CausGT-HS: An Energy-Based Causal MoE-GNN for Causal Reasoning

Shashank Tippanavar - IMT2022014
Siddharth Palod - IMT2022002
Kushal Jenamani - IMT2022057
Shanmukh Praneeth - IMT2022542

International Institute of Information Technology Bangalore

Contents

1 Introduction

1.1 Github Link

Github Repo Link: https://github.com/MightyShashank/CausGT-HS$_E BM - project$

1.2 Project Overview

We solve for the following 3 novelties throughout this project:

- **Causal meta-path discovery**

2 Discovering Latent Causal Mechanisms from Static, Observational Text

2.1 The Core Problem:

The central goal of advanced information retrieval is to move beyond simple, correlational (which has to do only with geometry) extraction to discover deep, explanatory, and **causal** mechanisms hidden within unstructured text (e.g., PDFs). Current systems, like GraphRAG are all "correlational engines", meaning they excel at identifying and summarizing explicitly stated relationships (i.e., just the retrieved facts), effectively answering **"what"** and not **"why"**. Our objective is to build a system that answers **"Why is this related?"** by discovering the underlying causal meta-paths.

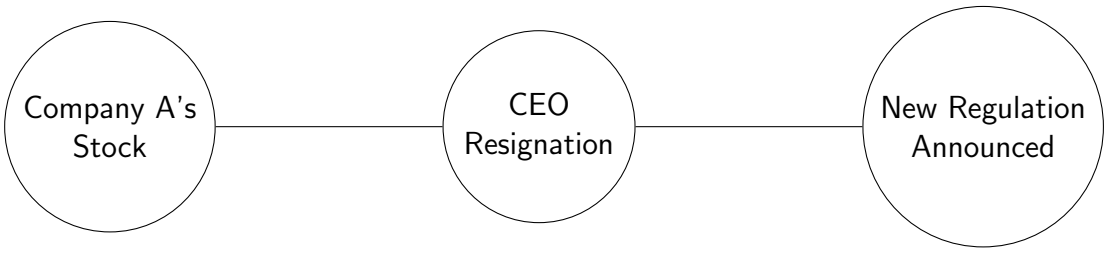
- latent = hidden
- meta-paths = path of classes and not instances
- causal = seeing the cause and not just structural similarity.

In simple terms, the problem is the difference between **"what"** and **"why"**.

Today we are excellent at building KGs that tell us what things are related. But we are terrible at building KGs that tell us why they are related, or which "what" caused the other. This is the **"correlation vs causation"** problem.

Lets see this with an example: `A financial news PDF` Imagine a news paper article of some company..

- **Current KGs (That "What"/Correlation):** Your system can easily extract a graph like this:

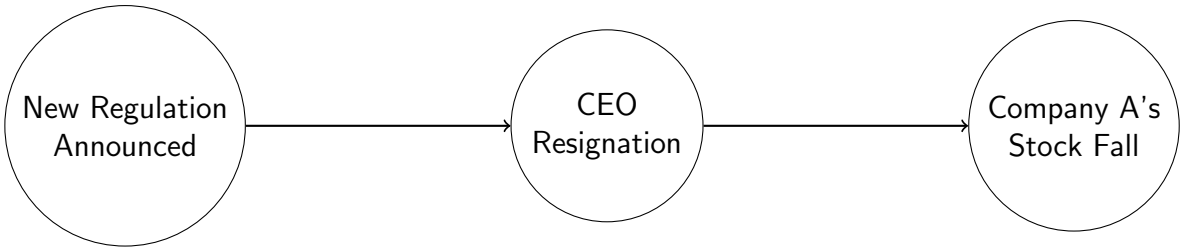


This graph is a "dumb" map. It is flat and undirected. It only tells you that these topics appeared together, but it offers no explanation.

Did the stock fall because the CEO resigned? Or did the new regulation cause the CEO to resign, which then caused the stock to fall? A standard Knowledge Graph (KG) cannot tell the difference.

- **Our Goal (The "Why"/Causation):** We want to automatically discover the real story - the **causal meta-path**

  A possible causal chain can be represented as:



  Unlike an undirected knowledge graph, this causal structure indicates how one event may influence another.

The Core Challenge

How can a computer discover this "why" graph ($A \rightarrow B$) when all it has to read is a single, static PDF?? This is called the **"static observational data trap"**. All the computer sees is that A and B are "observed" together, not which one "acted" first.

Static observational data trap

Refers to the methodological pitfalls and limitations that arise when researchers use data collected at a single point in time to make conclusions that require information about change, cause, or dynamic processes.

This presents a formidable challenge, which existing methods are unequipped to solve:

1. **The Observational Data Trap:** The gold standard methods for causal discovery (e.g., DAG-GNN, NOTEARS, PC-Algorithm) are designed for tabular, interventional, or time-series data. They require seeing how variables change in response to one another. Document KGs are **static and purely observational**. We only have one "snapshot" of the graph, derived from static text. Causality must be inferred from static linguistic cues and world knowledge, not observed changes.

