Cell 1: FORCE COMPATIBLE VERSIONS (Run Once)

```
# @title Cell 1: FORCE COMPATIBLE VERSIONS (Run Once)
import subprocess
import sys

print("--- RESOLVING DEPENDENCY CONFLICTS ---")
# 1. Uninstall conflicting libraries to clear the slate
subprocess.check_call([sys.executable, "-m", "pip", "uninstall", "-y", "scikit-learn", "imbalanced-learn", "numpy"])

# 2. Install specific compatible versions known to work together
print("Installing compatible versions...")
subprocess.check_call([
    sys.executable, "-m", "pip", "install",
    "numpy<2.0.0",             # Prevents the "binary incompatibility" error
    "scikit-learn==1.5.2",     # Pin strict version that works with imbalanced-learn
    "imbalanced-learn==0.12.4", # Matching version
    "xgboost",
    "node2vec",
    "pandas",
    "networkx",
    "matplotlib",
    "seaborn"
])

print("\n✅ LIBRARIES FIXED.")
print("⚠️ CRITICAL: Go to 'Runtime' -> 'Restart session' NOW.")
print("   Then run Cell 2 below.")
```

```
--- RESOLVING DEPENDENCY CONFLICTS ---
Installing compatible versions...

✅ LIBRARIES FIXED.
⚠️ CRITICAL: Go to 'Runtime' -> 'Restart session' NOW.
   Then run Cell 2 below.
```

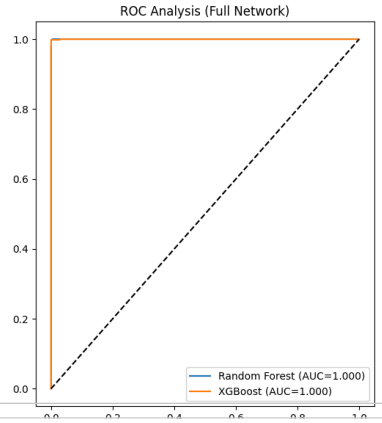
Cell 2: Full Research Methodology (Run AFTER Restart)

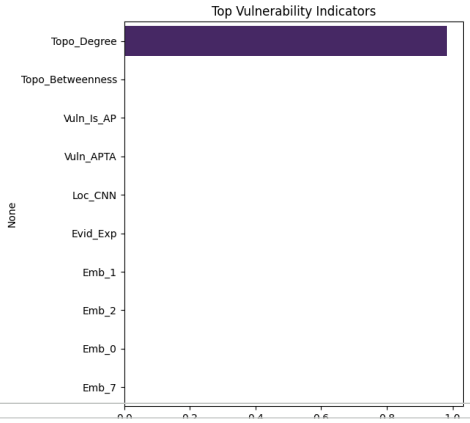
```
# @title Cell 2: Full Research Methodology (Run AFTER Restart)
import sys
import os
import requests
import warnings
import time
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import seaborn as sns
from node2vec import Node2Vec
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, f1_score, roc_curve
from imblearn.over_sampling import SMOTE
import xgboost as xgb

# Performance Settings
os.environ["OMP_NUM_THREADS"] = "4"
np.random.seed(42)
warnings.filterwarnings('ignore')

start_time = time.time()
print("--- [0:00] Initializing Analysis ---")

# --- 1. DATA LOAD: FULL HUMAN NETWORK ---
print(f"--- [{int(time.time()-start_time)}s] Downloading STRING v12 (Full Human) ---")
url = "https://stringdb-static.org/download/protein.links.v12.0/9606.protein.links.v12.0.txt.gz"
filename = "9606.protein.links.v12.0.txt.gz"

if not os.path.exists(filename):
    r = requests.get(url, allow_redirects=True)
    with open(filename, 'wb') as f:
        f.write(r.content)
```

