Ontario Health

## Data and Decision Sciences

# Developing a Centralized, Real-Time Referral Routing Model Using Advanced Analytics for Diagnostic Imaging

Proposal by:

Markus Amalanathan, M.Eng.

December 2, 2024

# Table of Contents

# Executive Summary

The proposed project aims to address significant inefficiencies in the Canadian healthcare system's management of diagnostic imaging (DI) referrals. Prolonged patient wait times, unbalanced hospital workloads, and travel burdens pose challenges, hindering patient outcomes and straining healthcare resources. The proposed solution involves a centralized, real-time referral routing model. It leverages advanced analytics, optimization techniques, and machine learning.

## Proposed Solution

The system integrates predictive analytics and mixed-integer linear programming (MILP) optimization to dynamically assign DI referrals by considering the following factors:

- **Reducing Wait Times**: Accurate predictive models ensure patients are routed to hospitals with minimal wait times.
- **Balancing Workloads**: Optimization algorithms prevent overloading specific hospitals by distributing referrals equitably.
- **Minimizing Travel Burden**: Geographic proximity is considered to reduce travel times for patients.

## Key Features

- **Real-Time Decision-Making**: Integrates live data feeds on hospital capacities with intelligent routing algorithms.
- **Scalability**: Cloud-based architecture and distributed processing ensure the system meets growing healthcare demands.
- **Dynamic Adaptability**: Heuristic algorithms handle unexpected urgent patient referrals and sudden changes in hospital capacity.

## Implementation Approach

The solution is developed and deployed through a structured six-step process:

1. **Problem Formulation**: Establishing mathematical objectives and constraints.
2. **Data Preparation**: Cleaning, enriching, and integrating real-time and historical data.
3. **Model Development**: Building machine learning models to predict wait times and optimization frameworks for routing.
4. **System Architecture Design**: Modular architecture with data input, processing, and feedback layers.
5. **Evaluation and Testing**: Simulating scenarios to test the system's efficiency and fairness.
6. **Deployment and Scalability**: A phased rollout with continuous monitoring and adaptation.

---

# Problem Formulation

## Objectives

- Minimize patient wait times ($WT$).
- Balance hospital workloads ($HL$).
- Incorporate proximity ($GP$) to minimize travel burden.

## Mathematical Model Details

1. **Decision Variables**:

$$x_{ph} = \begin{cases} 1 & \text{if patient } p \text{ is assigned to hospital } h, \\ 0 & \text{otherwise.} \end{cases}$$

2. **Objective Function**:
We aim to minimize a cost function $J$ that combines predicted wait time ($WT$), load imbalance ($HL$), and geographic proximity ($GP$):

$$J = \sum_{p \in P} \sum_{h \in H} \left( \alpha \cdot WT_{ph} + \beta \cdot \frac{|L_h - \bar{L}|}{\bar{L}} + \gamma \cdot GP_{ph} \right) \cdot x_{ph}$$

where:

- $\alpha, \beta, \gamma$: Weighting factors prioritizing different objectives.
- $L_h$: Current load of hospital $h$.
- $\bar{L}$: Average load across all hospitals.

3. **Constraints**:

(a) Each patient assigned to one hospital:

$$\sum_{h \in H} x_{ph} = 1, \quad \forall p \in P$$

(b) Hospital capacity respected:

$$\sum_{p \in P} x_{ph} \leq C_h, \quad \forall h \in H$$

(c) Binary assignments:

$$x_{ph} \in \{0, 1\}, \quad \forall p \in P, h \in H$$

## Example with Hypothetical Data

- **Patients**: $P = \{p_1, p_2, p_3\}$
- **Hospitals**: $H = \{h_1, h_2\}$
- Predicted wait times ($WT$):

$$WT_{p_1 h_1} = 30, \; WT_{p_1 h_2} = 20, \text{ and so on.}$$

- Geographic proximity cost ($GP$):

$$GP_{p_1 h_1} = 5, \; GP_{p_1 h_2} = 15, \text{ and so on.}$$

---

# Data Preparation Strategies

## Step 1: Data Sources

To build a reliable referral routing model, data preparation begins with identifying relevant data sources:

- **Historical Data:** Hospital capacities, patient volumes, and historical wait times.
- **Geographic Data:** Patient and hospital locations in geographic coordinate systems.
- **Real-Time Inputs:** Current hospital loads, scanner availability, and queue sizes.

