National College of Ireland

Symptom-Treatment Knowledge Graphs for Contagious and Chronic Disease Profiling

MSc Research Project
MSc. Data Analytics

Muralidhar Kukkala
Student ID: 23346795

School of Computing
National College of Ireland

Supervisor: Shubham Subhnil

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Muralidhar Kukkala

Student ID: 23346795

Programme: MSc in Data Analytics **Year:** 2025-2026

Module: MSc Research Project

Lecturer: Shubham Subhnil

Submission Due Date: 11-12-2025

Project Title: Symptom-Treatment Knowledge Graphs for Contagious and Chronic Disease Profiling

Word Count: **845 Page Count: 15**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Muralidhar Kukkala

Date: 11/12/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Symptom-Treatment Knowledge Graphs for Contagious and Chronic Disease Profiling

Muralidhar Kukkala
23346795

1 Introduction

This configuration manual describes the complete operational setup required to run the disease profiling framework that integrates biomedical text processing, symptom–treatment canonicalization, knowledge graph construction, community detection, machine-learning classification, BioBERT embeddings, and graph-neural-network (GNN) learning with explainability (SHAP & GNNExplainer).

The manual ensures that the system executes consistently on any local or cloud-based Python environment, including Jupyter, and colab.

2 System Requirements

2.1 Hardware

- Minimum 8 GB RAM (16 GB recommended)
- GPU recommended for BioBERT & GNN (CUDA-enabled)

2.2 Operating System

Supported platforms:

- Windows 10/11

3 Software Stack

3.1 Python Version

Python 3.9 – 3.12

3.2 Mandatory Libraries

Component	Libraries Used
Data handling	pandas, numpy
Plotting	matplotlib, seaborn
NLP	spaCy, SciSpaCy, BioBERT (Transformers)
Machine Learning	sklearn, xgboost
Deep Learning	PyTorch

Graph Algorithms	networkx
GNNs	PyTorch Geometric
Explainability	SHAP, GNNExplainer

4 Dataset Requirements

The system expects a CSV file with the following schema:

Column Name	Description
Name	Disease name
Symptoms	Comma-separated symptom list
Treatments	Comma-separated treatment list
Chronic	0/1 or Yes/No
Contagious	0/1 or Yes/No
Disease_Code	Optional ICD-style code

All text inputs may contain inconsistent formatting—canonicalization handles cleaning.

5 Execution Workflow Overview

5.1 Load & Inspect Dataset

Purpose:

- Read CSV
- Inspect missing values
- Validate schema

```
# ------------ 0) Load ------------
FILE_PATH = "Diseases_Symptoms.csv"
df = pd.read_csv(FILE_PATH)

print("Shape:", df.shape)
print("Columns:", df.columns.tolist())
display(df.head())

print("\n=== INFO ===")
print(df.info())

print("\n=== Missing values per column ===")
display(df.isna().sum())
```

Figure 1: Loading the Dataset and Inspection

Outputs:

- `df` (raw data)
- Shape, head(), dtype report
- Null-value diagnostics

5.2 Data Cleaning & Preprocessing

```
# ------------ 1) Basic cleaning ------------
# Trim text columns
for c in ["Name", "Symptoms", "Treatments", "Disease_Code"]:
    if c in df.columns:
        df[c] = df[c].astype(str).str.strip()

# Normalize Contagious/Chronic to 0/1 (handles 1/0, yes/no, true/false, y/n, strings)
def to01(x):
    if pd.isna(x): return 0
    s = str(x).strip().lower()
    return 1 if s in {"1","true","yes","y","t"} else 0

df["Contagious"] = df["Contagious"].apply(to01).astype(int)
df["Chronic"]    = df["Chronic"].apply(to01).astype(int)

# Parse Symptoms/Treatments as lists (split by comma)
def split_list(x):
    if pd.isna(x): return []
    parts = [p.strip().lower() for p in str(x).split(",")]
    return [p for p in parts if p]  # remove empty

df["Symptoms_list"]   = df["Symptoms"].apply(split_list)
df["Treatments_list"] = df["Treatments"].apply(split_list)

# Simple per-row counts
df["n_symptoms"]   = df["Symptoms_list"].apply(len)
df["n_treatments"] = df["Treatments_list"].apply(len)
```

Figure 2: Dataset Cleaning

Operations performed:

