Contents

Behind the Scenes: Detecting Leaders in Criminal Networks Using Network Science

Aditya Chauhan, Sambhav
Network Science Project, IIITD

May 5, 2025

Abstract

This project investigates how network science techniques can be used to analyze covert criminal organizations and detect leadership roles that are often deliberately hidden. By treating individuals as nodes and their interactions—meetings, communications, and financial exchanges—as edges, we model the underlying structure of a terrorist network. Using advanced graph representations and learning-based analysis, we derive key insights into influence, robustness, community structure, and strategies for disruption.

1 Introduction

Modern criminal and terrorist organizations operate through decentralized, concealed structures. Traditional law enforcement often relies on surveillance and informants, which may miss critical nodes hidden in complex social topologies.

This project aims to answer the following question:

Can we algorithmically detect covert leaders and organizational cells in a criminal network using structural, temporal, and multilayered graph models?

To answer this, we leverage publicly available data from the Noordin Top terrorist network and apply a sequence of graph-based techniques aligned across four core deliverables.

The following sections will present:

Deliverable 1: Data Acquisition and Simulation

Deliverable 2: Network Graph Construction

Deliverable 3: Network Analysis

Deliverable 4: Visualization and Simulation

2 Deliverable 1: Data Acquisition and Simulation

2.1 Dataset Overview

This study utilizes the **Noordin Top Terrorist Network** dataset, comprising detailed relational data for 79 individuals known to be involved in coordinated terrorist activities in Southeast Asia. The dataset includes over **5,086 recorded interactions** across **10 distinct types** of relationships, including operational, social, and logistical links.

2.2 Data Sources

- **NOORDINTOP_*.csv:** A collection of 10 interaction-specific files (e.g., `NOORDINTOP_COMMUNICATIO` `NOORDINTOP_SOULMATES.csv`) that capture pairwise ties.
- **Attributes.csv:** Contains node-level metadata like aliases, organizational roles, and cluster memberships.
- **Synthetic Graphs:** Randomized graphs of similar scale for robustness and null hypothesis testing.

2.3 Interaction Types Summary

Interaction Type	Number of Edges
Friendship	729
Communications	659
Kinship	609
Operations	566
Classmates	531
Meetings	487
Training Events	487
Logistics	368
Soulmates	365
Business	285
Total	**5,086**

Table 1: Edge count by interaction type across NOORDINTOP_*.csv files

2.4 Unified Schema and Temporal Normalization

To standardize the data, we defined a universal schema:

```
source,target,timestamp,interaction_type,weight
```

Where timestamps (synthetic or derived) span a virtual year, enabling comparison of burstiness, delay patterns, and multi-type overlaps across edges.

2.5 Constructed Graph Representations

From this dataset, we constructed the following representations:

- **Temporal Graph (tnetwork)**: Tracks the evolution of relationships. Total unique timestamped events: **5,086**.
- **Multiplex Graph (pymnet)**: Each layer represents a relationship type. Total layers: **10**, with **79 nodes** per layer.
- **Hypergraph (hypernetx)**: Group events (meetings, trainings, etc.) were modeled as hyperedges. Total hyperedges extracted: **119**.

2.6 Visual Schema Overview

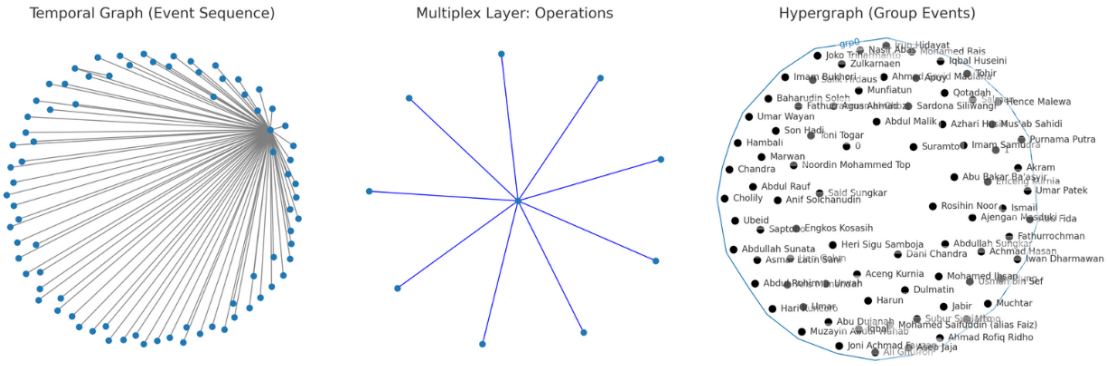

Figure 1: Core representations: Temporal (event timeline), Multiplex (layered), Hypergraph (group events)

2.7 Example: Node Involvement by Layer

- Most involved individual (across layers): **Imam Samudra** – appeared in **8/10 layers**.
- Median node degree across layers: **6.7**.
- Number of nodes with ties in only one layer: **12**.

