	Number of edge clients (low RSSI)
`client_rssi_p90`	float	90th percentile client RSSI
`client_rssi_p10`	float	10th percentile client RSSI

3.2.3 Edge Features Schema

Table 9: Edge CSV Schema (Inter-AP relationships)

Column	Type	Description
source_ap_id	int	Source AP identifier
dest_ap_id	int	Destination AP identifier
coupling_dbm	float	Signal coupling strength
channel_overlap	float	Channel overlap fraction $\in [0, 1]$
interference_power_dbm	float	Interference power in dBm
is_cochannel	bool	True if same channel
channel_separation	int	Channel number difference
roaming_client_count	int	Clients roaming between APs
spectral_overlap	float	Spectral overlap metric
interference_power_linear	float	Linear interference power
hidden_node_proxy	float	Hidden node indicator
power_delta_dbm	float	Power difference between APs

4 Problem Formulation

To optimize WiFi Radio Resource Management (RRM), it is insufficient to simply observe correlations between actions (e.g., changing channel width) and network metrics (e.g., throughput). Observational data is often confounded by environmental factors such as user density or interference.

This module formally defines the causal inference framework used to isolate the true effect of RRM interventions. Our goal is to estimate the counterfactual: *what would have happened to the network performance had the RRM action not been taken?*

4.1 Causal Estimand

We define the fundamental variables required to estimate the causal effect. In our context, a "unit" i represents a specific Access Point (AP) or a client session.

Notation:

- Y_{1i}: Potential outcome if unit i receives treatment (e.g., new power level)
- Y_{0i}: Potential outcome if unit i receives control (e.g., existing power level)
- $T_i \in \{0, 1\}$: Treatment indicator
- X_i: Vector of observed covariates (e.g., RSSI, noise floor, client count)
- $Y_i = T_i Y_{1i} + (1 - T_i) Y_{0i}$: Observed outcome

Our primary objective is to estimate the effect on those units that actually received the intervention:

$$\boxed{\tau_{ATT} = E[Y_{1i} - Y_{0i} \mid T_i = 1]} \tag{1}$$

4.2 Identification Assumptions

Causal inference from observational data requires strict assumptions to ensure that the estimated effect is not driven by selection bias. We invoke the following three standard assumptions:

1. **Unconfoundedness/Ignorability**:
$$(Y_{1i}, Y_{0i}) \perp T_i \mid X_i$$
Treatment assignment is independent of potential outcomes conditional on covariates. This implies we have measured all relevant confounders affecting both RRM decisions and network performance.

2. **Common Support/Overlap**:
$$0 < P(T_i = 1 \mid X_i) < 1 \quad \forall X_i$$
Both treatment and control units exist at all covariate values, ensuring valid comparisons can be made across the state space.

3. **SUTVA (Stable Unit Treatment Value Assumption)**:
   - No interference between units (an action on AP A does not implicitly affect AP B)
   - Single version of treatment

5 QoE Computation Algorithm

Before performing causal inference, we must compute a Quality of Experience (QoE) metric for each AP. This metric combines throughput with penalties for retries and latency, matching the RL agent's reward structure.

5.1 QoE Formula

$$\boxed{\text{QoE}_i = \log(T_i + 1) - w_r \cdot R_i - w_\ell \cdot L_i} \tag{2}$$

where:

- T_i: Average throughput (Mbps) for AP i
- R_i: Retry rate estimate for AP i
- L_i: Latency estimate (ms) for AP i
- $w_r = 10.0$: Retry penalty weight
- $w_\ell = 0.1$: Latency penalty weight

5.2 Retry Rate Estimation

Since graph CSV files do not directly contain retry rate, we estimate it from interference and airtime utilization:

$$\boxed{R_i = 0.02 + 0.08 \cdot \frac{I_i + 90}{30} \cdot A_i} \tag{3}$$

where:

- I_i: Interference power (dBm), normalized from $[-90, -60]$ range
- A_i: Airtime utilization $\in [0, 1]$

Rationale: High interference combined with high airtime leads to more packet collisions and retransmissions.

5.3 Latency Estimation

$$\boxed{L_i = 10 + 20 \cdot A_i + 0.5 \cdot C_i} \tag{4}$$

where:

- A_i: Airtime utilization $\in [0, 1]$
- C_i: Client count

Rationale: High airtime and many clients increase queueing delays and contention.