- Trim whitespace
- Normalize boolean labels (Yes/No → 0/1)
- Convert symptom/treatment text → Python lists
- Generate basic statistics (#symptoms, #treatments)

Outputs:

- Cleaned DataFrame df
- Ready for exploratory analysis

5.3 Exploratory Data Analysis

This module generates:

- Top disease distributions
- Symptom & treatment frequencies
- Prevalence of chronicity and contagiousness
- Visual summaries (matplotlib + seaborn)

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style="whitegrid", font_scale=1.05)

fig, axes = plt.subplots(3, 2, figsize=(16, 18))
fig.suptitle("Exploratory Data Analysis – Disease Dataset", fontsize=18, weight="bold")

# ----------------------------------------
# 1. Top 15 Diseases
# ----------------------------------------
df["Name"].value_counts().head(15).sort_values().plot(
    kind="barh",
    color=sns.color_palette("viridis", 15),
    ax=axes[0,0]
)
axes[0,0].set_title("Top 15 Diseases by Row Count", weight="bold")
axes[0,0].set_xlabel("Count")
axes[0,0].set_ylabel("Disease")
```

```
# ----------------------------------------
# 2. Contagious Distribution
# ----------------------------------------
sns.barplot(
    x=df["Contagious"].value_counts().sort_index().index,
    y=df["Contagious"].value_counts().sort_index().values,
    palette="coolwarm",
    ax=axes[0,1]
)
axes[0,1].set_title("Contagious Distribution (0 = No, 1 = Yes)", weight="bold")
axes[0,1].set_xticks([0,1])
axes[0,1].set_xticklabels(["No (0)", "Yes (1)"])
axes[0,1].set_ylabel("Rows")
axes[0,1].set_xlabel("")

# ----------------------------------------
# 3. Chronic Distribution
# ----------------------------------------
sns.barplot(
    x=df["Chronic"].value_counts().sort_index().index,
    y=df["Chronic"].value_counts().sort_index().values,
    palette="magma",
    ax=axes[1,0]
)
axes[1,0].set_title("Chronic Distribution (0 = No, 1 = Yes)", weight="bold")
axes[1,0].set_xticks([0,1])
axes[1,0].set_xticklabels(["No (0)", "Yes (1)"])
axes[1,0].set_ylabel("Rows")
axes[1,0].set_xlabel("")
```

```
# ----------------------------------------
# 4. Top 20 Symptoms
# ----------------------------------------
sym_counts.head(20).sort_values().plot(
    kind="barh",
    color=sns.color_palette("crest", 20),
    ax=axes[1,1]
)
axes[1,1].set_title("Top 20 Symptoms", weight="bold")
axes[1,1].set_xlabel("Count")
axes[1,1].set_ylabel("Symptom")

# ----------------------------------------
# 5. Top 20 Treatments
# ----------------------------------------
# Merge last row into one big axis
fig.delaxes(axes[2,1])  # remove empty axis
big_ax = axes[2,0]

treat_counts.head(20).sort_values().plot(
    kind="barh",
    color=sns.color_palette("rocket", 20),
    ax=big_ax
)
big_ax.set_title("Top 20 Treatments", weight="bold")
big_ax.set_xlabel("Count")
big_ax.set_ylabel("Treatment")

plt.tight_layout(rect=[0, 0, 1, 0.97])
plt.show()
```

Figure 3: Exploratory Data Analysis

5.4 Disease-Level Aggregation

The system aggregates raw rows into **unique disease profiles**, combining:

- unified symptom sets
- unified treatment sets
- majority labels for Chronic/Contagious

Generated fields:

- sym_text
- trx_text
- all_text

These are used for TF-IDF vectorization and later for BioBERT embeddings.

