■ MAJOR PROJECT REVIEW

Complete Preparation Guide

An Explainable GNN-Based Framework for Negative Knowledge Discovery Using Scientific Knowledge Graphs

"We built an explainable, domain-agnostic GNN framework that constructs scientific knowledge graphs from research papers and predicts missing research connections — discovering what scientists should study next but haven't yet. Currently demonstrated on Mental Health domain."

■ Project Identity Card

Field	Detail
Title	An Explainable GNN-Based Framework for Negative Knowledge Discovery Using Scientific Knowledge Graphs
Current Domain	Mental Health (framework is domain-agnostic — see Future Scope)
Core Technique	Graph Convolutional Network + Node2Vec (Link Prediction)
ROC-AUC	99.76%
Papers	653+ (Semantic Scholar + arXiv + PubMed)
Knowledge Graph	659 nodes, 2,428 edges
Output	Interactive 3D visualization with AI transparency

■■ **Important distinction for review:** The framework is domain-agnostic (works for any scientific field). The current demonstration is on Mental Health. The config.yaml already defines Diabetes and Cancer domains — they just need to be activated.

■ What is "Negative Knowledge"?

Connections between research concepts that **should exist** based on the structure of scientific knowledge, but have **never been studied** in any published paper.

- **Known Knowns** → Existing edges in the graph (2,428 edges)
- **Unknown Unknowns** → AI-predicted missing edges = research gaps

■ Complete Pipeline — Step by Step

Step 0 — Database Setup

Scripts: create_db.py, create_entities_table.py, create_relations_table.py

Creates local SQLite database at data/mindgap.db with 3 tables:

Table 1: papers

```
CREATE TABLE papers ( paper_id TEXT PRIMARY KEY, title TEXT, abstract TEXT, year INTEGER,
authors TEXT, venue TEXT, source TEXT );
```

Table 2: entities

```
CREATE TABLE entities ( id INTEGER PRIMARY KEY AUTOINCREMENT, paper_id TEXT, entity TEXT,
type TEXT, source TEXT, category TEXT, UNIQUE(paper_id, entity, type) );
```

Table 3: relations

```
CREATE TABLE relations ( id INTEGER PRIMARY KEY AUTOINCREMENT, paper_id TEXT, head TEXT,
relation TEXT, tail TEXT, UNIQUE(paper_id, head, relation, tail) );
```

Key point: All raw data, extracted entities, and relations are stored in a single SQLite file (mindgap.db). This makes the system portable — can be copied to any machine.

Step 1 — Data Collection from the Internet

Script: fetch_papers.py

We fetch research papers from 3 online APIs using HTTP GET requests via the Python requests library:

Source 1: Semantic Scholar API (~506 papers)

Detail	Value
API URL	https://api.semanticscholar.org/graph/v1/paper/search
Method	HTTP GET with query parameters
Fields fetched	title, abstract, year, authors, venue
Pagination	5 pages × 100 results per page × 10 search terms
Rate limiting	time.sleep(1) between each page

Source 2: PubMed / NIH Entrez API

Detail	Value
Search URL	https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi
Fetch URL	https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi
Method	Two-step: (1) Search → get paper IDs, (2) Fetch → get paper details
Response format	Step 1: JSON, Step 2: XML
Rate limiting	time.sleep(0.5) between each paper fetch

Source 3: arXiv API (~147 papers)

Detail	Value
API URL	http://export.arxiv.org/api/query
Response format	Atom XML
Rate limiting	time.sleep(3) (arXiv is strict)
Categories	neuroscience (q-bio.NC), computers & society (cs.CY), statistics (stat.AP)

10 Search Terms Used

- 1. "depression mental health"
- 2. "anxiety disorder treatment"
- 3. "PTSD therapy"
- 4. "suicidal ideation risk"
- 5. "cognitive behavioral therapy"
- 6. "dialectical behavior therapy"
- 7. "bipolar disorder treatment"
- 8. "mindfulness therapy"
- 9. "trauma mental health"
- 10. "loneliness depression"

After Step 1: The papers table in mindgap.db has 653+ rows — each row is one paper with its full abstract text.

Step 2 — NLP Entity Extraction

Script: extract_entities.py

We run two NLP models from the spaCy/scispaCy library on every paper's abstract:

- **en_core_sci_sm** — Trained on scientific text corpora. Extracts general biomedical terms: disorders, therapies, biological processes. Stored as: "biomedical_term"
- **en_ner_bc5cdr_md** — Trained on BioCreative V CDR corpus (14,000+ annotated abstracts). Specializes in disease names and drug/chemical names. Stored as: "disease_or_drug"

```
for paper_id, abstract in all_papers: doc = sci_nlp(abstract) for entity in doc.ents:
save("biomedical_term", entity.text.lower()) doc = disease_nlp(abstract) for entity in
doc.ents: save("disease_or_drug", entity.text.lower())
```