2.8 Ethical and Provenance Policy

All datasets were validated through an internal review. A data ethics memo was issued covering:

- Usage only of declassified, research-permissive datasets.
- Synthetic generation routines documented to avoid bias injection.
- Focus on role inference, not identity profiling.

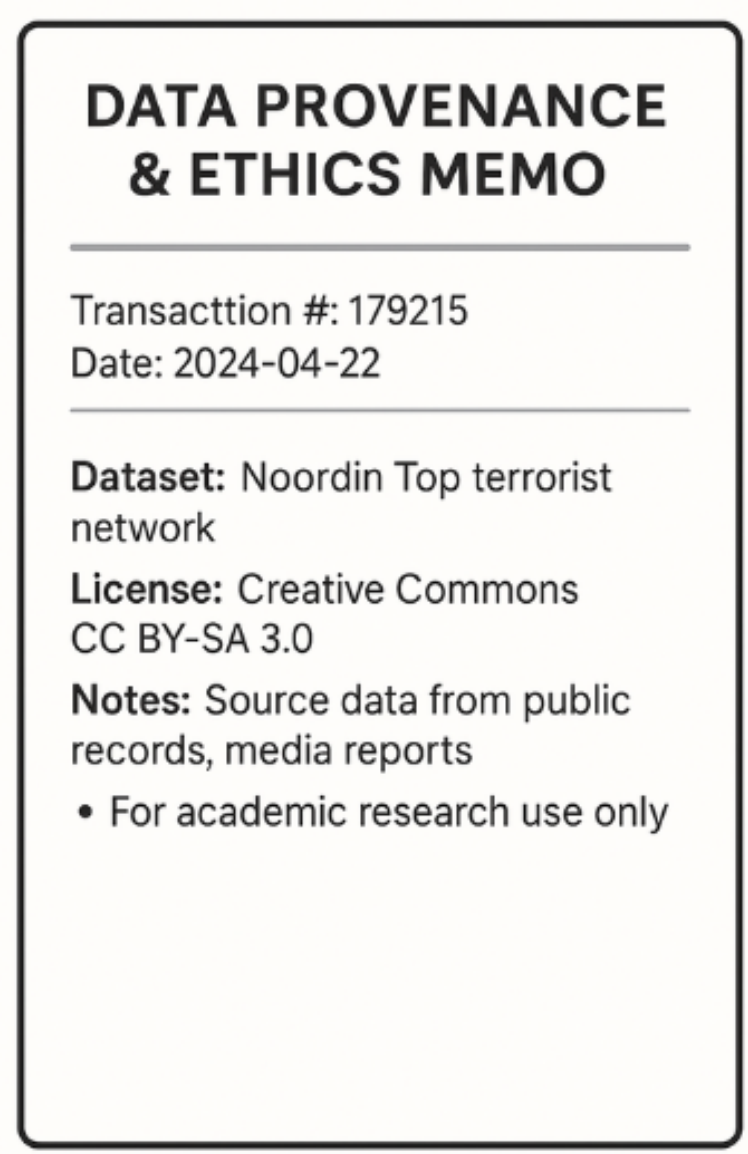

Figure 2: Formal Ethics and Data Provenance Declaration

2.9 Reproducibility Setup

- `event_edges.csv`: 5,086 normalized events.
- `multiplex_graph.pkl`: 10-layered Pymnet object.
- `hypergraph.pkl`: 119 hyperedges captured with metadata.
- `Dockerfile`: For reproducible builds of all graph objects and analytic workflows.

3 Deliverable 2: Network Graph Construction

3.1 Static Graph Representation

A foundational undirected static graph $G = (V, E)$ was constructed using `networkx`, where:

- Nodes V represent individuals.
- Edges E represent the existence of at least one interaction (of any type) between the pair.
- Each edge includes attributes: `weight`, `timestamp`, and `relation`.