```
# ------------ 5) Profiling by label (simple and useful) ------------
# Most common symptoms among contagious=1 vs chronic=1
def top_by_flag(flag_col, list_col, top=15):
    tmp = df.loc[df[flag_col]==1].explode(list_col)[list_col]
    tmp = tmp[tmp.notna() & (tmp != "")]
    return tmp.value_counts().head(top)

print("\n=== Top Symptoms in Contagious=1 ===")
display(top_by_flag("Contagious", "Symptoms_list", 15))

print("\n=== Top Symptoms in Chronic=1 ===")
display(top_by_flag("Chronic", "Symptoms_list", 15))

print("\n=== Top Treatments in Contagious=1 ===")
display(top_by_flag("Contagious", "Treatments_list", 15))

print("\n=== Top Treatments in Chronic=1 ===")
display(top_by_flag("Chronic", "Treatments_list", 15))
```

```
# Simple gap view: which symptoms differ most between contagious vs not
sym_in  = df.loc[df["Contagious"]==1].explode("Symptoms_list")["Symptoms_list"].value_counts(normalize=True)
sym_out = df.loc[df["Contagious"]==0].explode("Symptoms_list")["Symptoms_list"].value_counts(normalize=True)
gap = (sym_in - sym_out).abs().dropna().sort_values(ascending=False).head(15)

print("\n=== Symptoms with largest prevalence gap (Contagious vs Not) ===")
display(gap)
```

```
# ---------------------------
# Aggregate to DISEASE level
# ---------------------------
agg = (
    df.groupby("Name")
      .agg({
          "Disease_Code":"first",
          "Contagious": lambda s: int(s.mean() >= 0.5),
          "Chronic":    lambda s: int(s.mean() >= 0.5),
          "Symptoms_list":   lambda ll: sorted(set(x for L in ll for x in L)),
          "Treatments_list": lambda ll: sorted(set(x for L in ll for x in L))
      })
      .reset_index()
)

# Easy-to-use text for vectorizers
agg["sym_text"] = agg["Symptoms_list"].apply(lambda xs: ", ".join(xs))
agg["trx_text"] = agg["Treatments_list"].apply(lambda xs: ", ".join(xs))
agg["all_text"] = agg["sym_text"] + " || " + agg["trx_text"]

print("Disease-level shape:", agg.shape)
display(agg.head())
```

Figure 4: Unique Disease Profiles

5.5 Supervised Learning (Baseline Models)

The system trains and evaluates:

- Logistic Regression (Contagious prediction)
- Logistic Regression (Chronic prediction)

Each run outputs:

- Accuracy
- Precision
- Recall
- F1-score
- Confusion matrix plot
- Bar chart of metrics

```
# ---------------------------
def run_task(X, y, title):
    Xtr, Xte, ytr, yte = train_test_split(
        X, y, test_size=0.25, random_state=42, stratify=y
    )
    clf = LogisticRegression(max_iter=3000, class_weight="balanced")
    clf.fit(Xtr, ytr)
    ypr = clf.predict(Xte)

    acc = accuracy_score(yte, ypr)
    pre = precision_score(yte, ypr, zero_division=0)
    rec = recall_score(yte, ypr, zero_division=0)
    f1  = f1_score(yte, ypr, zero_division=0)
    print(f"\n{title} -> Accuracy={acc:.3f}  Precision={pre:.3f}  Recall={rec:.3f}  F1={f1:.3f}")

    # Confusion Matrix (clear for non-technical readers)
    fig, ax = plt.subplots(figsize=(4.5,4))
    ConfusionMatrixDisplay(confusion_matrix(yte, ypr)).plot(ax=ax, colorbar=False)
    ax.set_title(f"{title} — Confusion Matrix")
    plt.tight_layout(); plt.show()

    # Metrics bar (labels on bars)
    vals = [acc, pre, rec, f1]
    labs = ["Accuracy","Precision","Recall","F1"]
    plt.figure(figsize=(6,4))
    plt.bar(labs, vals)
    for i,v in enumerate(vals): plt.text(i, v+0.02, f"{v:.2f}", ha="center")
    plt.ylim(0, 1.05)
    plt.title(f"{title} — Performance")
    plt.ylabel("Score")
    plt.tight_layout(); plt.show()
```

Figure 5: Baseline Model Implementation

5.6 Canonicalization via SciSpaCy

Purpose:

- Extract biomedical entities
- Normalize symptom/treatment terms (NER tokenization)
- Reduce duplicates and surface-form variations

Where available:

- Uses SciSpaCy (en_core_sci_sm)
  Fallback:
- Uses custom regex-based tokenizer.

Outputs:

- sym_canon
- trx_canon

These feed directly into knowledge graph construction.