5.4 Algorithm: Compute QoE per AP

Algorithm 1 Compute QoE for All APs

Input: DataFrame $\mathcal{D}$ with columns: `ap_id`, `avg_throughput_mbps`, `client_count`, `airtime_util`, `interference_power_dbm`
Output: DataFrame $\mathcal{D}$ with additional column: `qoe_metric`

$w_r \leftarrow 10.0$ ▷ Retry penalty weight
$w_\ell \leftarrow 0.1$ ▷ Latency penalty weight
for each row i in $\mathcal{D}$ **do**
  $T_i \leftarrow \mathcal{D}[i]$.`avg_throughput_mbps`
  $A_i \leftarrow \mathcal{D}[i]$.`airtime_util`
  $C_i \leftarrow \mathcal{D}[i]$.`client_count`
  $I_i \leftarrow \mathcal{D}[i]$.`interference_power_dbm` ▷ Fill NaN with -80
  ▷ Estimate retry rate
  $I_{\text{norm}} \leftarrow (I_i + 90)/30$ ▷ Normalize to $[0, 1]$
  $R_i \leftarrow 0.02 + 0.08 \cdot I_{\text{norm}} \cdot A_i$
  ▷ Estimate latency
  $L_i \leftarrow 10 + 20 \cdot A_i + 0.5 \cdot C_i$
  ▷ Compute QoE
  $\text{QoE}_i \leftarrow \log(T_i + 1) - w_r \cdot R_i - w_\ell \cdot L_i$
  $\mathcal{D}[i]$.`qoe_metric` $\leftarrow \text{QoE}_i$
end for
return $\mathcal{D}$

6 Method 1: Propensity Score Matching (PSM)

Propensity Score Matching creates a "quasi-randomized" experiment from observational data. By pairing treated APs with control APs that have similar probabilities of receiving the treatment, we reduce selection bias caused by the inherent policy of the legacy RRM system.

6.1 Theory

The **propensity score** is the probability of treatment given covariates. It serves as a dimensionality reduction technique, collapsing multidimensional covariates X into a single scalar.

$$\boxed{e(X_i) = P(T_i = 1 \mid X_i)} \tag{5}$$

Rosenbaum-Rubin Theorem: If unconfoundedness holds given X, it also holds given $e(X)$:

$$(Y_{1i}, Y_{0i}) \perp T_i \mid e(X_i)$$

6.2 Algorithm: Propensity Score Matching

Algorithm 2 Propensity Score Matching (PSM)

Input: Treatment data $\mathcal{D}_1$, Control data $\mathcal{D}_0$, Covariates $\mathcal{X} = \{X_1, \ldots, X_p\}$
Input: Configuration: K (neighbors), δ (caliper threshold)
Output: ATT estimate $\hat{\tau}_{PSM}$

Step 1: Prepare combined dataset
$\mathcal{D}_1$.treatment $\leftarrow 1$
$\mathcal{D}_0$.treatment $\leftarrow 0$
$\mathcal{D} \leftarrow$ concatenate$(\mathcal{D}_1, \mathcal{D}_0)$
Step 2: Extract and scale covariates
$X \leftarrow \mathcal{D}[\mathcal{X}]$ ▷ Extract covariate matrix
$X \leftarrow$ fillna$(X, 0)$ ▷ Impute missing values
$X_{\text{scaled}} \leftarrow$ StandardScaler.fit_transform(X)
Step 3: Estimate propensity scores
Train logistic regression: $\hat{e}(X) = \sigma(\beta_0 + \sum_{j=1}^{p} \beta_j X_j)$
$\hat{e}_i \leftarrow$ LogisticRegression.predict_proba$(X_{\text{scaled},i})[:, 1]$ for all i
$\mathcal{D}$.propensity_score $\leftarrow \hat{e}$
Step 4: Perform nearest neighbor matching
$\mathcal{T} \leftarrow \{i : \mathcal{D}_i.\text{treatment} = 1\}$ ▷ Treatment indices
$\mathcal{C} \leftarrow \{i : \mathcal{D}_i.\text{treatment} = 0\}$ ▷ Control indices
Initialize K-NN model on control propensity scores
effects $\leftarrow []$ ▷ Store matched pair effects
for each $i \in \mathcal{T}$ **do**
  Find K nearest neighbors $\mathcal{N}_i \subset \mathcal{C}$ where:
    $|\hat{e}_i - \hat{e}_j| \leq \delta$ for all $j \in \mathcal{N}_i$ ▷ Caliper constraint
  if $\mathcal{N}_i \neq \emptyset$ **then**
    $Y_i^{\text{treat}} \leftarrow \mathcal{D}_i$.qoe_metric
    $Y_i^{\text{control}} \leftarrow \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \mathcal{D}_j$.qoe_metric
    effects.append$(Y_i^{\text{treat}} - Y_i^{\text{control}})$
  end if
end for
Step 5: Compute ATT
$\hat{\tau}_{PSM} \leftarrow \frac{1}{|\text{effects}|} \sum_{\text{effect} \in \text{effects}} \text{effect}$
return $\hat{\tau}_{PSM}$