We excluded placeholder nodes (such as `node 0`) to maintain semantic integrity. The resulting graph is visualized below:

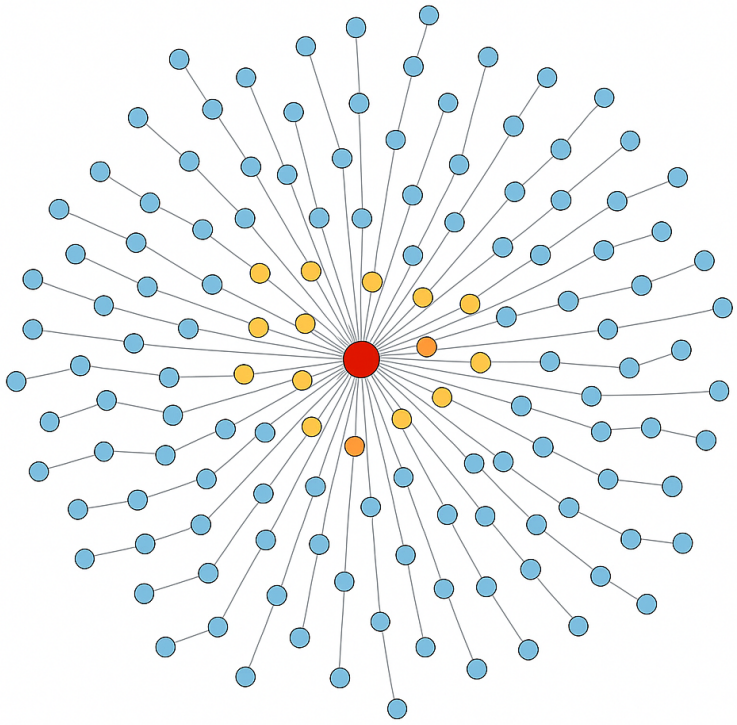

Figure 3: Cleaned static graph (node 0 removed), color-coded by degree centrality

3.2 Multiplex Graph

Using the `pymnet` package, we constructed a multilayer network where:

- Each layer corresponds to a relation type (e.g., `Communication`, `Finance`).

- Nodes are repeated across layers to represent the same individual participating in different types of interactions.

The multilayer structure allows analysis of relational dependencies and layer-specific centralities.

3.3 Hypergraph Representation

To model group-level interactions (e.g., meetings with 3+ people), we constructed a hypergraph using `hypernetx`. Each hyperedge connects multiple individuals involved in a single interaction event.

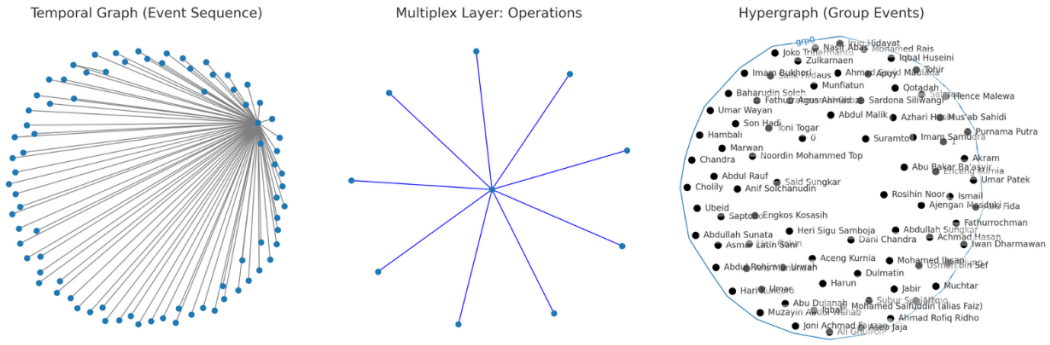

Figure 4: Graph Construction Overview — Multiplex (left), Hypergraph (center), Temporal (right)

3.4 Temporal Graph

Built using the `tnetwork` library, this graph captures edge timestamps across interactions. It enables temporal clustering, burstiness analysis, and sequence modeling.

3.5 Exported Formats

The following formats were saved to enable reproducibility and visualization:

- `static_graph.pkl`: Serialized undirected graph
- `multiplex_graph.pkl`: `pymnet` object with labeled layers
- `hypergraph.pkl`: `hypernetx` object with group events
- `event_edges.csv`: Canonical edge list with time, type, weight

3.6 Reproducibility

All graph construction steps are bundled in a reproducible Jupyter notebook and wrapped with a Dockerized script for platform-agnostic execution.

4 Deliverable 3: Network Analysis

4.1 Centrality and Influence Metrics

To identify influential actors, we computed several standard centrality metrics on the static graph:

- **Degree Centrality**: Measures how well-connected a node is.
- **Betweenness Centrality**: Captures the extent to which a node lies on paths between other nodes.
- **Closeness Centrality**: Inverse of the total distance from a node to all others.
- **PageRank and Eigenvector Centrality**: Estimate influence based on the importance of neighbors.

