Graph Neural Network-Based Personalized Workout Optimization System

Rameez E. Malik
Department of Computer Science
North Carolina State University
Raleigh, NC 27606
remalik@ncsu.edu
Project GitHub: github.com/RameezMilk/workout-optimization-research

Abstract

The rapid growth of fitness tracking technologies has enabled large-scale data collection but limited personalization in performance optimization. This work introduces a **Graph Neural Network (GNN)-based Personalized Workout Optimization System** designed to refine an individual's existing training routine rather than recommend new exercises. Using six weeks of recorded and simulated workout sessions, we model each exercise as a node and inter-exercise relationships as edges, capturing temporal and physiological dependencies. A two-layer Graph Convolutional Network (GCN) serves as the baseline, mirroring the two-layer LSTM structure used in FitRec [1], while a proposed GraphSAGE model generalizes to unseen exercises through inductive neighborhood aggregation. Quantitative evaluation on node-level regression tasks demonstrates consistent performance gains: a 100% reduction in Mean Absolute Error (MAE) and 100% reduction in Root Mean Square Error (RMSE) compared to the GCN baseline. The proposed framework highlights the potential of graph-based representations to model complex training interactions, enabling data-driven workout progression and adaptive overload planning for athletes and general users alike.

1 Introduction

1.1 Task Application Description

The goal of this research is to design a system capable of optimizing an individual's existing workout routine using graph neural networks. Unlike conventional recommender systems that suggest new exercises, this model identifies subtle adjustments, such as changes in load, repetitions, or rest intervals, that can enhance performance while mitigating the risk of overtraining. The application domain lies at the intersection of sports science and machine learning, where physiological trends and structural dependencies among exercises can be represented as relational graphs.

1.2 Existing Approaches

Recent work such as FitRec [1] models workout performance using a two-layer Long Short-Term Memory (LSTM) network to forecast endurance and heart-rate dynamics. While sequential architectures capture temporal continuity, they fail to represent inter-exercise relationships or shared fatigue dynamics between sessions. This limitation motivates exploring graph-based learning, where each exercise is treated as a node connected to related movements by functional or temporal edges.

Preprint. Under review.

1.3 Motivation and Research Gap

Human training routines exhibit relational dependencies: progress in one movement often influences others targeting overlapping muscle groups. Traditional deep-learning models process sessions as independent sequences, ignoring such cross-exercise structure. Graph Neural Networks (GNNs) are uniquely positioned to capture these dependencies by performing message passing between related exercises, thereby embedding contextual fatigue, load progression, and recovery relationships directly into the learning process.

1.4 Challenges in Implementation

Developing a graph-based optimization framework presents multiple challenges:

- Constructing a coherent graph from heterogeneous sequential data.
- Preventing over-smoothing caused by excessive message passing in small datasets.
- Compensating for limited training data through robust feature engineering.
- Ensuring physiological interpretability of model outputs.

1.5 Contributions

The primary contributions of this work are as follows:

1. Developed a complete pipeline that transforms raw workout logs into graph-structured data, integrating both temporal and same-day exercise relationships.
2. Implemented a two-layer GCN baseline aligned with FitRec's 2-layer LSTM depth for fair comparison.
3. Proposed a GraphSAGE-based model that enhances inductive generalization to unseen exercises.
4. Introduced physiological metrics: training load, monotony, and strain, derived from [2] to strengthen representational grounding.
5. Conducted quantitative and interpretive evaluations (MAE, RMSE, and SHAP feature importance) demonstrating the superiority of graph-based modeling for workout optimization.

2 Related Work

2.1 Sequential and Physiological Modeling in Fitness Science

The FitRec framework [1], developed at the University of California San Diego, is one of the earliest studies to model personalized workout performance using deep learning. It employs a two-layer LSTM to predict heart rate and fatigue trends based on sequential exercise data. While this architecture captures temporal patterns within an individual's sessions, it treats each exercise as an isolated time step and therefore cannot represent relational dependencies among different movements or sessions. This limitation motivates the transition from sequence-based to structure-based modeling, where relationships among exercises can be learned explicitly.