```
def scispacy_extract(text):
    ents = []
    if nlp is None:
        return simple_tokenize_and_norm(text)
    doc = nlp(str(text))
    # take entity spans (text) and normalize whitespace/lowercase
    for ent in doc.ents:
        e = re.sub(r"[^a-z0-9\s\-]", "", ent.text.lower()).strip()
        if e: ents.append(e)
    if not ents:
        # fallback to simple split
        return simple_tokenize_and_norm(text)
    return sorted(set(ents))

# Run canonicalization and replace agg columns
if 'agg' not in globals():
    raise RuntimeError("The disease-level table `agg` is required. Run previous cells first.")

agg = agg.copy()  # work on a copy
agg["sym_canon"] = agg["sym_text"].apply(scispacy_extract)
agg["trx_canon"] = agg["trx_text"].apply(scispacy_extract)

print("Sample canonical symptoms/treatments:")
display(agg[["Name","sym_text","sym_canon","trx_canon"]].head(4))
```

Figure 6: Canocalization

5.7 Knowledge Graph Construction

A heterogeneous graph is built with:

5.7.1 Node Types

- Disease nodes
- Symptom nodes
- Treatment nodes

5.7.2 Edge Types

- has_symptom
- treated_with

Graph metrics produced:

- Node count
- Edge count
- Degree statistics

A neighborhood visualization for a selected disease is generated.

5.8 Community Detection & Disease Clustering

Two alternative community-detection strategies:

5.8.1 1. Greedy Modularity Communities (NetworkX)

- Creates disease clusters from symptom overlap.

5.8.2 2. Louvain (python-louvain)

- More robust for large biomedical graphs.
- Outputs modularity score Q.

5.8.3 Visual outputs:

- Bar charts of top communities
- PCA scatter plot of diseases colored by community

```
# ----------------------------
# Disease-Disease projection by shared symptoms
# ----------------------------
diseases = [n for n in G.nodes() if G.nodes[n].get("kind")=="disease"]
proj = nx.Graph(); proj.add_nodes_from(diseases)

# Map disease -> symptom set
dz_sym = {d: {nbr for _, nbr, dat in G.out_edges(d, data=True) if dat.get("rel")=="has_symptom"} for d in diseases}
for i,d1 in enumerate(diseases):
    for d2 in diseases[i+1:]:
        w = len(dz_sym[d1] & dz_sym[d2])
        if w > 0:
            proj.add_edge(d1, d2, weight=w)

from networkx.algorithms.community import greedy_modularity_communities, quality
comms = list(greedy_modularity_communities(proj, weight="weight"))
Q = quality.modularity(proj, comms, weight="weight")
print(f"Communities: {len(comms)} | Modularity: {Q:.3f}")

# Bar chart of top community sizes (non-technical summary)
sizes = sorted([len(c) for c in comms], reverse=True)[:10]
plt.figure(figsize=(6,4))
plt.bar([f"C{i+1}" for i in range(len(sizes))], sizes)
plt.title("Top community sizes (disease clusters)")
plt.ylabel("#Diseases")
plt.tight_layout(); plt.show()
```

Figure 7: Disease Clustering

5.9 Embeddings (TF-IDF or BioBERT)

The system automatically selects the best available embedding method.

5.9.1 Case A: BioBERT Available

- Text encoded via BioBERT mean pooling
- High semantic fidelity

5.9.2 Case B: BioBERT Unavailable

- TF-IDF vectorization
- Fully compatible fallback

```
transformers = ensure("transformers")
torch = ensure("torch")
USE_BIOBERT = transformers is not None and torch is not None

if USE_BIOBERT:
    from transformers import AutoTokenizer, AutoModel
    try:
        tok = AutoTokenizer.from_pretrained("dmis-lab/biobert-base-cased-v1.1")
        mdl = AutoModel.from_pretrained("dmis-lab/biobert-base-cased-v1.1")
        mdl.eval()

        def mean_pool(text):
            if not text or str(text).strip()=="":
                return np.zeros(mdl.config.hidden_size, dtype="float32")
            with torch.no_grad():
                inp = tok(str(text), return_tensors="pt", truncation=True, max_length=128)
                out = mdl(**inp).last_hidden_state.mean(1).squeeze().cpu().numpy().astype("float32")
            return out

        print("Encoding disease texts with BioBERT (symptoms+treatments)...")
        emb_sym = np.stack([mean_pool(t) for t in agg["sym_text"]])
        emb_trx = np.stack([mean_pool(t) for t in agg["trx_text"]])
        X_dense_bio = (emb_sym + emb_trx) / 2.0
        BIO_SOURCE = "BioBERT"
    except Exception as e:
        print(f"BioBERT unavailable at runtime ({e}). Falling back to TF-IDF.")
        USE_BIOBERT = False