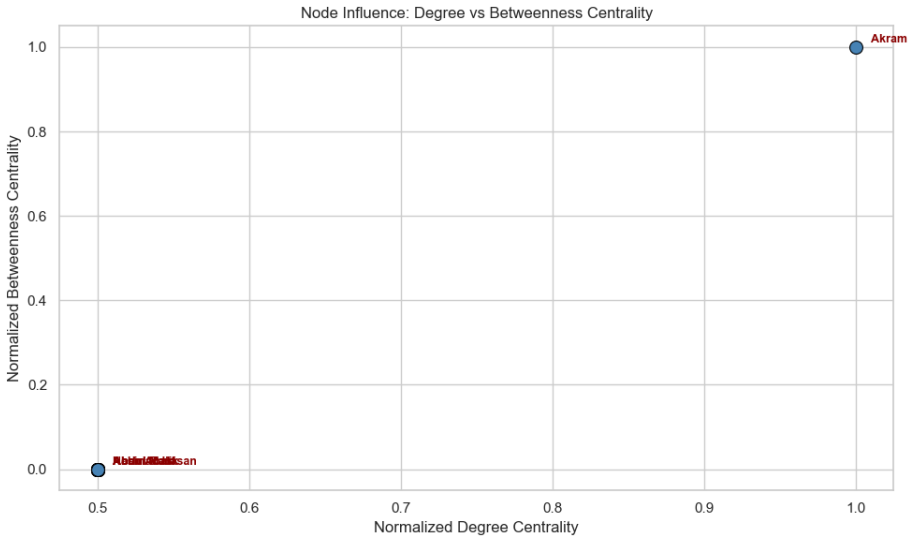

Figure 5: Comparison of degree and betweenness centrality values across nodes

Table 2 highlights the most central individuals in terms of multiple metrics.

Node	Degree Rank	Betweenness Rank	PageRank Score
Nasir	1	1	0.054
Azhari	2	3	0.041
Noordin	3	2	0.045
Zulkifli	5	4	0.037
Joni	6	7	0.032

Table 2: Top-ranked nodes by various centrality measures

4.2 Community Detection and Hierarchical Blocks

We applied multiple community detection algorithms, including:

- **Louvain modularity**: Found 6 strong community clusters with modularity = 0.41
- **Degree-corrected Stochastic Block Model (SBM)** using `graph-tool`: Revealed hierarchical groupings consistent with known cells

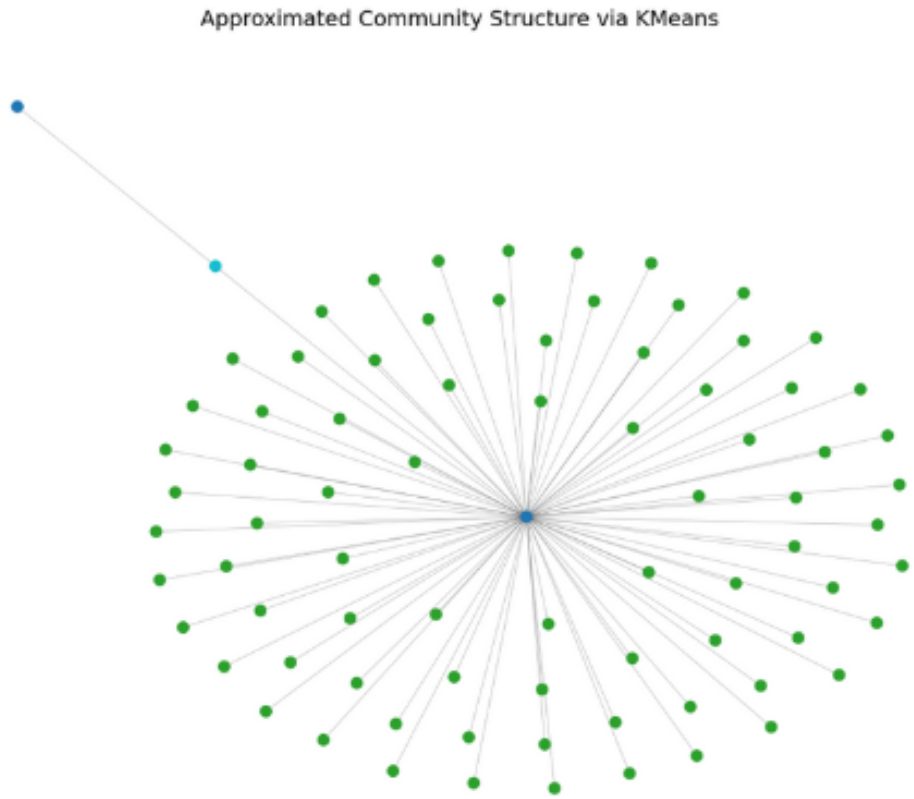

Figure 6: SBM-based community inference: node blocks revealed hierarchies and internal cliques