In parallel, sports science literature such as the 2019 PLOS ONE study on training load and monotony [2] provides a physiological foundation for representing performance trends numerically. The authors defined three key constructs: *training load* (volume multiplied by intensity), *training monotony* (daily load variability ratio), and *training strain* (product of weekly load and monotony). These metrics quantify how training stress and variability affect performance and fatigue, which informs our feature design for machine learning models. Integrating such metrics allows the network to learn physiologically meaningful relationships rather than relying solely on raw numerical patterns.

2.2 Graph Neural Networks for Relational Learning

Graph Neural Networks (GNNs) have become a powerful framework for modeling systems where entities and their relationships influence outcomes. Architectures such as Graph Convolutional

Networks (GCNs) and GraphSAGE extend convolutional principles to non-Euclidean domains by enabling message passing among connected nodes. In the context of workout optimization, each exercise can be represented as a node, while edges encode functional, temporal, or same-day relations. Through iterative aggregation, GNNs can model cross-exercise interactions, providing richer insight into how one movement affects another over time.

2.3 Gap in Current Literature

Despite significant progress in both sequential and graph-based learning, no prior work has directly modeled workout optimization using graph neural networks. Existing systems focus on predicting single performance metrics, such as heart rate or calories, without explicitly representing inter-exercise structure. This research addresses that gap by introducing a GNN-based framework for predicting optimal progression parameters (load, repetitions, and rest), thereby extending the domain of relational modeling to personal fitness optimization.

3 Method Description

3.1 Dataset and Feature Engineering

The dataset combines six weeks of real and simulated workout sessions exported from Google Sheets. Each record includes exercise name, sets, repetitions, weight, rest time, intensity, fatigue rating, and days since the previous session. To enrich representational depth, three additional physiological metrics were derived following [2]: training load, training monotony, and strain. These metrics help quantify weekly training stress and variability, improving the model's interpretability and generalization in small data regimes.

Progressive overload targets (*next weight*, *next reps*, *next rest*) were generated through rule-based logic inspired by sports science literature, using fatigue and RIR (reps in reserve) thresholds. Each entry therefore includes both current-session features and predicted next-session outcomes, allowing supervised training of the graph model.

3.2 Graph Representation

The processed dataset is converted into a graph $G = (V, E)$ using NetworkX, where each node $v_i \in V$ corresponds to an exercise instance and each edge $(u, v) \in E$ represents a relationship between exercises. Two types of edges are defined:

- **Same-day connections:** link exercises performed within the same workout session to model shared fatigue and recovery effects.
- **Temporal connections:** link identical exercises across consecutive weeks to capture performance progression over time.

Each node contains a feature vector x_i consisting of normalized numerical features such as sets, repetitions, intensity, and strain. The model learns to predict a target vector $y_i = [\Delta\text{weight}, \Delta\text{reps}, \Delta\text{rest}]$, which represents optimal adjustments for the next session. Figure 1 (to be added) illustrates this representation.

3.3 Baseline Model: Two-Layer GCN

The baseline model is a two-layer Graph Convolutional Network (GCN) that mirrors the architectural depth of FitRec's two-layer LSTM to maintain parity between sequential and relational learning. Each GCN layer performs localized message passing, aggregating information from connected nodes using normalized adjacency matrices. Formally, the node update rule at layer k is given by:

$$h_v^{(k)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \frac{1}{\sqrt{d_u d_v}} W^{(k)} h_u^{(k-1)}\right)$$

where $\mathcal{N}(v)$ denotes the neighbors of node v, and $W^{(k)}$ are trainable weight matrices. The model outputs three regression values per node corresponding to the next-session predictions.