2. **The "Causality-Blind" GNN:** Standard GNNs (GCN, GAT) are fundamentally "correlation" based. Their message-passing mechanisms (even with attention) are typically symettric (same in both directions of edges). They cannot, by design, differentiate between $A \rightarrow B$ (causation) and $A - B$ (correlation).

3. **The Noise of Reality:** Real-world knowledge isnt binary. The initial graph *G(V,E,R)* extracted from a PDF is noisy. A superior model must operate on a **weighted, multi-relational graph G(V,E,R,W)**, where W represents initial "correlational" confidences or like proximity.

4. **The $O(N^2)$ Scalability Bottleneck:** Real-world documents (e.g., a 100-page technical manual or legal filing) can contain $N > 100,000$ unique entities. Any algorithm that requires building or computing $N \times N$ matrices (e.g., a full attention mechanism or a dense adjacency matrix) is **computationally infeasible**. A good solution must scale linearly or near-linearly ($O(N)$ or $O(N \log N)$).

We propose a novel architecture, **CausGT-W**, that solves all the above challenges by creating a new end-to-end Graph trannsformer that learns causality by being "taught" by a LLM's **counterfactual reasoning** to generate a **Causal Prior dataset**. Its a self-supervised framework for discovering directed, causal meta-paths from static, observational text.

We do this by not asking the LLM "is this causal?" but by forcing it to perform counterfactual reasoning ("What if...?") on text snippets. For example, "If Algorithm A were NOT used, what would be the impact on Accuracy?" The answers to these "what if" questions, which the LLM can infer from its vast world knowledge, become our only supervisory signal for causality.

2.2 Abstract: The CausGT-HS solution:

We introduce **CausGT-HS (An Energy-Based Causal MoE-GNN for Causal Reasoning)**, a novel, end-to-end, dual-stream graph transformer architecture. Its purpose is to transform a noisy, weighted, multi-relational, and correlational graph (extracted from a PDF) into a single clean, directed and weighted causal meta-path graph.

Our architecture is founded on 4 key novelties:

1. **End-to-End Weighted Meta-Path Learning:** We introduce a **Weighted Graph Transformer Network (GTN-W)** module inside the main architecture. This module learns to compose the input weighted correlational paths into **latent, multi-hop, weighted meta-paths** (W_{CorrMeta} and $W_{\text{CausalMeta}}$). This module is trained end-to-end, guided by the causal loss.

2. **Hierarchical-Sparse Architecture (Solves the scalability issue):** This architecture ain't monoithic. It first runs a fast graph-coarsening step to cluster N nodes into C "supernodes" ($C << N$). A small, dense CausGT model finds the $O(C^2)$ "highway" causal paths. Then, a highly-efficient sparse CausGT model runs in parallel within each cluster to find the "local" paths, reducing the $O(N^2)$ attention problem to a linear-time $O(N.k)$ operation.

3. **Dual-Stream Causal-Transformer:** The core of CausGT-W is a Graphormer-based encoder with a **dual-stream attention mechanism**. One stream is a **correlational head** (using W_{CorrMeta}) and the other is a **causal head** (using $W_{\text{CausalMeta}}$). The causal head uses **asymmetric attention** ($W^Q \neq W^K$) and is explicitly trained to replicate the LLM's counterfactual reasoning.

   Dual-Stream Causal-Transformer

   A Dual-Stream Transformer is a model architecture where two different data streams (e.g., visual and textual, or node and relation, or source and target) are processed separately but interact through a specialized attention mechanism.
   So instead of a single Transformer encoding one sequence (as in vanilla BERT or ViT), you have two parallel Transformers, each maintaining its own set of representations.

4. **Dynamic Gated Fusion:** A learnable gating mechanism (λ) for each node dynamically learns how much to trust the correlational stream vs the causal stream, creating robust, context-aware node representations.

5. **Mediator-Controlled Counterfactural Distillation**: We devise a novel self-supervised paradigm. An LLM acts as a "Generalist Teacher," performing Mediator-Controlled Counterfactual (MCC) reasoning on candidate edges identified by our pipeline. Crucially, it employs Latent Mediator Elicitation (LME) to hypothesize unobserved mediating concepts (C in $A \rightarrow C \rightarrow B$) based on its world knowledge. The LLM outputs a sparse Causal Prior (C_{prior}) representing direct causal links only. This prior becomes the sole supervisory signal for causality, distilling complex, interventional reasoing into our GNN.

This architecture fundamentally distills the slow, symbolic, high-level causal reasoning of an LLM into a fast, numerically-stable, GNN architecture, solving the "static" problem.