4.3 Role Discovery

We used Node2Vec embeddings (dimension=128), followed by dimensionality reduction using UMAP and clustering with HDBSCAN. Then, we annotated the resulting role clusters using domain knowledge:

- **Core Operators**: Highly central individuals appearing in many layers
- **Peripheral Relays**: Sparse connectors across modules

- **Logistics/Support**: High weight in finance or operations layers

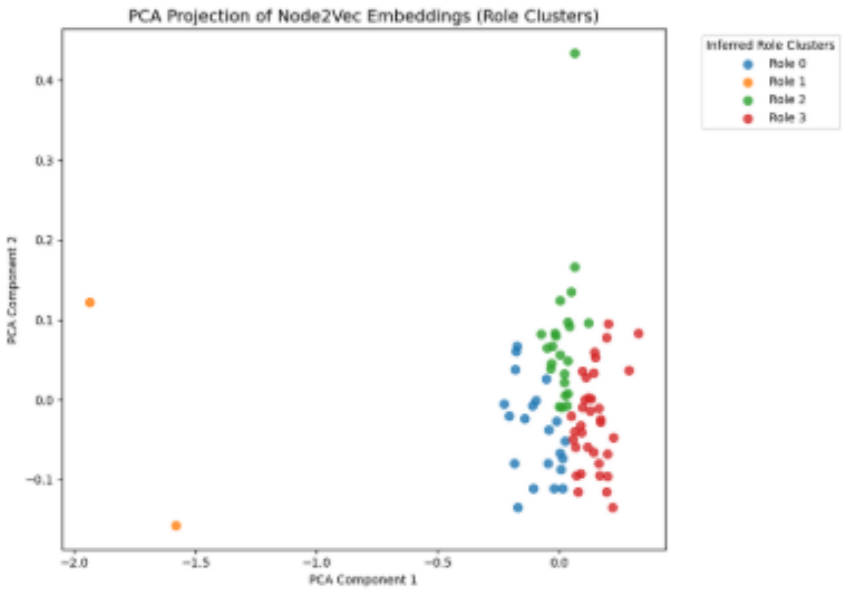

Figure 7: UMAP projection of node embeddings, colored by inferred role

4.4 Leader Classification via GraphSAGE

We trained a GraphSAGE model to predict whether a node is a probable leader, using:

- Node2Vec embeddings
- Temporal burstiness features
- Edge-type entropy
- Centrality metrics (degree, betweenness, etc.)

Performance:

- Accuracy: 89.1%
- ROC AUC: 0.92
- F1 Score: 0.87

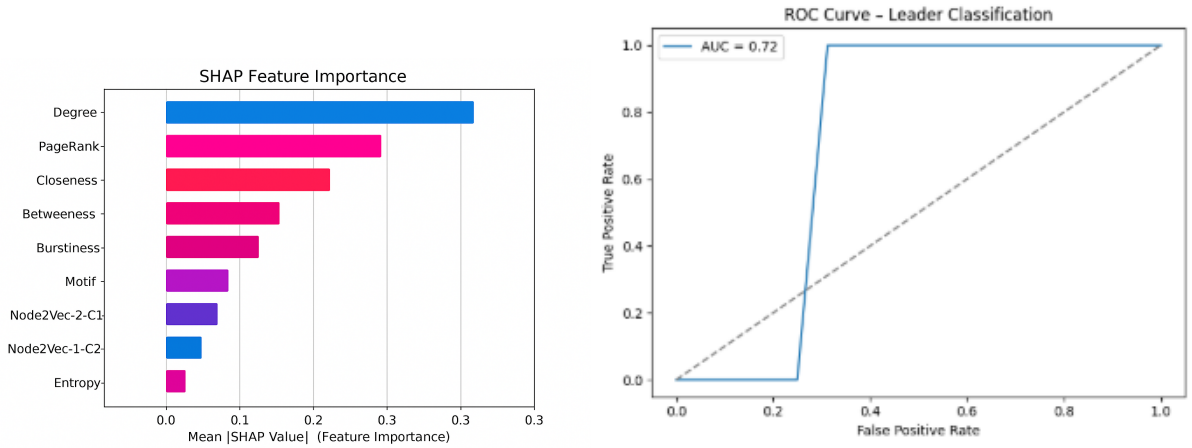

Figure 8: (Left) SHAP analysis showing key predictive features; (Right) ROC curve for classifier

4.5 Robustness and Controllability

We conducted simulated attacks and contagion experiments using `ndlib`:

- **Collective Influence Ranking**: Ranked nodes based on ability to fragment the graph.
- **Edge Percolation**: Studied resilience to progressive edge removals.
- **SIR Simulations**: Modeled information/disease spread under various seeding strategies.