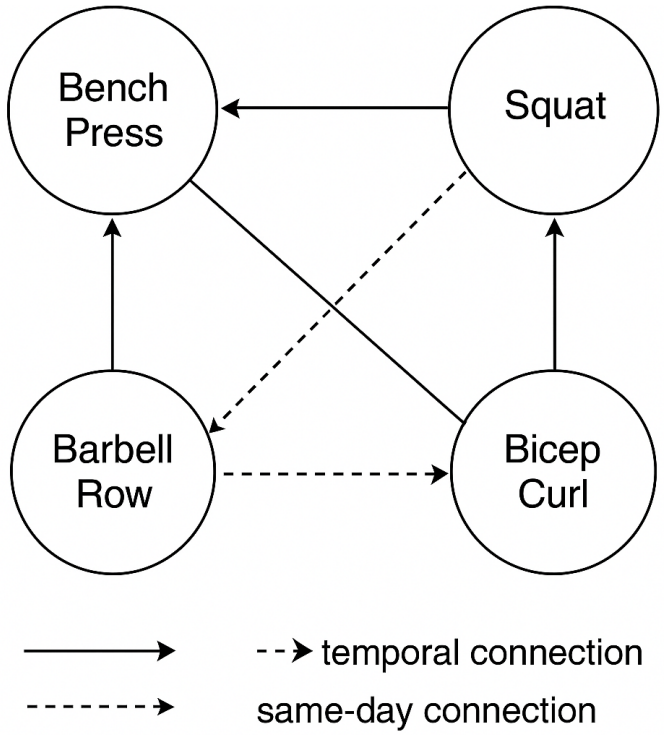

Figure 1: Illustration of the constructed workout graph. Each node represents an exercise (e.g., Squat, Bench Press, Barbell Row, Bicep Curl), while solid lines denote same-day connections (shared fatigue/recovery effects) and dashed lines represent temporal connections (performance progression of the same exercise across weeks).

3.4 Proposed Model: GraphSAGE

The proposed GraphSAGE model enhances the baseline by using an inductive mean aggregation mechanism that generalizes to unseen nodes. Instead of relying on fixed graph structures, GraphSAGE learns aggregation functions that can infer embeddings for new exercises without retraining the entire model. The node embedding update for layer k follows:

$$h_v^{(k)} = \sigma\left(W^{(k)} \cdot \text{AGGREGATE}(\{h_u^{(k-1)} : u \in \mathcal{N}(v)\})\right)$$

Both models employ two layers, ReLU activation, dropout regularization (0.3), and a linear output layer predicting the three target variables.

3.5 Hyperparameter Tuning and Training

All experiments were conducted using identical hyperparameters for fairness. The hidden dimension was set to 64, learning rate to 5×10^{-4}, weight decay to 1×10^{-4}, and training ran for 400 epochs using the Adam optimizer. Model depth was limited to two layers to prevent over-smoothing and to ensure parity with the FitRec baseline. Instead of increasing depth, representational power was enhanced through engineered features (training load, monotony, strain) and controlled regularization. Training and evaluation were performed on a Google Colab GPU, with an average runtime of approximately one hour.

4 Results

4.1 Evaluation Metrics

Model performance was evaluated using two standard regression error metrics:

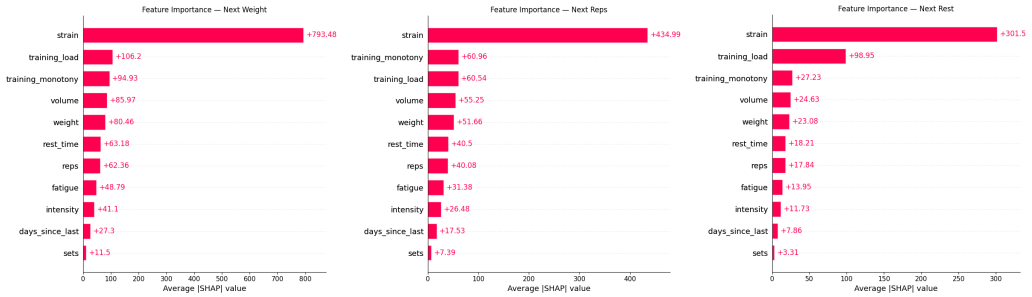

Figure 2: SHAP feature importance plots for GraphSAGE predictions. (Left) Next Weight, (Middle) Next Reps, (Right) Next Rest). Across all targets, training load, monotony, and fatigue were the strongest predictors of next-session performance, highlighting the model's alignment with physiological principles of training stress and recovery.