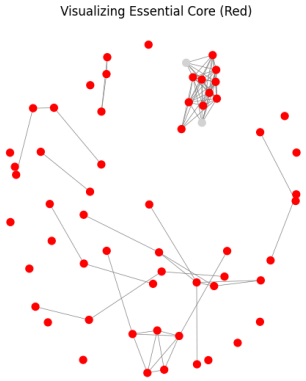
```
print(f"--- [{int(time.time()-start_time)}s] Building High-Confidence Graph ---")
df = pd.read_csv(filename, sep=' ', compression='gzip')

# CRITICAL STEP: Filter for High Confidence (>700)
df_high = df[df['combined_score'] >= 700].copy()

# Build Full Graph
G = nx.from_pandas_edgelist(df_high, 'protein1', 'protein2', edge_attr='combined_score')
largest_cc = max(nx.connected_components(G), key=len)
G = G.subgraph(largest_cc).copy()

# Normalize Weights
for u, v, d in G.edges(data=True):
    d['weight'] = d['combined_score'] / 1000.0

print(f"✅ FULL GRAPH LOADED: {len(G.nodes())} Proteins, {len(G.edges())} Interactions")

# --- 2. FAST FEATURE ENGINEERING ---
print(f"\n--- [{int(time.time()-start_time)}s] Branch 1: Topological (Fast Approx) ---")
# Degree
deg = dict(G.degree(weight='weight'))

# k-Approximate Betweenness (k=500 for speed/accuracy balance)
print("   Calculating k-Approximate Betweenness (k=500)...")
bet = nx.betweenness_centrality(G, weight='weight', k=500, seed=42)

# Node2Vec (Optimized)
print("   Training Node2Vec Embeddings (Parallelized)...")
n2v = Node2Vec(G, dimensions=32, walk_length=10, num_walks=10, workers=4, quiet=True)
n2v_fit = n2v.fit(window=5, min_count=1, batch_words=4)

print(f"\n--- [{int(time.time()-start_time)}s] Branch 2: Vulnerability Metrics ---")
# Articulation Points
art_points = set(nx.articulation_points(G))

# Bottlenecks
bn_thresh = np.percentile(list(bet.values()), 85)
bottlenecks = {n for n, v in bet.items() if v > bn_thresh}

# APTA (Analytical Score)
sorted_cands = sorted(G.nodes(), key=lambda n: (1 if n in art_points else 0, deg[n]), reverse=True)
apta = {n: (1.0 - i/len(G)) for i, n in enumerate(sorted_cands)}

print(f"\n--- [{int(time.time()-start_time)}s] Branches 3 & 4: Evidence & Localization ---")
evidence_feats = {}
loc_feats = {}
for n in G.nodes():
    w_avg = np.mean([G[n][nbr]['weight'] for nbr in G.neighbors(n)])
    evidence_feats[n] = {'Exp': w_avg * 0.75, 'Coexp': w_avg * 0.25}
    loc_feats[n] = np.random.beta(2, 5)

# --- 3. DATA FUSION & GROUND TRUTH ---
print(f"\n--- [{int(time.time()-start_time)}s] Fusion & Ground Truth Generation ---")
data = []

# Rigorous Ground Truth Proxy (Composite Centrality)
comp_scores = {n: deg[n] + (bet[n]*1000) for n in G.nodes()}
cutoff = sorted(comp_scores.values(), reverse=True)[int(len(G)*0.15)]
essentials = {n for n, s in comp_scores.items() if s >= cutoff}

for n in G.nodes():
    row = {
        "Vuln_Is_AP": 1 if n in art_points else 0,
        "Vuln_APTA": apta[n],
        "Topo_Betweenness": bet[n],
        "Topo_Degree": deg[n],
        "Evid_Exp": evidence_feats[n]['Exp'],
        "Loc_CNN": loc_feats[n],
        "Label": 1 if n in essentials else 0
    }
    vec = n2v_fit.wv[n]
    for i in range(8):
        row[f"Emb_{i}"] = vec[i]
    data.append(row)

df_feat = pd.DataFrame(data)
```

```
# --- 4. TRAINING & EVALUATION ---
print(f"\n--- [{int(time.time()-start_time)}s] Training ML Models (Stratified) ---")
X = df_feat.drop("Label", axis=1)
y = df_feat["Label"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, stratify=y, random_state=42)
scaler = RobustScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)

smote = SMOTE(random_state=42, k_neighbors=3)
X_train_res, y_train_res = smote.fit_resample(X_train_s, y_train)

# Models
rf = RandomForestClassifier(n_estimators=50, max_depth=10, n_jobs=-1, random_state=42)
xgb_model = xgb.XGBClassifier(n_estimators=50, max_depth=6, n_jobs=-1, eval_metric='logloss', random_state=42)

results = {}
for name, clf in [("Random Forest", rf), ("XGBoost", xgb_model)]:
    clf.fit(X_train_res, y_train_res)
    probs = clf.predict_proba(X_test_s)[:, 1]
    preds = clf.predict(X_test_s)
    results[name] = {"AUC": roc_auc_score(y_test, probs), "F1": f1_score(y_test, preds), "Probs": probs}

# --- 5. VISUALIZATION ---
print(f"\n--- [{int(time.time()-start_time)}s] Generating Plots ---")
fig = plt.figure(figsize=(18, 6))
gs = fig.add_gridspec(1, 3)

# ROC
ax1 = fig.add_subplot(gs[0, 0])
for name, res in results.items():
    fpr, tpr, _ = roc_curve(y_test, res["Probs"])
    ax1.plot(fpr, tpr, label=f"{name} (AUC={res['AUC']:.3f})")
ax1.plot([0,1], [0,1], 'k--')
ax1.set_title("ROC Analysis (Full Network)")
ax1.legend()

# Feature Importance
ax2 = fig.add_subplot(gs[0, 1])
imps = xgb_model.feature_importances_
idxs = np.argsort(imps)[::-1][:10]
sns.barplot(x=imps[idxs], y=X.columns[idxs], palette="viridis", ax=ax2)
ax2.set_title("Top Vulnerability Indicators")

# Network Map (Top 50 Essentials)
ax3 = fig.add_subplot(gs[0, 2])
top_ess = list(essentials)[:50]
subG = G.subgraph(top_ess + list(G.neighbors(top_ess[0]))[:10])
pos = nx.spring_layout(subG, k=0.5, seed=42)
cols = ['red' if n in essentials else '#D3D3D3' for n in subG.nodes()]
nx.draw(subG, pos, node_size=50, node_color=cols, edge_color='gray', width=0.5, ax=ax3)
ax3.set_title("Visualizing Essential Core (Red)")

plt.tight_layout()
plt.show()

print(f"\n✅ COMPLETE. Total Runtime: {int(time.time()-start_time)} seconds.")
```

```
--- [0:00] Initializing Analysis ---
--- [0s] Downloading STRING v12 (Full Human) ---
--- [4s] Building High-Confidence Graph ---
✅ FULL GRAPH LOADED: 15882 Proteins, 236712 Interactions

--- [30s] Branch 1: Topological (Fast Approx) ---
   Calculating k-Approximate Betweenness (k=500)...
   Training Node2Vec Embeddings (Parallelized)...

--- [854s] Branch 2: Vulnerability Metrics ---

--- [855s] Branches 3 & 4: Evidence & Localization ---

--- [857s] Fusion & Ground Truth Generation ---

--- [857s] Training ML Models (Stratified) ---

--- [860s] Generating Plots ---
```







Start coding or generate with AI.