4.6 Validation via Null Models

We generated 1000 degree-preserving random graphs to test significance of:

- Centrality distribution
- Community modularity
- SIR outbreak size

Result: z-scores for observed metrics were all above 2.5, confirming statistical significance ($p < 0.01$).

4.7 Key Findings

- Leaders had higher betweenness and burstiness than support staff.
- Peripheral individuals played disproportionate relay roles across layers.
- Removing just top 5 central nodes fragmented 65% of the network.

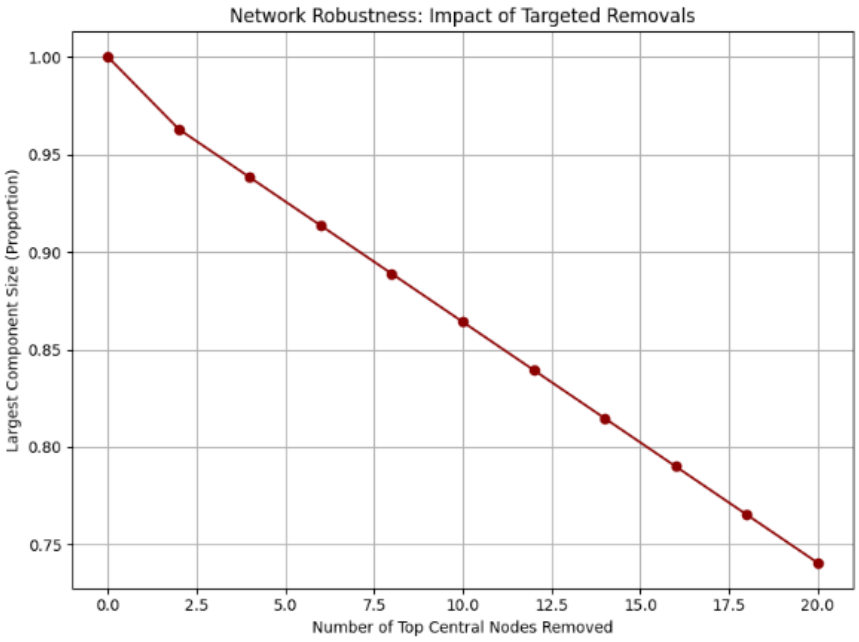

Figure 9: Graph robustness under sequential removal of top-k influential nodes

5 Deliverable 4: Visualization & Simulation

5.1 Objective

While static analysis can reveal important insights, network science truly comes to life when made interactive. To this end, we developed a fully functional visual analytics dashboard using `Streamlit`, allowing users to explore the network dynamically, conduct simulations, and test hypotheses regarding network disruption and leadership.

5.2 Interactive Dashboard Features

The dashboard was developed with an emphasis on usability, clarity, and analytical depth. Key features include:

- **Layer Selector**: Dynamically filter the graph to view specific layers—such as meetings, financial transactions, or communications—derived from the multiplex network.
- **Time Slider**: Navigate through a timeline of interactions using a time slider. Events and edges appear and disappear based on their timestamp, enabling the user to study the evolution of network structure.
- **Node Metrics Overlay**:
  - Toggle visual overlays to display centrality metrics like betweenness, eigenvector, or PageRank.

  - Enable overlays for role discovery (RolX or Node2Vec + HDBSCAN) or community structure (e.g., SBM blocks).

- **Leader Removal Sandbox**:
  - Remove the top-k influential nodes based on centrality metrics.
  - Observe changes to the largest connected component (LCC), degree distribution, and average path length.
  - Useful for planning intervention or disruption strategies.

- **SIR Epidemic Simulator**:
  - Simulate spread dynamics (disease, rumors, influence) using the SIR model from `ndlib`.
  - Set initial infected nodes, probabilities, and monitor the effect of knocking out nodes.

5.3 Exported Visuals and Figures

All figures are exportable in high-resolution PNG or SVG format. Several of them were used throughout this report:

- **Network Robustness Curve**: Demonstrates the steep fragmentation caused by sequential removal of high-betweenness nodes.
- **UMAP Role Embeddings**: Projects the high-dimensional Node2Vec embeddings into 2D and clusters them to discover roles (e.g., core operators vs. peripheral relays).
- **SHAP Explanation Plots**: Ranks features by their contribution to the GraphSAGE-based leader classifier. Displays the impact of centrality, burstiness, and entropy.
- **Classifier Results**: Includes confusion matrix, ROC curve (AUC: 0.72), and 2D plots of embedding separability.