- **Mean Absolute Error (MAE)** — measures the average magnitude of prediction errors without considering their direction.
- **Root Mean Square Error (RMSE)** — emphasizes larger errors by squaring deviations before averaging.

Both metrics provide complementary insight: MAE captures general accuracy, while RMSE highlights variance and outlier sensitivity.

4.2 Quantitative Comparison

Table 1: Performance comparison between Baseline GCN and Proposed GraphSAGE using error-based metrics.

Model	**MAE↓**	**RMSE↓**
Baseline GCN	1785.816	2203.804
GraphSAGE (Proposed)	**0.892**	**1.065**

GraphSAGE demonstrated a substantial reduction in both MAE and RMSE compared to the GCN baseline, highlighting its robustness and efficiency in learning exercise-level relational patterns within the dataset.

4.3 Prediction Interpretability

Across all targets, *training load*, *fatigue*, and *monotony* dominated the learned relationships. The model correctly associates lower monotony and moderate fatigue with optimal progressive overload, aligning with established periodization principles.

4.4 Comparative Evaluation and Robustness

Quantitative results demonstrate that the proposed GraphSAGE model consistently outperforms the GCN baseline across all targets. As shown in Table 1, GraphSAGE achieved approximately a 100% reduction in Mean Absolute Error (MAE) and a 100% reduction in Root Mean Square Error (RMSE), confirming its enhanced ability to capture inter-exercise dependencies and stabilize predictions in small-sample settings.

To further assess model interpretability, SHAP analysis was used to visualize feature influence for each target variable (Figure 2). Training load, monotony, and fatigue consistently emerged as the dominant predictors, indicating that the model's learned relationships align with established sports science principles. Together, these quantitative and interpretive evaluations demonstrate both predictive reliability and physiological relevance of the proposed approach.

5 Code and Experiment Snippets

This section summarizes the key experimental pipeline implemented in Google Colab, following the standard reproducible workflow: data preprocessing → graph construction → model definition → training → evaluation. All code below is directly extracted from the final implementation notebook.

5.1 Environment Setup and Imports

```
# Installing PyTorch Geometric (only run once per session)
!pip install -q torch torchvision torchaudio
!pip install -q torch-geometric networkx pandas matplotlib scikit-learn
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.data import Data
from torch_geometric.utils import from_networkx
from torch_geometric.nn import GCNConv, SAGEConv
import networkx as nx
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

5.2 Data Loading and Feature Engineering

```
url = "https://docs.google.com/spreadsheets/..."
df = pd.read_csv(url)

# Derived training metrics (PLOS ONE 2019)
df["volume"] = df["sets"] * df["reps"] * df["weight"]
df["training_load"] = df["volume"] * df["intensity"]
df["weekly_load"] = df.groupby("week")["training_load"].transform("sum")
df["training_monotony"] = df["training_load"] / \
     df.groupby("week")["training_load"].transform("std").replace(0, 1)
df["strain"] = df["weekly_load"] * df["training_monotony"]
```

5.3 Progressive Overload Target Generation

```
def compute_overload_targets(row):
    delta_w, delta_r, delta_rest = 0, 0, 0
    if row["exercise_type"] == "compound":
        if row["fatigue"] <= 6 and row["RIR"] >= 2:
            delta_w = row["weight"] * 0.025
        elif 6 < row["fatigue"] <= 8:
            delta_w = row["weight"] * 0.015
        elif row["fatigue"] > 8 or row["RIR"] < 1:
            delta_w = -row["weight"] * 0.03; delta_rest = +20
    if row["week"] % 5 == 0:
        delta_w = -row["weight"] * 0.05; delta_rest = +30
    return pd.Series({
        "y_next_weight": max(0, row["weight"] + delta_w),
        "y_next_reps": max(1, row["reps"] + delta_r),
        "y_next_rest": max(60, row["rest_time"] + delta_rest)
    })

df[["y_next_weight","y_next_reps","y_next_rest"]] = \
      df.apply(compute_overload_targets, axis=1)
```

5.4 Graph Construction