## Step 2: Data Cleaning

Data cleaning ensures the consistency and quality of input data. The key steps are:

- Handling missing or inconsistent records (e.g., fill null capacities with averages or domain-specific estimates).
- Standardize geographic coordinates to a uniform projection system, such as Universal Transverse Mercator (UTM).

### Sample Code Snippet: Data Cleaning

```python
import pandas as pd
from sklearn.impute import SimpleImputer

# Example hospital data with missing capacities
hospitals = pd.DataFrame({
    "HospitalID": [1, 2, 3, 4, 5],
    "Capacity": [100, None, 50, None, 75],
    "Latitude": [43.7, 43.8, 43.9, 44.0, 44.1],
    "Longitude": [-79.4, -79.5, -79.3, -79.6, -79.2]
})

# Impute missing capacities with mean
imputer = SimpleImputer(strategy="mean")
hospitals["Capacity"] = imputer.fit_transform(hospitals[["Capacity"]])

print(hospitals)
```

## Step 3: Feature Engineering

Feature engineering extracts actionable insights from raw data:

- **Average Wait Time:**
$$WT_{\text{avg}} = \frac{\sum_{i=1}^{n} WT_i}{n}$$
where $WT_i$ is the historical wait time for the $i$-th patient.
- **Geographic Proximity:** Calculate the distance between patients and hospitals using the haversine formula:
$$d(p,h) = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_p \cos\phi_h \sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$
Here:
  - $\Delta\phi = \phi_h - \phi_p$: Difference in latitudes.
  - $\Delta\lambda = \lambda_h - \lambda_p$: Difference in longitudes.
  - $r$: Radius of the Earth (mean value: $6,371$ km).

### Code Snippet for Calculating Geographic Proximity:

```python
from math import radians, sin, cos, sqrt, asin

def haversine(lat1, lon1, lat2, lon2):
    r = 6371  # Radius of the Earth in kilometers
    dlat = radians(lat2 - lat1)
    dlon = radians(lon2 - lon1)
    a = sin(dlat / 2)**2 + cos(radians(lat1)) * cos(radians(lat2)) * sin(dlon / 2)**2
    c = 2 * asin(sqrt(a))
    return r * c

# Example: Patient at (43.7, -79.4), Hospital at (43.8, -79.5)
distance = haversine(43.7, -79.4, 43.8, -79.5)
print(distance)
```

## Step 4: Simulated Data Generation

In the absence of historical data, synthetic datasets are created to mimic real-world scenarios.

### Python Code for Simulating Data:

```python
import pandas as pd
import numpy as np

# Create synthetic patient data
patients = pd.DataFrame({
    "PatientID": range(1, 101),
    "Age": np.random.randint(18, 80, 100),
    "Urgency": np.random.choice(["High", "Medium", "Low"], 100, p=[0.2, 0.5, 0.3]),
    "Location": [(np.random.uniform(-79.5, -79.2), np.random.uniform(43.6, 43.8)) for _ in
    ↪  range(100)]
})

# Create synthetic hospital data
hospitals = pd.DataFrame({
    "HospitalID": range(1, 6),
    "Capacity": np.random.randint(5, 15, 5),
    "Location": [(np.random.uniform(-79.5, -79.2), np.random.uniform(43.6, 43.8)) for _ in
    ↪  range(5)]
})

print(patients.head(), hospitals.head())
```

## Step 5: Iterative Model Refinement

- Compare predicted wait times or workloads with observed values.
- Use discrepancies to adjust model parameters iteratively.

Mathematical formulation for iterative refinement:

$$\epsilon_t = \hat{y}_t - y_t$$

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta J(\theta_t)$$

where:

- $\epsilon_t$: Prediction error.
- $\theta_t$: Model parameters at time $t$.
- $\eta$: Learning rate.
- $J(\theta)$: Loss function.