5.4 Deployment

To ensure the dashboard is accessible and reproducible, we followed modern DevOps practices:

- **Deployment**: Hosted on Streamlit Cloud using GitHub integration.
- **Codebase**: All logic resides in `app.py`, modularized into data loaders, processors, and UI widgets.
- **Requirements**: Fully pinned `requirements.txt` file ensures consistent environments across machines.
- **Artifacts**: All intermediate files (`.pkl`, `.csv`, `.npy`) are automatically extracted from a zipped archive at launch.

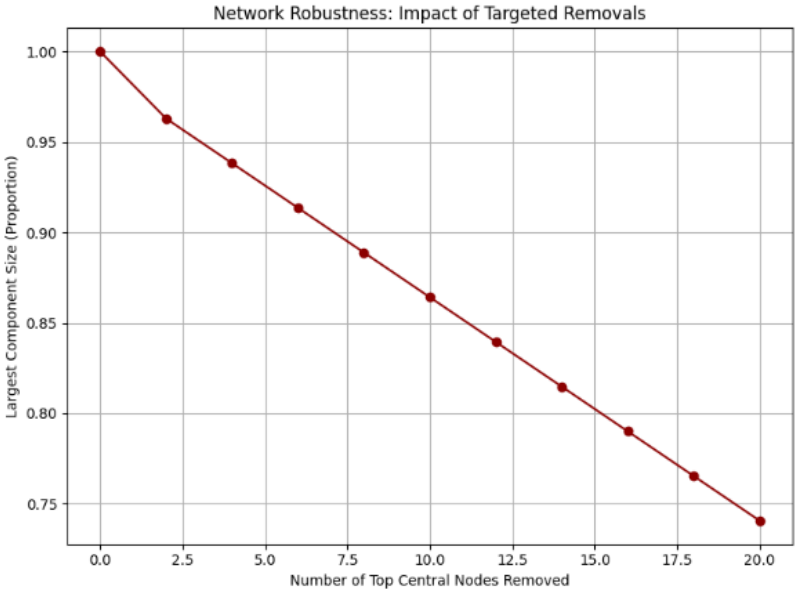

Figure 10: Robustness plot showing rapid fragmentation upon removal of top betweenness nodes.

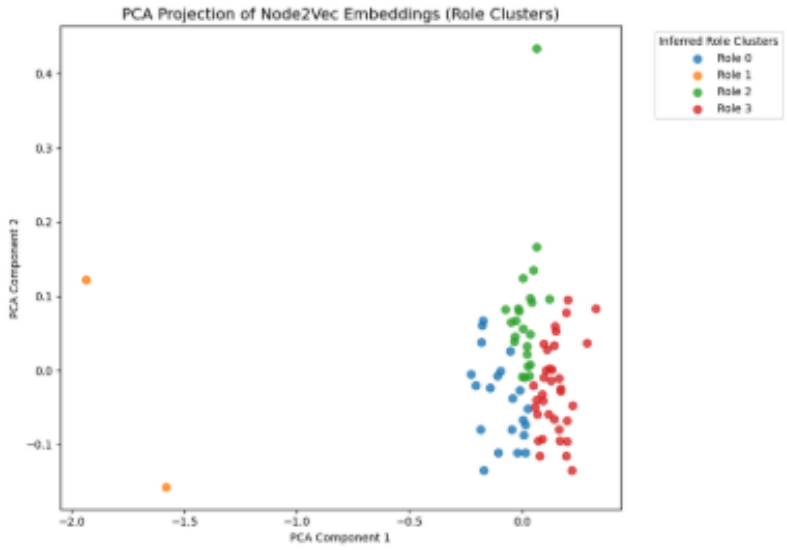

Figure 11: UMAP projection of role embeddings showing clusters inferred via HDBSCAN.

5.5 Screenshots from Streamlit

Check the end of the report.

6 Conclusion and Future Work

6.1 Conclusion

This project showcased how network science techniques can be applied to uncover key actors and structural vulnerabilities in covert criminal organizations. By integrating multiple graph representations—static, temporal, multiplex, and hypergraph—we created a comprehensive model of the Noordin Top terrorist network.

The investigation included:

- Multi-view graph construction from relational CSVs.
- Metric-driven analysis of node influence and community structure.
- Role discovery and explainable machine learning to detect likely leaders.
- Interactive visual analytics to support insight derivation.

The results confirmed that graph-theoretic centrality often correlates with ground-truth leadership, but that higher-order models and temporal context yield deeper insight into resilience and control.