6.3 PSM Process Flow

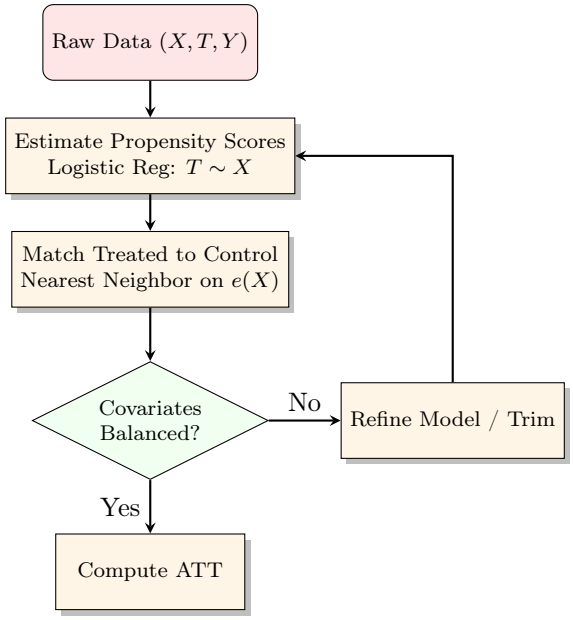

Figure 2: Propensity Score Matching Workflow

6.4 Implementation Details

To ensure robustness in the WiFi environment where noise is high:

- **Model**: Logistic regression with L2 regularization (max iterations = 1000) to handle collinear network metrics
- **Matching**: K-nearest neighbors ($K = 3$ default) to reduce variance
- **Caliper**: $\delta = 0.15$ standard deviations of propensity score
- **Variance estimation**: Bootstrap with replacement ($B = 100$) to generate confidence intervals

7 Method 2: Double/Debiased Machine Learning (DML)

Modern WiFi networks generate high-dimensional telemetry data. Traditional linear models may fail to capture complex non-linear relationships, while standard Machine Learning models introduce regularization bias. Double Machine Learning (DML) solves this by using ML to model nuisance parameters while preserving valid statistical inference for the treatment effect.

7.1 Theory

DML uses two ML models: one to predict the outcome from covariates, and another to predict the treatment from covariates. It then correlates the residuals of these predictions.

$$\boxed{\psi(W;\tau,\eta) = (Y - \ell(X)) - \tau(T - m(X))} \tag{6}$$

where $m(X) = E[T \mid X]$ (propensity score) and $\ell(X) = E[Y \mid X]$ (outcome regression).

The key insight is that this score is **orthogonal** to nuisance parameters $\eta = (m, \ell)$, meaning small errors in estimating m and ℓ do not bias the treatment effect τ.

7.2 Cross-Fitting to Avoid Overfitting

To prevent overfitting bias from learning nuisance functions and treatment effect on same data, DML employs **K-fold cross-fitting**.