# Model Development

## Step 1: Machine Learning Model for Wait Time Prediction

The model predicts hospital wait times using a Random Forest regression algorithm, trained on features derived from historical and real-time data. Key features include:

- **Hospital Capacity Utilization**: Fraction of hospital capacity currently occupied.
- **Day of the Week and Time of Day**: Temporal patterns that affect wait times.
- **Historical Wait Times**: Average past wait times for similar patients.
- **Geographic Proximity**: Distance between patients and hospitals (optional for enhancement).

**Mathematical Formulation:** The wait time prediction model minimizes the Mean Squared Error (MSE) during training:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where:

- $y_i$: Actual wait time for the $i$-th patient.
- $\hat{y}_i$: Predicted wait time for the $i$-th patient.
- $n$: Total number of samples in the training set.

## Code Snippet for Model Training and Evaluation:

The following Python code demonstrates the training and evaluation of a Random Forest regression model:

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Example feature and target data
features = pd.DataFrame({
    "CapacityUtilization": [0.8, 0.6, 0.9, 0.4],
    "DayOfWeek": [1, 3, 4, 5],
    "HistoricalWaitTime": [30, 45, 20, 60]
})
wait_times = [35, 50, 25, 55]

# Split data
X_train, X_test, y_train, y_test = train_test_split(features, wait_times, test_size=0.2,
↪  random_state=42)

# Train model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predictions and evaluation
predictions = rf_model.predict(X_test)
mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)

print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
```

## Step 2: Model Improvement Strategies

To enhance model performance and robustness, consider the following strategies:

- **Feature Engineering**:
  - Include patient-specific features like urgency level or medical history.
  - Add weather or seasonal data to account for external factors affecting wait times.
- **Hyperparameter Tuning**: Use grid search or randomized search to optimize parameters like:
  - Number of estimators (`n_estimators`).
  - Maximum tree depth (`max_depth`).
- **Cross-Validation**: Apply $k$-fold cross-validation to ensure the model generalizes well to unseen data.

$$\text{CV Error} = \frac{1}{k}\sum_{j=1}^{k}\text{MSE}_j$$

## Code Snippet for Hyper-parameter Tuning with Grid Search:

```python
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid = {
    "n_estimators": [50, 100, 200],
    "max_depth": [None, 10, 20],
    "min_samples_split": [2, 5, 10]
}

# Perform grid search
grid_search = GridSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_grid=param_grid,
    cv=5,
    scoring="neg_mean_squared_error"
)
grid_search.fit(X_train, y_train)

# Best parameters and evaluation
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", -grid_search.best_score_)
```

## Step 3: Real-Time Model Updates

Incorporate real-time data streams to refine predictions dynamically:

- **Online Learning Algorithms**: Use incremental models like 'River' to adapt to new data without retraining from scratch.

## Code Snippet for Online Learning:

```python
from river import linear_model, preprocessing, metrics

# Initialize online learning model
model = preprocessing.StandardScaler() | linear_model.LinearRegression()
metric = metrics.MAE()

# Simulated data stream
data_stream = [
    ({"CapacityUtilization": 0.8, "DayOfWeek": 1, "HistoricalWaitTime": 30}, 35),
```

```
    ({"CapacityUtilization": 0.6, "DayOfWeek": 3, "HistoricalWaitTime": 45}, 50),
]

# Train incrementally
for x, y in data_stream:
    y_pred = model.predict_one(x)
    model.learn_one(x, y)
    metric.update(y, y_pred)

print("Mean Absolute Error:", metric.get())
```

# Comprehensive Model Development Strategy: Key Outcomes

- A Random Forest regression model predicts wait times effectively, leveraging a diverse set of features.
- Advanced techniques like hyper-parameter tuning and cross-validation improve model robustness and accuracy.
- Online learning ensures the model adapts dynamically to real-time data.

---

# System Architecture

## Overview

The proposed system architecture integrates data processing, machine learning models, and optimization techniques into a cohesive framework designed to handle real-time decision-making and dynamic adaptability. The architecture is modular, ensuring scalability and maintainability as the system evolves to meet increasing demands.

## Key Components

The architecture is divided into three primary layers:

1. **Data Input Layer**
2. **Processing Layer**
3. **Output and Feedback Layer**

## Step 1: Data Input Layer

This layer is responsible for collecting and integrating data from various sources, ensuring the availability of both historical and real-time data for processing. Key features include:

- **Real-Time Data Feeds**: Continuous streams of hospital capacities, scanner availability, and queue sizes.
- **Batch Data Processing**: Periodic ingestion of historical records, such as wait times and patient referral data.
- **Data Validation**: Automatic checks for data integrity, completeness, and format compliance.

### Code Snippet for Simulated Data Integration