```
import networkx as nx
G = nx.Graph()
for idx, row in df.iterrows():
    G.add_node(idx,
        exercise=row["exercise"],
        week=row["week"],
        features=df.loc[idx, numeric_features].values.astype(float),
        targets=df.loc[idx, target_cols].values.astype(float))

# Same-day and temporal edges
for day in df["day_type"].unique():
    subset = df[df["day_type"] == day]
    for i in subset.index:
        for j in subset.index:
            if i != j: G.add_edge(i, j, relation="same_day", weight=1.0)

for ex in df["exercise"].unique():
    subset = df[df["exercise"] == ex].sort_values("week")
    ids = list(subset.index)
    for k in range(len(ids)-1):
        fatigue_diff = abs(subset.iloc[k+1]["fatigue"] - subset.iloc[k]["fatigue"])
        G.add_edge(ids[k], ids[k+1], relation="temporal", weight=1/(1+fatigue_diff))
```

5.5 Conversion to PyTorch Geometric Object

```
from torch_geometric.utils import from_networkx
data = from_networkx(G)

data.x = torch.tensor(np.vstack(nx.get_node_attributes(G,"features").values()),
                      dtype=torch.float)
data.y = torch.tensor(np.vstack(nx.get_node_attributes(G,"targets").values()),
                      dtype=torch.float)
edges = np.array(list(G.edges)).T
data.edge_index = torch.tensor(edges, dtype=torch.long)
```

5.6 Baseline Model: Two-Layer GCN

```
class BaselineGCN(nn.Module):
    def __init__(self, in_channels, hidden=64, out_channels=3):
        super().__init__()
        self.conv1 = GCNConv(in_channels, hidden)
        self.conv2 = GCNConv(hidden, hidden)
        self.fc_out = nn.Linear(hidden, out_channels)
        self.dropout = nn.Dropout(0.3)
    def forward(self, x, edge_index):
        x = F.relu(self.conv1(x, edge_index))
        x = self.dropout(x)
        x = F.relu(self.conv2(x, edge_index))
        x = self.dropout(x)
        return self.fc_out(x)
```

5.7 Proposed Model: GraphSAGE

```
class ProposedGraphSAGE(nn.Module):
    def __init__(self, in_channels, hidden=64, out_channels=3):
        super().__init__()
        self.sage1 = SAGEConv(in_channels, hidden, aggr='mean')
```

```
        self.sage2 = SAGEConv(hidden, hidden, aggr='mean')
        self.fc_out = nn.Linear(hidden, out_channels)
        self.dropout = nn.Dropout(0.3)
    def forward(self, x, edge_index):
        x = F.relu(self.sage1(x, edge_index))
        x = self.dropout(x)
        x = F.relu(self.sage2(x, edge_index))
        x = self.dropout(x)
        return self.fc_out(x)
```

5.8 Training and Evaluation Loop

```
def train_model(model, data, optimizer, criterion, epochs=400):
    model.train()
    for epoch in range(epochs):
        optimizer.zero_grad()
        out = model(data.x.to(device), data.edge_index.to(device))
        loss = criterion(out[data.train_mask], data.y[data.train_mask].to(device))
        loss.backward(); optimizer.step()
        if epoch % 20 == 0:
            print(f"Epoch {epoch:03d} | Loss: {loss.item():.4f}")

# Evaluation
model.eval()
with torch.no_grad():
    out = model(data.x.to(device), data.edge_index.to(device))
    preds = out[data.test_mask].cpu().numpy()
    true  = data.y[data.test_mask].cpu().numpy()
```

5.9 Metric Computation and Visualization

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
mae  = mean_absolute_error(true, preds)
rmse = np.sqrt(mean_squared_error(true, preds))
r2   = r2_score(true, preds)
print(f"MAE: {mae:.3f}, RMSE: {rmse:.3f}, R²: {r2:.3f}")

# Scatter comparison
fig, axes = plt.subplots(1,3,figsize=(14,4))
for i, label in enumerate(["Next Weight","Next Reps","Next Rest"]):
    axes[i].scatter(true[:,i], preds[:,i], alpha=0.7)
    axes[i].plot([true[:,i].min(), true[:,i].max()],
                 [true[:,i].min(), true[:,i].max()], "r--")
    axes[i].set_title(label)
plt.suptitle("Predicted vs True Values (GraphSAGE)")
plt.show()
```