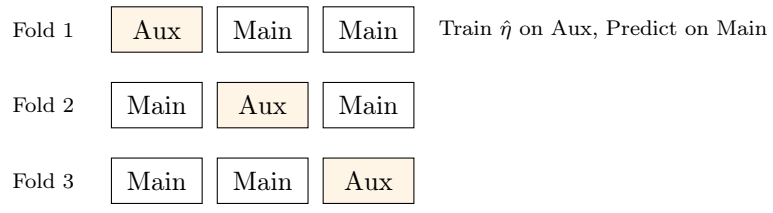

Figure 3: K-Fold Cross-Fitting Concept (K=3)

7.3 Algorithm: Double Machine Learning

Algorithm 3 Double/Debiased Machine Learning (DML)

Input: Treatment data $\mathcal{D}_1$, Control data $\mathcal{D}_0$, Covariates $\mathcal{X}$
Input: Configuration: K (number of folds)
Output: ATT estimate $\hat{\tau}_{DML}$

Step 1: Prepare combined dataset
$\mathcal{D}_1$.treatment $\leftarrow 1$
$\mathcal{D}_0$.treatment $\leftarrow 0$
$\mathcal{D} \leftarrow$ concatenate$(\mathcal{D}_1, \mathcal{D}_0)$
$\mathcal{D} \leftarrow$ shuffle$(\mathcal{D})$ ▷ Random shuffle for cross-validation
Step 2: Extract features
$X \leftarrow \mathcal{D}[\mathcal{X}]$.fillna(0).values
$T \leftarrow \mathcal{D}$.treatment.values
$Y \leftarrow \mathcal{D}$.qoe_metric.values
$n \leftarrow$ len$(\mathcal{D})$
Step 3: K-Fold Cross-Fitting
theta_estimates $\leftarrow []$ ▷ Store treatment effects from each fold
for $k = 1$ to K **do**
▷ Split data into auxiliary and main samples
  $\mathcal{I}_{\text{aux}} \leftarrow \{\text{fold } k \text{ indices}\}$
  $\mathcal{I}_{\text{main}} \leftarrow \{1, \ldots, n\} \setminus \mathcal{I}_{\text{aux}}$
  $X_{\text{aux}}, T_{\text{aux}}, Y_{\text{aux}} \leftarrow X[\mathcal{I}_{\text{aux}}], T[\mathcal{I}_{\text{aux}}], Y[\mathcal{I}_{\text{aux}}]$
  $X_{\text{main}}, T_{\text{main}}, Y_{\text{main}} \leftarrow X[\mathcal{I}_{\text{main}}], T[\mathcal{I}_{\text{main}}], Y[\mathcal{I}_{\text{main}}]$
  if $|\text{unique}(T_{\text{aux}})| < 2$ **then**
    continue ▷ Skip if insufficient treatment variation
  end if
▷ Fit propensity model on auxiliary sample
  Train: $\hat{m}_k(X) = P(T = 1 \mid X)$ using LogisticRegression on $(X_{\text{aux}}, T_{\text{aux}})$
  $\hat{m}_{\text{main}} \leftarrow \hat{m}_k$.predict_proba$(X_{\text{main}})[:, 1]$
▷ Fit outcome model on auxiliary sample
  Train: $\hat{\ell}_k(X) = E[Y \mid X]$ using GradientBoostingRegressor on $(X_{\text{aux}}, Y_{\text{aux}})$
  $\hat{\ell}_{\text{main}} \leftarrow \hat{\ell}_k$.predict$(X_{\text{main}})$
▷ Compute orthogonal residuals on main sample
  $V \leftarrow T_{\text{main}} - \hat{m}_{\text{main}}$ ▷ Treatment residuals
  $W \leftarrow Y_{\text{main}} - \hat{\ell}_{\text{main}}$ ▷ Outcome residuals
▷ Compute fold-specific treatment effect
  denominator $\leftarrow \sum_i V_i \cdot T_{\text{main},i}$
  if $|\text{denominator}| > 10^{-6}$ **then**
    $\hat{\theta}_k \leftarrow \frac{\sum_i V_i \cdot W_i}{\text{denominator}}$
    theta_estimates.append$(\hat{\theta}_k)$
  end if
end for
Step 4: Aggregate across folds
if theta_estimates $= \emptyset$ **then**
  return $\bar{Y}_1 - \bar{Y}_0$ ▷ Fallback to simple difference
else
  $\hat{\tau}_{DML} \leftarrow \frac{1}{K} \sum_{k=1}^{K} \hat{\theta}_k$
  return $\hat{\tau}_{DML}$
end if

7.4 Implementation Details

- **Propensity Model**: Logistic Regression (max_iter=500, L2 regularization)
- **Outcome Model**: Gradient Boosting Regressor (n_estimators=50, learning_rate=0.1)
- **Folds**: $K = 3$ for computational efficiency with small samples
- **Robustness**: Skips folds with insufficient treatment variation or numerical instability

8 Aggregation and Statistical Inference

Each method (PSM, DML) has different strengths and biases. To produce a robust causal estimate, we employ a meta-aggregation strategy that combines estimates while quantifying uncertainty.