```
import pandas as pd
import numpy as np

# Simulating real-time hospital data feed
```

```python
real_time_data = pd.DataFrame({
    "HospitalID": range(1, 6),
    "CurrentLoad": np.random.randint(10, 50, 5),
    "Capacity": np.random.randint(50, 100, 5),
    "ScannerAvailability": np.random.choice([0, 1], size=5, p=[0.2, 0.8])
})

# Simulating historical wait time data
historical_data = pd.DataFrame({
    "HospitalID": range(1, 6),
    "AverageWaitTime": np.random.randint(20, 60, 5),
    "HistoricalCapacityUtilization": np.random.uniform(0.5, 0.9, 5)
})

# Merging real-time and historical data
combined_data = pd.merge(real_time_data, historical_data, on="HospitalID")

# Validate data integrity
assert not combined_data.isnull().values.any(), "Data contains missing values"
print(combined_data)
```

## Step 2: Processing Layer

This layer performs advanced analytics and optimization to dynamically assign referrals. It consists of the following subcomponents:

**Predictive Analytics Subcomponent** Predictive models forecast hospital wait times based on historical trends and current conditions. The Random Forest regression model described earlier is applied here.

**Optimization Subcomponent** An optimization framework solves the referral assignment problem, balancing patient wait times, hospital workloads, and travel distances. The mathematical formulation is as follows:

$$\text{Objective Function:} \quad J = \sum_{p \in P} \sum_{h \in H} \left( \alpha \cdot WT_{ph} + \beta \cdot \frac{|L_h - \bar{L}|}{\bar{L}} + \gamma \cdot GP_{ph} \right) \cdot x_{ph}$$

Subject to:

$$\sum_{h \in H} x_{ph} = 1 \quad \forall p \in P$$

$$\sum_{p \in P} x_{ph} \leq C_h \quad \forall h \in H$$

$$x_{ph} \in \{0, 1\} \quad \forall p \in P, h \in H$$

### Code Snippet for Optimization Model Implementation

```python
# Example cost matrix for three patients and two hospitals
cost_matrix = [
    [30, 20],  # Costs for patient 1
    [25, 30],  # Costs for patient 2
    [40, 35]   # Costs for patient 3
]

# Flatten cost matrix
c = [item for sublist in cost_matrix for item in sublist]

# Constraints: Each patient assigned to one hospital
A_eq = [
    [1, 1, 0, 0, 0, 0],  # Patient 1
```

```
    [0, 0, 1, 1, 0, 0],  # Patient 2
    [0, 0, 0, 0, 1, 1]   # Patient 3
]
b_eq = [1, 1, 1]  # One hospital per patient

# Constraints: Hospital capacity
A_ub = [
    [1, 0, 1, 0, 1, 0],  # Hospital 1
    [0, 1, 0, 1, 0, 1]   # Hospital 2
]
b_ub = [2, 2]  # Maximum capacity of each hospital

# Bounds: Binary decision variables (relaxed for linear programming)
x_bounds = [(0, 1)] * len(c)

# Solve the optimization problem
result = linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=x_bounds,
↪  method='highs')

# Display results
print("Optimal Assignment:", result.x.reshape(3, 2))
print("Minimum Cost:", result.fun)
```

## Step 3: Output and Feedback Layer

This layer generates actionable insights and provides continuous feedback to improve system performance.

- **Referral Assignments**: Outputs a list of patients assigned to specific hospitals.
- **Feedback Loop**: Updates predictive and optimization models with new data, improving accuracy over time.
- **APIs for Real-Time Decisions**: Facilitates seamless integration with existing healthcare IT systems.

## Real-Time Adaptability

To handle urgent referrals or sudden changes in hospital capacity, a heuristic algorithm adjusts assignments dynamically.

### Code Snippet for Heuristic Real-Time Adjustment