Step 3 — Entity Classification

Script: classify_entities.py

Each entity gets a category based on keyword matching:

Category	Keywords (examples)
disorder	depression, anxiety, ptsd, bipolar, panic, ocd, schizophrenia
therapy	therapy, cbt, dbt, treatment, counseling, ssri, mindfulness
risk_factor	trauma, abuse, stress, insomnia, sleep, loneliness, poverty
outcome	suicide, relapse, recovery, self harm, quality of life
population	adolescent, child, teen, student, women, veteran, elderly

Step 4 — Relation Extraction (Co-occurrence)

Script: extract_relations.py

We find which entities appear together in the same sentence within a paper's abstract. If entities A and B both appear in the same sentence → they are "related."

```
for sent in doc.sents: sentence_text = sent.text.lower() present = [entity for entity in
paper_entities if entity in sentence_text] for i in range(len(present)): for j in range(i+1,
len(present)): save_relation(paper_id, present[i], "related_to", present[j])
```

Step 5 — Knowledge Graph Construction

Script: build_graph.py

Build a NetworkX graph from the database:

- **Nodes** = unique entities (with category IS NOT NULL)
- **Edges** = relations from the relations table (excluding self-loops)

```
G = nx.Graph() for entity, category in nodes: G.add_node(entity, category=category) for head,
tail in edges: if head != tail: G.add_edge(head, tail, relation="related_to")
```

Metric	Value
Nodes	659 biomedical concepts
Edges	2,428 co-occurrence connections
Type	Undirected, unweighted
Output	data/mental_health_graph.pkl

Step 6 — Node2Vec Embedding Training

Script: train_node2vec.py

Since the graph has no numerical features, we create vector representations using Node2Vec (random walk-based graph embedding).

How Node2Vec works:

- **Random walks** — Start at each node, take random walks through the graph
- **Word2Vec** — Treat walks as "sentences" and nodes as "words" → learn embeddings

Parameter	Value	Meaning
dimensions	64	Each node becomes a 64-dimensional vector
walk_length	20	Each random walk is 20 steps long
num_walks	200	200 walks started from each node
workers	2	Parallel processing threads
window	10	Word2Vec context window size
min_count	1	Include all nodes, even rare ones

Output: data/node_embeddings.wv — 659 nodes × 64 dimensions

Why Node2Vec? The graph has no inherent features — just node names and edges. Node2Vec learns features from graph structure. Nodes with similar neighborhoods get similar embeddings.

Step 7 — Convert to PyTorch Geometric Format

Script: build_pyg_graph.py

- Load NetworkX graph from mental_health_graph.pkl
- Load Node2Vec embeddings from node_embeddings.wv
- Build node feature matrix X: [659 × 64] tensor
- Build edge index: [2 × 4856] tensor (2,428 × 2 for both directions)

Output: data/pyg_graph.pt — ready for GNN training

Step 8 — GNN Model Training

Script: train_gnn.py

Architecture:

```
Input: X [659 × 64] | GCNConv(64, 64) <- First graph convolution layer | ReLU <- Non-linear activation | GCNConv(64, 32) <- Second graph convolution layer | Output: Z [659 x 32] <- 32D embedding per node
```

Link Prediction Decoder:

```
def decode(z, edge_index): return (z[edge_index[0]] * z[edge_index[1]]).sum(dim=1) # Dot product of node pair embeddings -> scalar score
```

High dot product → model predicts edge exists. Low dot product → model predicts no edge.

Training Loop (200 epochs):

- **Forward pass** — get 32D embeddings for all 659 nodes
- **Positive samples** — score all 2,428 real edges → should be high
- **Negative sampling** — randomly sample 2,428 NON-edges → should be low
- **Loss** — Binary Cross-Entropy with Logits
- **Backprop** — Adam optimizer (lr=0.01)

Evaluation: Score 1,000 real + 1,000 non-edges → **ROC-AUC = 99.76%**

Output: data/gnn_model.pt

Step 9 — Research Gap Prediction + Visualization

Script: visualize_credible_ai.py

- Load trained GCN model and get 32D embeddings for all nodes
- Sample 15,000 random node pairs that are NOT connected
- Score each pair: $P(\text{edge}) = \text{sigmoid}(z_u \cdot z_v)$
- Sort by confidence → pick Top 20 as research gap predictions
- Build 3D interactive visualization using Plotly with transparency panel

Output: data/graph_credible_ai.html — opens in any browser

■ Complete File Map

```
data/ ■■■ mindgap.db <- SQLite database (papers + entities + relations) ■■■
mental_health_graph.pkl <- NetworkX graph (659 nodes, 2428 edges) ■■■ node_embeddings.wv <-
Node2Vec embeddings (659 x 64D) ■■■ pyg_graph.pt <- PyTorch Geometric graph (for GNN) ■■■
gnn_model.pt <- Trained GCN weights ■■■ graph_credible_ai.html <- Final interactive 3D
visualization
```