3 The CausGT-HS Architecture: A Detailed Formulation:

3.1 Basic info on the model:

3.1.1 Model inputs:

- **Node Features:**

$$H^{(0)} = X \in \mathbb{R}^{X \times d_{in}}$$

, where N = number of nodes (entities) and d_{in} is the initial embedding dimension.

- **Weighted Relational Graphs:** A set of K weighted adjacency matrices

$$A_W = \{W_1, W_2, \ldots, W_K\} \quad \text{where} \quad W_r \in [0,1]^{N \times N}$$

Each one of these matrices above matches to one type of single, correlational relationship. Each of the W_r above represents relationships between any of the N nodes and any of the other N nodes for a specific relation r. $|E_r|$ is the number of non-zero entries in W_r. Let $|E| = \sum |E_r|$.

- **The Causal Prior** C_{prior}: Our GNN is supposed to learn to immitate this. This stores a causal score for every possible directed pair from node i to node j.

3.1.2 Model outputs:

- **The primary output:** The causal meta-path graph $W_{\text{CausalMeta}} \in [0,1]^{N \times N}$ that represents the underlying causal mechanisms. They represent final learned causal strength for any directed path from node i to node j.

- **The secondary output:** Our GNN's other job was to update node features. Hence an another output is $H^{(final)}$ which are our causal aware node embeddings. This is the final list of node embeddings after they have passed through all the causal transformer layers. (So now these embeddings are causal aware).

3.2 Learning our correlational matrices A_W:

This is a one-time, per-document pre-processing that uses the LLM "Teacher".

1. **Initial Extraction and Indexing:** Here we extract the following:

   - Nodes **N**
   - Sparse graphs $\mathbf{A_W}$
   - Inverted Index $\mathbf{T_{map}}$: For every node this lists all the sentence numbers where that node appears.

   Here we extract the above from say a large pdf ($100+$ pages, potentially $N > 100k\ entities$) efficiently. Naive methods face many bottlenecks like:

   - **LLM context limits:** Feeding entire large documents to LLMs for extraction exceeds context windows.

- $\mathbf{O(N^2)}$ **Pair Explosion** : Checking all possible entity pairs for relations is computationally impossible.
- **Redundant LLM Calls:** Processing sentence-by-sentence or pair-by-pair leads to excessive, slow, and costly LLM queries.
- **Ignoring Structure:** Simple text chunking misses relations spanning sections.

Our entire Setup:

- **Input Parameter:** User provides K (A hardcoded desired number of relation matrices, the more the better).
- **Input Text (D):** The plain text extracted via OCR from the document.
  Example Text (Document D):
  P1: (S1) The XGBoost model (XGB) utilizes gradient boosting (GB) principles. (S2) XGBoost provides superior performance (PERF) compared to traditional gradient boosting.
  P2: (S3) We evaluated XGBoost on the ImageNet dataset (IMG). (S4) The performance achieved was 92% accuracy (ACC). (S5) Accuracy is a key metric.
- **Sentence Embedder (f_{embed}):** A pre-trained sentence embedding model (e.g., all-MiniLM-L6-v2 from sentence-transformers).
- **Initialise Outputs:**
  - $N = \phi$ (Our Node set: $\{(\texttt{node\_id}, \texttt{entity\_name})\}$)
  - $T_{\text{map}} = \{\}$ (Maps `node_id` to $(\texttt{paragraph\_id}, \texttt{sentence\_id})$).
  - ExtractedTriplets=[],
  - $A_W = \{W_1, \ldots, W_K\}$ (Its a set of K empty sparse matrix builders in COO format)

COO format

Above W_k is not yet a dense matrix, but rather a sparse representation being constructed in the COO format (Coordinate list format).
In the COO format for sparse matrices, instead of storing all entires of a matrix (most of which might be 0s), we only store the coordinates and values of non-zero elements.

$$\begin{aligned} \text{rows}_k &= [i_1, i_2, \ldots] \\ \text{cols}_k &= [j_1, j_2, \ldots] \\ \text{data}_k &= [v_1, v_2, \ldots] \end{aligned}$$

Each triple (i_t, j_t, v_t) represents a nn-zero entry:

$$W_k[i_t, j_t] = v_t$$

Formally:

$$W_k \equiv \{(\text{rows}_k, \text{cols}_k, \text{data}_k)\}$$

where

* $\text{rows}_k[i]$ = source node index of edge i,
* $\text{cols}_k[i]$ = destination node index of edge i,
* $\text{data}_k[i]$ = (optional) weight of edge i.