8.1 Meta-Estimator

$$\boxed{\hat{\tau}_{\text{causal}} = \text{median}\{\hat{\tau}_{PSM}, \hat{\tau}_{DML}\}} \tag{7}$$

Rationale: The median is robust to outliers from model misspecification, whereas the mean would be sensitive to a single failed method.

8.2 Confidence Interval Construction

We derive uncertainty from cross-method variance:

Standard Error:

$$\text{SE}(\hat{\tau}) = \frac{1}{\sqrt{M}} \sqrt{\frac{1}{M-1} \sum_{m=1}^{M} (\hat{\tau}_m - \hat{\tau})^2} \tag{8}$$

where M is the number of successful methods.

95% Confidence Interval:

$$\boxed{\text{CI}_{0.95} = [\hat{\tau} - 1.96 \times \text{SE}(\hat{\tau}), \hat{\tau} + 1.96 \times \text{SE}(\hat{\tau})]} \tag{9}$$

8.3 Hypothesis Testing

Null Hypothesis: $H_0 : \tau = 0$ (no treatment effect)

Test Statistic:

$$t = \frac{\hat{\tau}}{\text{SE}(\hat{\tau}) + \epsilon} \tag{10}$$

where $\epsilon = 10^{-10}$ prevents division by zero.

P-value:

$$p = 2 \times (1 - \Phi(|t|)) \tag{11}$$

where Φ is the Student's t-distribution CDF with $df = M - 1$.

Decision Rule:

- If $p < \alpha$ (typically $\alpha = 0.05$): Reject H_0, effect is **significant**
- If $p \geq \alpha$: Fail to reject H_0, effect is **not significant**

9 Covariate Balance Assessment

Before accepting a causal estimate, we validate the quality of matching/weighting by checking covariate balance between treatment and control groups.

9.1 Standardized Mean Difference (SMD)

For each covariate X_j:

$$\boxed{\text{SMD}_j = \frac{\bar{X}_{j,1} - \bar{X}_{j,0}}{\sqrt{(s_{j,1}^2 + s_{j,0}^2)/2}}} \tag{12}$$

where:

- $\bar{X}_{j,1}$: Mean of covariate j in treatment group
- $\bar{X}_{j,0}$: Mean of covariate j in control group
- $s_{j,1}^2, s_{j,0}^2$: Variances in treatment and control groups

Balance Criterion: $|\text{SMD}_j| < 0.1$ indicates excellent balance.

9.2 Overall Balance Score

We compute a global balance score $\beta \in [0, 1]$:

$$\boxed{\beta = \max\left(0, 1 - \frac{\overline{\text{SMD}}}{0.5}\right)} \tag{13}$$

where $\overline{\text{SMD}} = \frac{1}{p}\sum_{j=1}^{p} |\text{SMD}_j|$ is the average absolute SMD across all covariates.
Interpretation:

- $\beta = 1.0$: Perfect balance ($\overline{\text{SMD}} = 0$)
- $\beta \geq 0.8$: Good balance
- $\beta < 0.5$: Poor balance, estimates may be unreliable

10 Reward Adjustment for RL Integration

This section connects the Causal Inference module to the Reinforcement Learning agent. Standard RL rewards can be noisy; by filtering through causal validation, we ensure the agent learns only from statistically significant improvements.

10.1 Causal Reward Function

The final reward incorporates both the causal effect and its statistical significance:

$$\boxed{R_{\text{causal}} = \begin{cases} R_{\text{RL}}(\hat{\tau}) \times \beta & \text{if } p < \alpha \\ R_{\text{RL}}(\hat{\tau}) \times 0.1 & \text{if } p \geq \alpha \end{cases}} \tag{14}$$

where $R_{\text{RL}}(\cdot)$ is the RL-compatible reward function (detailed below), and β is the balance score.