if not USE_BIOBERT:
    # TF-IDF fallback (same as earlier but rebuilt here for completeness)
    vec_fallback = TfidfVectorizer(ngram_range=(1,2), min_df=2, max_features=4000)
    X_sparse = vec_fallback.fit_transform(agg["all_text"])
    X_dense_bio = X_sparse.toarray().astype("float32")
    BIO_SOURCE = "TF-IDF (fallback)"
```

Figure 8: BioBert Implementation

5.10 Machine Learning with Cross-Validation

Models evaluated via 5-Fold CV:

- Random Forest
- XGBoost

Metrics printed:

- Accuracy

- Precision
- Recall
- F1-score

Plot bar charts for each fold average.

```
def cv_report_clf(clf, X, y, title):
    cv = StratifiedKFold(5, shuffle=True, random_state=42)
    scores = cross_validate(clf, X, y, cv=cv,
                            scoring=["accuracy","precision","recall","f1"],
                            n_jobs=-1, return_train_score=False)
    metrics = {m: float(scores[f"test_{m}"].mean()) for m in ["accuracy","precision","recall","f1"]}
    print(title, "→", {k: round(v,3) for k,v in metrics.items()})
    show_metric_bar(title, metrics)

y_cont = agg["Contagious"].values
y_chro = agg["Chronic"].values

# RandomForest (tree-based, SHAP-friendly)
rf_cont = RandomForestClassifier(n_estimators=400, class_weight="balanced", random_state=42, n_jobs=-1)
rf_chro = RandomForestClassifier(n_estimators=400, class_weight="balanced", random_state=42, n_jobs=-1)
cv_report_clf(rf_cont, X_dense_bio, y_cont, "CV - RandomForest (Contagious)")
cv_report_clf(rf_chro, X_dense_bio, y_chro, "CV - RandomForest (Chronic)")

# XGBoost (optional)
xgb = ensure("xgboost")
if xgb:
    from xgboost import XGBClassifier
    xgb_cont = XGBClassifier(
        n_estimators=600, learning_rate=0.05, max_depth=6, subsample=0.9, colsample_bytree=0.9,
        reg_lambda=1.0, random_state=42, n_jobs=-1, objective="binary:logistic", eval_metric="logloss",
        tree_method="hist"
    )
    xgb_chro = XGBClassifier(
        n_estimators=600, learning_rate=0.05, max_depth=6, subsample=0.9, colsample_bytree=0.9,
        reg_lambda=1.0, random_state=42, n_jobs=-1, objective="binary:logistic", eval_metric="logloss",
        tree_method="hist"
    )
    cv_report_clf(xgb_cont, X_dense_bio, y_cont, "CV - XGBoost (Contagious)")
```

Figure 9: Machine Learning with Cross-Validation

5.11 Module 11 — SHAP Explainability

SHAP analysis explains:

- Which biomedical terms influence contagion classification
- Which features drive chronic disease classification

Outputs:

- SHAP summary plots
- SHAP bar plots

```
# ------------------------------------------------------
# SHAP explanations for tree models (robust, actual plots)
# ------------------------------------------------------
shap = ensure("shap")
if shap:
    import shap as _shap

    # Fit RF on full disease vectors (BioBERT or TF-IDF fallback)
    rf_cont_fit = RandomForestClassifier(
        n_estimators=400,
        class_weight="balanced",
        random_state=42,
        n_jobs=-1
    ).fit(X_dense_bio, y_cont)

    rf_chro_fit = RandomForestClassifier(
        n_estimators=400,
        class_weight="balanced",
        random_state=42,
        n_jobs=-1
    ).fit(X_dense_bio, y_chro)

    # Small sample for speed
    rng = np.random.RandomState(42)
    samp_idx = rng.choice(len(agg), size=min(300, len(agg)), replace=False)
    X_sample = X_dense_bio[samp_idx]

    # --------------------------------
    # Robust SHAP value extraction
    # --------------------------------
    def get_shap_values(model, X):
        """
        Return 2D SHAP array (n_samples, n_features) and explainer,
```

Figure 10: SHAP Implementation

5.12 GNN Pipeline

5.12.1 Graph Setup

- Disease + symptom + treatment nodes embedded
- Edges mapped to PyTorch Geometric format
- Masks created for train/test separation

5.12.2 Model

A 2-layer GCN:

GCNConv → ReLU → GCNConv → Softmax

5.12.3 Training

- Optimizer: Adam
- Loss: CrossEntropy
- Epochs: 80

Outputs:

- Test accuracy, precision, recall, F1

Graph-based prediction models exploit relational biomedical structure.