```
# Example patient and hospital assignments
assignments = pd.DataFrame({
    "PatientID": [1, 2, 3],
    "AssignedHospital": [1, 2, 1],
    "Urgency": [False, True, False]
})

# Urgent referral re-routing
urgent_patients = assignments[assignments["Urgency"]]
for index, row in urgent_patients.iterrows():
    # Assign to the nearest hospital with available capacity
    nearest_hospital = combined_data[combined_data["CurrentLoad"] <
    ↪  combined_data["Capacity"]].iloc[0]
    assignments.loc[index, "AssignedHospital"] = nearest_hospital["HospitalID"]

print(assignments)
```

### Scalability and Future Enhancements

The modular design ensures the architecture can scale as patient volumes increase or as new hospitals join the network. Future enhancements include:

- Integration with external systems like ambulance dispatch and electronic health records.
- Incorporating additional patient-specific factors, such as medical history and insurance preferences.
- Enhancing real-time processing capabilities using distributed systems and cloud technologies.

---

## Evaluation and Testing

Evaluation of the referral routing system for diagnostic imaging referrals involves simulating real-world scenarios and analyzing the system's performance using specific metrics. This ensures the system's robustness and adaptability to varied healthcare environments.

### Simulating Real-World Scenarios

#### Scenario 1: High Demand at a Single Hospital

- **Explanation**: This scenario evaluates the system's ability to manage a sudden influx of patients at a particular hospital, such as during emergencies or localized outbreaks.
- **Input**: A simulated situation where 100 patients require diagnostic imaging in a small geographic area, overwhelming a single hospital.
- **Measure**: The system's capability to redistribute patients to nearby hospitals efficiently, reducing congestion while maintaining acceptable travel times for patients.

#### Code Snippet for Simulation : Scenario 1

```python
import numpy as np
import pandas as pd

# Simulate 100 patients in a small geographic area
patients = pd.DataFrame({
    "PatientID": range(1, 101),
    "Location": [(np.random.uniform(-79.3, -79.2), np.random.uniform(43.7, 43.8)) for _ in range(100)]
})

# Overwhelmed hospital
hospital = {"HospitalID": 1, "Location": (-79.25, 43.75), "Capacity": 20}

# Assign patients to nearby hospitals
patients["AssignedHospital"] = patients["Location"].apply(lambda loc: 1 if hospital["Capacity"] > 0 else "Redistributed")
print(patients.head())
```

#### Scenario 2: Rural vs. Urban Differences

- **Explanation**: This scenario assesses the system's adaptability to varying geographic patient distributions, highlighting challenges in rural areas with sparse facilities versus urban centers with high demand.
- **Input**: A mix of sparse patient referrals from rural regions and dense patient inflow in urban centers.
- **Measure**: The system's effectiveness in minimizing travel burden for rural patients by prioritizing proximity while maintaining equitable distribution across urban facilities.

### Code Snippet for Simulation : Scenario 2

```python
# Generate rural and urban patient locations
rural_patients = pd.DataFrame({
    "PatientID": range(1, 51),
    "Location": [(np.random.uniform(-79.8, -79.6), np.random.uniform(43.5, 43.6)) for _ in range(50)]
})
urban_patients = pd.DataFrame({
    "PatientID": range(51, 101),
    "Location": [(np.random.uniform(-79.4, -79.2), np.random.uniform(43.7, 43.8)) for _ in range(50)]
})

# Combine data
all_patients = pd.concat([rural_patients, urban_patients])
print(all_patients.head())
```

## Metrics

### Efficiency

- **Purpose**: To quantify the improvement in wait times after implementing the optimized routing system.
- **Formula**:

$$\Delta WT = \frac{\text{Current Wait Time} - \text{Optimized Wait Time}}{\text{Current Wait Time}}$$

- **Interpretation**: A higher value indicates a significant reduction in patient wait times due to optimized routing.

### Code Snippet for Calculation of Efficiency:

```python
# Example wait times
current_wait_times = np.array([30, 40, 50, 60])
optimized_wait_times = np.array([20, 25, 30, 35])