10.2 RL-Compatible Reward Computation

Algorithm 4 Compute RL-Compatible Reward

Input: Causal effect $\hat{\tau}$, Treatment data $\mathcal{D}_1$, Is significant: sig
Input: Weights: $w_{\text{penalty}} = 3.0$, $w_{\text{uplift}} = 1.0$, $w_{\text{state}} = 0.5$
Output: Causal reward $R_{\text{causal}} \in [-10, 10]$

Step 1: Apply significance discount
$\Delta_{\text{QoE}} \leftarrow \hat{\tau}$
if sig = False **then**
    $\Delta_{\text{QoE}} \leftarrow \Delta_{\text{QoE}} \times 0.1$ ▷ 90% discount for non-significant
end if
Step 2: Apply asymmetric weights
if $\Delta_{\text{QoE}} < 0$ **then**
    $R_{\Delta} \leftarrow w_{\text{penalty}} \times \Delta_{\text{QoE}}$ ▷ Penalize degradation 3x
else
    $R_{\Delta} \leftarrow w_{\text{uplift}} \times \Delta_{\text{QoE}}$ ▷ Reward improvement 1x
end if
Step 3: Add state bias
$\text{QoE}_{\text{next}} \leftarrow \text{mean}(\mathcal{D}_1.\text{qoe\_metric})$
$R \leftarrow R_{\Delta} + w_{\text{state}} \times \text{QoE}_{\text{next}}$
Step 4: Clip to bounds
$R_{\text{causal}} \leftarrow \text{clip}(R, -10, 10)$
return R_{causal}

10.3 Rationale for Reward Shaping

- **Significant effect + Good balance**: Full reward scaled by balance score β. This encourages actions where causal link is clear
- **Non-significant effect**: 90% discount prevents "hallucinated" learning from noise
- **Asymmetric penalties**: Penalize QoE degradation 3x more than rewarding improvement (conservative RL policy)
- **State bias**: Encourages maintaining high absolute QoE, not just relative improvements

11 Model Persistence and Progressive Learning

Real-world WiFi deployments may have limited initial data. To handle this, the engine implements **progressive learning** through model caching.

11.1 Strategy

1. **Initial Phase** ($n < 30$ samples): Train models from scratch, cache for future use
2. **Growth Phase** ($30 \leq n < 100$ samples):
   - If new batch has ≥ 30 samples: Retrain and update cache
   - If new batch has < 30 samples: Use cached models (avoid overfitting)
3. **Mature Phase** ($n \geq 100$ samples): Always retrain on latest data

11.2 Cached Artifacts

Table 10: Model Cache Contents

File	Contents
`propensity_model.pkl`	Trained LogisticRegression for $\hat{e}(X)$
`outcome_model.pkl`	Trained GradientBoostingRegressor for $\hat{\ell}(X)$
`scaler.pkl`	Fitted StandardScaler for feature normalization
`model_metadata.json`	Training metadata (sample count, timestamp, config)