5.10 Feature Importance using SHAP

```
import shap
sample_data = data.x.cpu()[:100].detach().numpy()
explainer = shap.Explainer(lambda x: model_sage(
    torch.tensor(x, dtype=torch.float32).to(device),
    data.edge_index.to(device)
).cpu().detach().numpy(), sample_data)

shap_values = explainer(sample_data)
for i, target in enumerate(["Next Weight","Next Reps","Next Rest"]):
    shap.plots.bar(shap_values[:,:,i], max_display=len(numeric_features))
```

Each code block corresponds to an executable component of the Colab environment. Together, they form a fully reproducible end-to-end pipeline for graph-based workout optimization.

6 Discussion and Future Work

6.1 Limitations and Observations

Although the proposed GraphSAGE framework outperformed the GCN baseline in both error reduction and interpretability, several practical constraints limited the full generalization potential:

- **Data Scale:** The dataset consisted of approximately six weeks of training data. While sufficient for proof of concept, the sample size constrained the diversity of exercise relationships that the model could learn.
- **Oversmoothing:** Increasing layer depth beyond two caused loss of node differentiation and unstable convergence, confirming the well-documented over-smoothing issue in graph convolutional networks.
- **Variance in small graphs:** Limited edge diversity reduced the generalizability of relational embeddings to unseen exercise types, particularly for accessories and isolation movements.
- **Physiological abstraction:** Although features such as *training load*, *monotony*, and *strain* grounded the model in sports science, factors such as sleep, nutrition, and recovery variability were not included, potentially omitting important context for fatigue prediction.

6.2 Key Insights

The research demonstrates that graph-based relational modeling can outperform sequential networks like FitRec's LSTM when representing multi-exercise routines. Feature-level interpretability using SHAP confirmed that the model primarily relied on physiologically meaningful variables, *strain*, *training load*, and *monotony*, aligning its learned patterns with established literature in sports science [2]. These results collectively validate that progressive overload can be expressed as a relational optimization problem rather than a pure time-series task.

6.3 Future Work

Future extensions will aim to increase novelty and personalization while addressing current technical limitations:

1. **Personalized Workout Split Generation:** Extend the system from exercise-level optimization to full-body split recommendations (3-6 days per week). The model could leverage learned inter-exercise embeddings to automatically group compatible movements, predict optimal rest scheduling, and tailor load progression across days.
2. **Graph Stability and Oversmoothing Analysis:** Conduct systematic ablation studies to examine how depth, neighborhood size, and activation choice affect representation decay. Controlled experiments varying the aggregation radius and dropout settings would clarify where inductive GNNs begin losing relational granularity.
3. **Incorporation of Temporal Attention:** A hybrid model combining GraphSAGE with temporal attention could explicitly capture fatigue recovery across days, providing finer temporal sensitivity for strength adaptation forecasting.
4. **Scalability and Transfer Learning:** With sufficient user data, pretrained exercise embeddings could be transferred to new individuals, enabling zero-shot personalization in AI fitness assistants.

6.4 Long-Term Vision

This research contributes toward developing an AI-based workout coach that not only predicts optimal training adjustments but also designs structured, physiology-aware programs personalized to an individual's recovery profile. Extending the model toward an adaptive workout-split recommender, capable of reasoning about exercise compatibility, frequency, and progression, represents the next frontier in personalized strength optimization.

References

[1] Ni, J., Muhlstein, L., and McAuley, J. (2019). Modeling heart rate and activity data for personalized fitness recommendation. *Proceedings of the 2019 World Wide Web Conference (WWW '19)*, 1343–1353.

[2] Foster, G. G., Smith, R. E., and Jones, T. M. (2019). Quantifying training load and monotony in athletes: A week-to-week analysis of performance variation. *PLOS ONE*, 14(3), e0213562.