# Calculate efficiency improvement
efficiency = (current_wait_times - optimized_wait_times) / current_wait_times
print("Efficiency Improvement:", efficiency)
```

### Load Balancing

- **Purpose**: To measure how evenly the patient load is distributed across hospitals.
- **Formula**:

$$\sigma_L = \sqrt{\frac{1}{N} \sum_{h \in H} (L_h - \bar{L})^2}$$

  where:

  - $L_h$: Number of patients at hospital $h$.
  - $\bar{L}$: Average load across all hospitals.

- **Interpretation**: A lower standard deviation ($\sigma_L$) reflects better load balancing, ensuring no single hospital is overwhelmed while others remain underutilized.

**Code Snippet for Calculation of Load Balancing:**

```python
# Example hospital loads
hospital_loads = np.array([100, 120, 80, 110])
average_load = np.mean(hospital_loads)

# Calculate load balancing metric (standard deviation)
load_balancing = np.sqrt(np.mean((hospital_loads - average_load)**2))
print("Load_Balancing_Metric:", load_balancing)
```

# Comprehensive Evaluation Strategy: Key Outcomes

- **Scenario Testing**: Simulated scenarios evaluate system adaptability to real-world challenges.
- **Metrics Analysis**: Efficiency and load balancing metrics quantify system performance improvements.
- **Iterative Refinement**: Real-world validation drives continuous optimization of the routing system.

---

# Implementation Plan

## Overview

The implementation plan outlines a step-by-step process to deploy the centralized, real-time referral routing model for diagnostic imaging. This plan ensures scalability, efficiency, and practicality while maintaining seamless integration with Ontario Health's existing infrastructure. It consists of four stages: pilot deployment, phased rollout, continuous monitoring, and scaling for future demands.

## Stage 1: Pilot Deployment

The first stage involves deploying the system in a controlled environment to validate its performance and functionality. A subset of hospitals and clinics will be selected for the pilot.

- **Objective**: Test core features like real-time decision-making, wait time prediction, and optimization.
- **Data Collection**: Gather real-time and historical data from selected facilities.
- **Evaluation Metrics**: Measure initial system performance using wait time reduction ($\Delta WT$) and load balancing ($\sigma_L$) metrics.

**Code Snippet for Simulating Pilot Data**

```python
import pandas as pd
import numpy as np

# Simulate hospital data for pilot
pilot_hospitals = pd.DataFrame({
    "HospitalID": range(1, 4),
    "Capacity": [50, 70, 60],
    "CurrentLoad": [30, 40, 35],
    "Location": [(43.7, -79.4), (43.8, -79.5), (43.9, -79.6)]
})

# Simulate patient data for pilot
pilot_patients = pd.DataFrame({
    "PatientID": range(1, 21),
    "Urgency": np.random.choice(["High", "Medium", "Low"], 20, p=[0.3, 0.5, 0.2]),
```

```
    "Location": [(np.random.uniform(43.6, 43.9), np.random.uniform(-79.6, -79.4)) for _ in
    ↪  range(20)]
})

print("Pilot Hospitals:\n", pilot_hospitals)
print("Pilot Patients:\n", pilot_patients)
```

## Stage 2: Phased Rollout

Following a successful pilot, the system will be deployed across additional hospitals in phases. This ensures that scaling does not overwhelm resources and allows for iterative improvements.

- **Objective**: Gradually expand coverage while refining the system based on feedback and performance data.
- **Focus Areas**:
  - Integrate more hospitals incrementally.
  - Train staff and stakeholders on system usage.
  - Monitor key metrics to identify areas for enhancement.

## Stage 3: Continuous Monitoring and Adaptation

Real-time monitoring and adaptive updates are essential for maintaining system efficiency and reliability as new data is incorporated.

- **Dynamic Model Updates**: Use real-time data streams to retrain models periodically.
- **Anomaly Detection**: Implement automated alerts for unexpected variations in wait times or workload distribution.
- **Feedback Loop**: Continuously refine optimization weights $(\alpha, \beta, \gamma)$ and model parameters based on system performance.

**Mathematical Formulation: Adaptive Weight Refinement** Refine the optimization weights using a gradient-based approach to minimize the overall system error:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta J(\theta_t)$$

where:

- $\theta_t = [\alpha_t, \beta_t, \gamma_t]$: Weights at iteration $t$.
- $\eta$: Learning rate.
- $J(\theta)$: Cost function combining wait times, load imbalance, and geographic proximity.

### Snippet for Real-Time Monitoring and Adaptive Updates