11.3 Progressive Learning Workflow

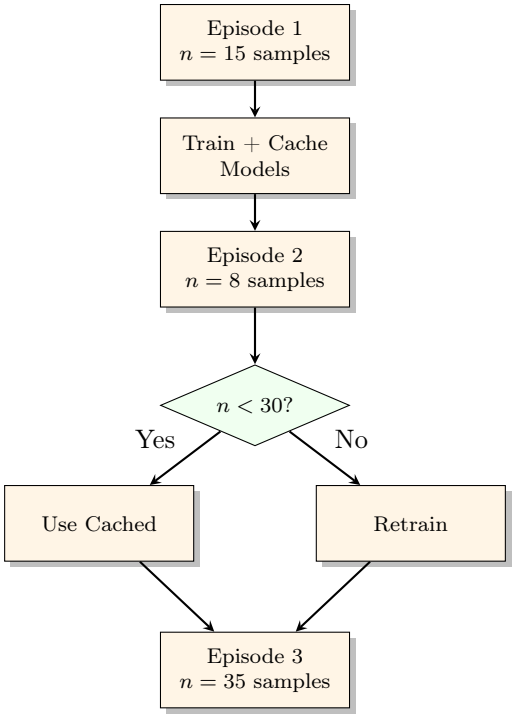

Figure 4: Progressive Learning Across Episodes

12 Integration with RL System

12.1 Usage Example

Listing 3: Basic Usage

```python
from causal_reward_engine import CausalRewardEngine, CausalConfig

# Configure engine
config = CausalConfig(
    graph_data_dir="causal_data/graphs",
    action_file_path="approved_changes.csv",
    min_graphs_required=10,
    max_graphs_to_load=20
)

# Create engine
engine = CausalRewardEngine(config=config)

# Compute causal reward
result = engine.compute_causal_reward_from_graphs()

# Extract reward for RL agent
causal_reward = result.causal_reward
print(f"Causal Reward: {causal_reward:.4f}")
print(f"Is Significant: {result.is_significant}")
print(f"95% CI: [{result.confidence_interval[0]:.4f}, {result.confidence_interval[1]:.4f}]")
```

12.2 Batch Integration

For offline RL training, replace GNN predictions with causal rewards:

Listing 4: Batch Replacement

```python
# Traditional approach (using GNN predictions)
gnn_reward = gnn_model.predict(state, action)

# Causal approach (using real data)
causal_result = engine.compute_causal_reward_from_graphs(
    graph_data_dir="episode_graphs/",
    action_file_path="episode_actions.csv"
)
causal_reward = causal_result.causal_reward

# Use causal_reward in experience replay buffer
buffer.add(state, action, causal_reward, next_state)
```

13 Computational Complexity

Table 11: Computational Complexity Analysis

Component	**Time Complexity**	**Space Complexity**
Load Graphs	$O(G \cdot N)$	$O(G \cdot N)$
QoE Computation	$O(N)$	$O(N)$
PSM	$O(n_1 n_0 \log n_0)$	$O(n_1 + n_0)$
DML (K-fold)	$O(K \cdot n \cdot d \log n)$	$O(n \cdot d)$
Balance Score	$O(p \cdot n)$	$O(p)$
Total	$O(G \cdot N + K \cdot n \cdot d \log n)$	$O(G \cdot N)$

Legend:

- G: Number of graph snapshots (typically 10-20)
- N: Number of APs per graph (typically 5-50)
- $n = n_1 + n_0$: Total samples (treatment + control)
- d: Dimensionality of covariates (typically 7-15)
- K: Number of cross-validation folds (3)
- p: Number of covariates to check for balance

Typical Runtime: For $G = 15$, $N = 20$, $n = 300$, $d = 10$:

- Load + QoE: ~ 0.5 seconds
- PSM: $\sim 1 - 2$ seconds
- DML: $\sim 3 - 5$ seconds (dominated by GBM training)
- **Total:** $\sim 5 - 8$ **seconds per evaluation**

14 Summary and Conclusion

This document presents a comprehensive design for a Causal Reward Engine that validates RRM actions through rigorous causal inference. The system:

14.1 Key Achievements

- **Replaces correlation with causation** by using PSM and DML to isolate true treatment effects
- **Maintains RL compatibility** by producing rewards in the same format as existing GNN predictions
- **Handles small data** through progressive learning and model persistence
- **Provides statistical guarantees** via significance testing, confidence intervals, and balance checks
- **Implements robust fallbacks** to ensure reliability in production environments

14.2 System Capabilities

- Propensity Score Matching with K-nearest neighbors and caliper constraints
- Double Machine Learning with cross-fitting to prevent overfitting
- Meta-aggregation for robust effect estimation
- Comprehensive balance assessment using Standardized Mean Differences
- Progressive learning with model caching for small sample scenarios
- Sub-10 second inference for real-time deployment

14.3 Integration Benefits

The engine is designed as a drop-in replacement for GNN-based reward prediction, enabling causally-validated RL training for WiFi network optimization.

Key Takeaway: By filtering RL rewards through causal validation, we ensure the agent learns from real improvements rather than spurious correlations, leading to more reliable and effective RRM policies.

14.4 Future Enhancements

- Network interference modeling using graph structure
- Time-series methods with Difference-in-Differences
- Synthetic control for single-unit treatments
- Heterogeneous treatment effect estimation
- Active learning integration with RL exploration