```
# -------------------------
# Define small GCN
# -------------------------
class SmallGCN(torch.nn.Module):
    def __init__(self, in_channels, hidden=64, out_channels=2):
        super().__init__()
        self.conv1 = GCNConv(in_channels, hidden)
        self.conv2 = GCNConv(hidden, out_channels)
        self.act = torch.nn.ReLU()

    def forward(self, x, edge_index):
        h = self.act(self.conv1(x, edge_index))
        h = self.conv2(h, edge_index)
        return h

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SmallGCN(feature_dim, hidden=64, out_channels=2).to(device)
data = data.to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)
loss_fn = torch.nn.CrossEntropyLoss()

# -------------------------
# Train GCN
# -------------------------
model.train()
epochs = 80
for epoch in range(1, epochs + 1):
    optimizer.zero_grad()
    out = model(data.x, data.edge_index)
    loss = loss_fn(out[data.train_mask], data.y[data.train_mask])
    loss.backward()
    optimizer.step()
```

Figure 11: GCN Model Implementation

5.13 GNNExplainer

Purpose:

- Identify most influential symptoms/treatments for disease classification.

Outputs:

- Subgraph highlighting important nodes
- Edge thickness proportional to importance weight

```python
# -------------------------
# GNNExplainer: explain a single disease node (choose one from test)
# -------------------------
try:
    explainer = GNNExplainer(model, epochs=200, return_type='log_prob')

    # choose a test disease node to explain
    node_to_explain = test_idx[0]
    node_to_explain = torch.tensor(node_to_explain, dtype=torch.long).to(device)

    model.eval()
    node_feat_mask, edge_mask = explainer.explain_node(
        node_to_explain.item(), data.x, data.edge_index
    )

    # Get top-k edges by importance
    edge_mask = edge_mask.cpu().detach().numpy()
    edge_index_np = data.edge_index.cpu().numpy()
    edges = list(zip(edge_index_np[0], edge_index_np[1], edge_mask))

    top_k = min(50, len(edges))
    edges_sorted = sorted(edges, key=lambda x: x[2], reverse=True)[:top_k]

    important_nodes = set()
    for u, v, w in edges_sorted:
        important_nodes.add(u)
        important_nodes.add(v)

    # map back to original node ids
    imp_nodes = [all_nodes[n] for n in important_nodes]
    subG = G.subgraph(imp_nodes).copy()

    # importance weights dict (for edge widths)
    edge_weights = {}
```

Figure12: GNN Explainer

This is the interpretability layer of the knowledge graph model.

References

Abe, S., Tago, S., Yokoyama, K., Ogawa, M., Takei, T., Imoto, S. and Fuji, M., 2023. Explainable AI for estimating pathogenicity of genetic variants using large-scale knowledge graphs. *Cancers*, *15*(4), p.1118. https://doi.org/10.3390/cancers15041118

Ben Abdessalem Karaa, W., Alkhammash, E.H. and Bchir, A., 2021. Drug disease relation extraction from biomedical literature using NLP and machine learning. *Mobile Information Systems*, *2021*(1), p.9958410. https://doi.org/10.1155/2021/9958410

Chen, J., Dong, H., Hastings, J., Jiménez-Ruiz, E., López, V., Monnin, P., Pesquita, C., Škoda, P. and Tamma, V., 2023. Knowledge graphs for the life sciences: Recent developments, challenges and opportunities. *arXiv preprint arXiv:2309.17255*. https://doi.org/10.4230/TGDK.1.1.5

Cho, H.N., Jun, T.J., Kim, Y.H., Kang, H., Ahn, I., Gwon, H., Kim, Y., Seo, J., Choi, H., Kim, M. and Han, J., 2024. Task-specific transformer-based language models in health care: scoping review. *JMIR Medical Informatics*, *12*, p.e49724. https://doi.org/10.2196/49724