■ Numbers to Memorize

Metric	Value
ROC-AUC	99.76%
Papers	653+
Graph Nodes	659
Graph Edges	2,428
Embedding Dimension	64D (Node2Vec) → 32D (GCN output)
Training Epochs	200
Learning Rate	0.01
Random Walks	200 walks × 20 steps
Candidate Pairs Evaluated	15,000
Top Predictions Shown	20
Domains Configured	3 (Mental Health active, Diabetes + Cancer ready)

■ Likely Reviewer Questions & Answers

Question	Answer
How do you fetch data?	HTTP GET requests to 3 APIs: Semantic Scholar (JSON), PubMed (XML), arXiv (Atom XML). Res
Where is the data stored?	All in data/mindgap.db — SQLite DB with tables for papers, entities, relations. Models stored as .pk
How do you handle duplicates?	INSERT OR REPLACE for papers (unique by paper_id), UNIQUE constraints on entities and relati
How do you handle rate limiting?	time.sleep(1) for Semantic Scholar, time.sleep(0.5) for PubMed, time.sleep(3) for arXiv.
Why SQLite and not MySQL?	SQLite is embedded (no server needed), portable (single file), sufficient for our scale. Production –
What if an entity has no embedding?	Zero vector (64D) as fallback in build_pyg_graph.py. GCN learns purely from graph structure for th
How is this different from co-occurrence?	Co-occurrence misses multi-hop patterns. GCN aggregates info from 2-hop neighbors through 2 la
Why not use BERT/LLMs?	spaCy handles entity extraction (NLP). GCN operates on graph structure, not text. They solve diffe
What is the novelty?	Applying GNN link prediction as a research gap discovery tool for mental health, with full AI transpa
What are limitations?	No train/val/test split, no negation handling, English only, batch-only, undirected graph, single dom
Can this work for other domains?	Yes — domain-agnostic. config.yaml already defines Diabetes and Cancer. Future: user selects ar

■ Current Scope & Future: Multi-Domain Support

What is already built (in config.yaml)

Domain	Search Terms (examples)	Entity Categories
Mental Health ■ (active)	depression, anxiety, PTSD, CBT, DBT	disorder, therapy, risk_factor, outcome, population
Diabetes ■ (configured)	insulin resistance, diabetic neuropathy, blood glucose	condition, treatment, complication, outcome, population
Cancer ■ (configured)	breast cancer, immunotherapy, tumor biomarkers	cancer_type, treatment, risk_factor, outcome, population

Future Vision: Dynamic Domain Selection

In the next version, a researcher will be able to select or type a domain (e.g., "Cardiology", "Neurology") and the system will automatically:

- Generate relevant search terms using LLM or predefined config
- Fetch papers from Semantic Scholar, PubMed, and arXiv for that domain
- Run NLP entity extraction with domain-appropriate models
- Classify entities into domain-relevant categories
- Build a new knowledge graph specific to that domain
- Train a fresh GCN model and predict/visualize research gaps

How to command-switch domains (already designed):

```
python run_pipeline.py --domain cancer python run_pipeline.py --domain diabetes
```

Key point: The framework is domain-agnostic by design. Mental Health is our proof of concept. The same architecture generalizes to any scientific field — only the config changes.

■■ Limitations (Shows Maturity)

- Single domain active — only Mental Health fully tested; Diabetes and Cancer configs exist but not yet executed
- No formal train/val/test split — evaluating on all edges currently
- Negation blindness — "CBT is ineffective for PTSD" still creates a CBT-PTSD edge
- English-only — misses research published in other languages
- Batch processing — not real-time; new papers require re-running the pipeline
- Undirected graph — loses causal direction ("A causes B" = "B causes A")

■ Future Work

- Multi-domain UI → Web interface where researchers select/type any domain → full pipeline runs automatically using config.yaml
- Directed causal graph → Use dependency parsing to extract causal relations, convert to directed GCN
- Real-time updates → WebSocket pipeline: new paper published → auto-extraction → graph re-inference
- Cross-domain fusion → Single unified graph across Mental Health + Diabetes + Cancer to find inter-disciplinary research gaps

■ Pre-Review Checklist

- ■ Run visualize_credible_ai.py to regenerate fresh predictions
- ■ Open data/graph_credible_ai.html in browser
- ■ Know the 9 pipeline steps + table schemas
- ■ Memorize: 99.76% ROC-AUC, 653 papers, 659 nodes, 2,428 edges
- ■ Practice rotating the 3D graph while explaining
- ■ Be ready to explain dot-product decoder and negative sampling
- ■ Know all 3 API sources and their response formats
- ■ Be ready to explain multi-domain architecture and config.yaml

■ Confidence tip: You built a full end-to-end, domain-agnostic AI framework — from raw API calls through NLP, graph construction, GNN training, to an interactive 3D visualization. Very few students build something this complete. Own it.