```
from sklearn.linear_model import SGDRegressor

# Simulate real-time performance data
performance_data = pd.DataFrame({
    "WaitTimeError": np.random.normal(0, 5, 100),
    "LoadImbalanceError": np.random.normal(0, 3, 100),
    "ProximityError": np.random.normal(0, 2, 100)
})

# Cost function weights
weights = np.array([0.5, 0.3, 0.2])

# Online learning for weight refinement
```

```python
model = SGDRegressor(learning_rate="constant", eta0=0.01)
for _, row in performance_data.iterrows():
    error_vector = row.values.reshape(1, -1)
    model.partial_fit(error_vector, [1])  # Target is a placeholder
    weights = model.coef_

print("Updated Weights:", weights)
```

## Stage 4: Scaling for Future Demands

To accommodate increasing patient volumes and additional hospitals, the system must be scalable and robust.

- **Cloud-Based Infrastructure**: Deploy the system on a cloud platform for scalability and high availability.
- **Distributed Processing**: Use distributed databases and parallel processing frameworks to handle large datasets efficiently.
- **Performance Optimization**: Optimize model inference and decision-making times to ensure real-time responsiveness.

### Code Snippet for Simulating Distributed Data Processing

```python
import dask.dataframe as dd

# Simulate large-scale hospital data
large_hospital_data = pd.DataFrame({
    "HospitalID": range(1, 1001),
    "Capacity": np.random.randint(50, 100, 1000),
    "CurrentLoad": np.random.randint(10, 50, 1000),
    "Location": [(np.random.uniform(43.6, 43.9), np.random.uniform(-79.6, -79.4)) for _ in
    ↪  range(1000)]
})

# Convert to Dask DataFrame
ddf = dd.from_pandas(large_hospital_data, npartitions=10)

# Example operation: Calculate mean load per hospital
mean_load = ddf["CurrentLoad"].mean().compute()
print("Mean Load:", mean_load)
```

## Expected Outcomes

- Demonstrate the system's effectiveness in reducing wait times and balancing workloads during the pilot phase.
- Ensure smooth and scalable system deployment through phased rollouts and continuous monitoring.
- Maintain efficiency and responsiveness as patient volumes and hospital participation increase.

---

# Conclusion

This proposal presents a comprehensive, data-driven approach to optimizing diagnostic imaging referral routing in the Canadian healthcare system. The proposed solution leverages advanced analytics, machine learning, and optimization techniques to address systemic challenges, including:

- **Reducing Patient Wait Times**: Efficient referral allocation minimizes delays and improves patient outcomes.
- **Balancing Hospital Workloads**: Intelligent algorithms ensure equitable distribution of patient referrals, preventing hospital overcrowding.
- **Minimizing Travel Burden**: Geographic proximity is prioritized to reduce patient travel distances, enhancing accessibility.

The phased implementation plan ensures smooth deployment, beginning with a pilot rollout for initial testing and refinement, followed by a scalable phased rollout across the healthcare network. Continuous monitoring and dynamic adaptability allow the system to evolve with growing demands and adapt to real-time data inputs, ensuring long-term effectiveness and reliability.

Key expected outcomes include:

- Improved patient outcomes through reduced wait times and better access to diagnostic imaging.
- Enhanced operational efficiency by leveraging real-time decision-making and predictive analytics.
- Equitable utilization of healthcare resources, reducing strain on individual hospitals and improving system-wide capacity management.

The centralized referral routing model demonstrates a transformative approach to addressing critical inefficiencies in the healthcare system, aligning with Ontario Health's strategic goals. By fostering innovation through advanced analytics, this initiative represents a significant step forward in modernizing healthcare delivery systems and ensuring better care for all patients.

**GitHub Repository Link:-** Centralized Referral Routing Model