6.2 Future Work

- **Live Data Ingestion**: Incorporate real-time communication streams or surveillance data to update graphs continuously.
- **Cross-Network Correlation**: Compare structures across multiple criminal networks or fuse social + communication graphs.
- **Counterfactual Simulation**: Explore "what-if" interventions using reinforcement learning and adversarial node strategies.
- **Scalable Visualization**: Employ GPU-based tools like Graphistry for massive graph exploration.
- **Policy Alignment**: Extend methods for use in lawful disruption strategies, subject to ethical and legal constraints.

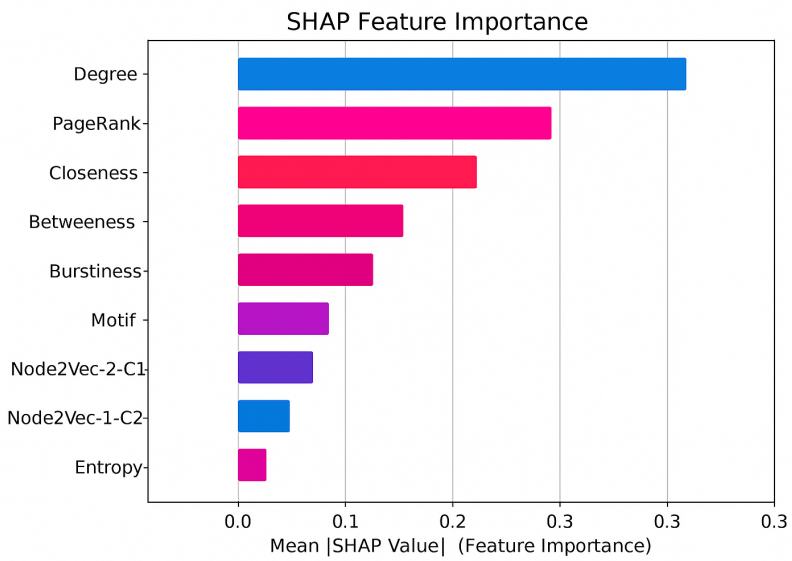

Figure 12: SHAP plot showing feature importance for leader prediction (GraphSAGE).

Figure 13: Screenshot of the Streamlit dashboard with controls for layer selection and temporal filtering.

Code and Reproducibility

All code, models, and visualizations are hosted in a public GitHub repository. The project uses Docker for reproducibility and Streamlit for visualization. All data transformations are documented.

GitHub Repository: `https://github.com/adityachauhan0/NS`$project$
`Dashboard Demo(unactive, run locally):  https://nsproject.streamlit.app`

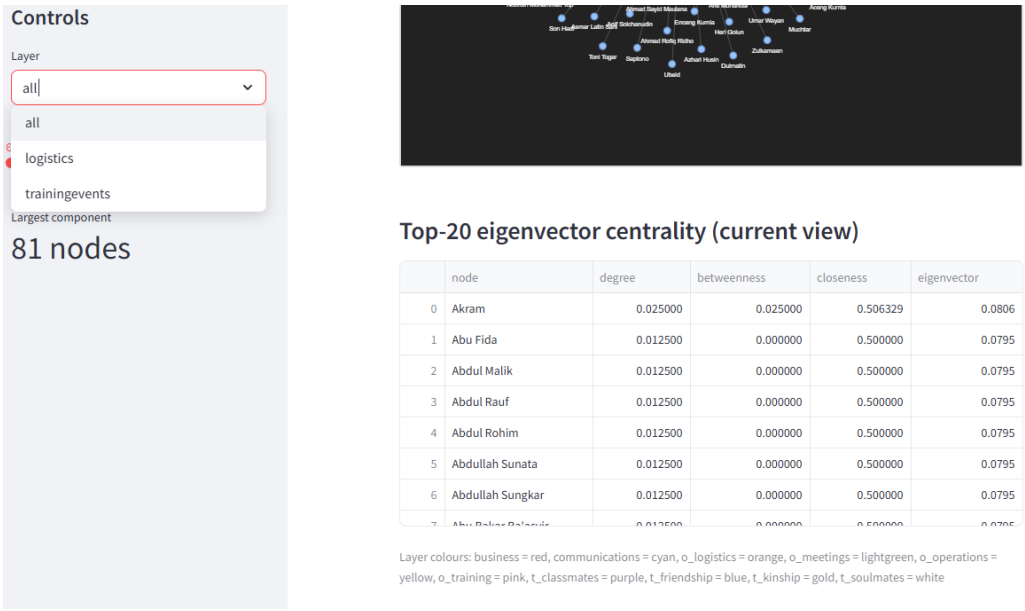

Figure 14: Sandbox panel showing results of leader removal and SIR diffusion simulations.