So, each W_k starts as an empty list of triplets, ready to be filled with edge connections or weighted relationships. Each relation type has its own adjacency structure W_k.
So:

$$A_W = \{W_1, \ldots, W_K\}$$

represents K different adjacency builder, one per relation type.
During computation:

* The model will learn or generate which node pairs are connected (and with what strength).
* Initially, each W_k starts empty (no edges added yet).
* Then, as the model discovers or constructs edges, entries get added to row_k, $cols_k$, $data_k$.

$$A_W = \{W_1, \ldots, W_K\}, \quad W_k = \text{SparseMatrixBuilder}(\text{rows}_k, \text{cols}_k, \text{data}_k)$$

where each W_k will ultimately form an adjacency matrix encoding one *type of relation* or *meta-path* in the graph.
A_W is a collection of K relational adjacency builder that will later become the weighted connectivity patterns (edge) used by the model to reason over the graph.
These W_k later on when we materialise or instantiate the adjacency representation (i.e., when the model needs to perform matrix multiplication or attention propagation using these edges) become $N \times N$ matrix.

Text Parsing and Node extraction (N, T_{map}):

- **Parse Text:** Segment D into paragraphs (P_p) and sentences (s_m). Assign unique IDs. Structure

$$\text{Structure} = \left\{ \begin{array}{l} \text{Doc} : \{ \\ \quad \text{P1} : [\text{S1},\ \text{S2}], \\ \quad \text{P2} : [\text{S3},\ \text{S4},\ \text{S5}] \\ \} \end{array} \right\}$$

  Above
  - S1 = "The XGBoost model (XGB) utilizes gradient boosting (GB) principles."
  - S2 = "XGBoost provides superior performance (PERF) compared to traditional gradient boosting."
  - S3 = "We evaluated XGBoost on the ImageNet dataset (IMG)."
  - S4 = "The performance achieved was 92% accuracy (ACC)."
  - S5 = "Accuracy is a key metric."
- **Entity Extraction (LLM NER) and Build Node set:** Run NER the full text.
  - **Prompt:** "Extract all significant technical entities. Output as JSON list."
  - **LLM Output:** ["XGBoost", "gradient boosting", "performance", "ImageNet", "accuracy"]
  - **Building Node Set (N):** Assign unique IDs.

$$N = \left\{ \begin{array}{l} (1, \texttt{XGBoost}), \\ (2, \texttt{gradient boosting}), \\ (3, \texttt{performance}), \\ (4, \texttt{ImageNet}), \\ (5, \texttt{accuracy}) \end{array} \right\}$$

  - **Build T_{map}:** We iterate through sentences and nodes and create a mapping for nodes (their corresponding IDs) as below:

$$T_{\text{map}} = \left\{ \begin{array}{ll} 1: & [(P_1, S_1), (P_1, S_2), (P_2, S_3)], \quad // \text{ XGBoost} \\ 2: & [(P_1, S_1), (P_1, S_2)], \quad // \text{ gradient boosting} \\ 3: & [(P_1, S_2), (P_2, S_4)], \quad // \text{ performance} \\ 4: & [(P_2, S_3)], \quad // \text{ ImageNet} \\ 5: & [(P_2, S_4), (P_2, S_5)] \quad // \text{ accuracy} \end{array} \right\}$$

Note

- $\mathbf{P_p}$:
  This simply refers to a specific paragraph in your document, identified by its index or ID p.
  Ex: P_1 is the first paragraph, P_2 is the second paragraph.

- $\mathbf{N_p}$:
  Refers to nodes in paragraph p. This is the set of node IDs (representing entities) that appear anywhere within that specific paragraph P_p.
  Ex: If paragraph P_1 contains sentences mentioning Node 1, Node 2, and Node 3 (but not Node 4 or Node 5), then $N_1 = \{1, 2, 3\}$

- $\mathbf{E_{probe}}$ (Paragraph-Level Candidate Pair Pruning):
  This is the list of all entity pairs that will eventually be checked with the LLM. It is constructed by iterating through all paragraphs. For each paragraph P_p, all pairs of nodes (i, j) that both appear in that paragraph (i.e., $i \in N_p$ and $j \in N_p$) are identified and added to the global set E_{probe}. Using a set automatically handles duplicates when a pair appears together in multiple paragraphs.
  Example:
  From P_1 (containing nodes $\{1, 2, 3\}$), you add pairs $(1, 2)$, $(1, 3)$, $(2, 3)$ to E_{probe}.
  From P_2 (containing nodes $\{1, 3, 4, 5\}$), you add pairs $(1, 3)$, $(1, 4)$, $(1, 5)$, $(3, 4)$, $(3, 5)$, $(4, 5)$ to E_{probe}. (Note that $(1, 3)$ is already present, sets handle duplicates).
  The final E_{probe} for the document (so far) would be:

$$E_{\text{probe}} = \left\{ \begin{array}{lll} (1,2),\ (1,3),\ (2,3), & \text{// From } P_1 \\ (1,4),\ (1,5), & \text{// From } P_2 \text{ (1,3 already exists)} \\ (3,4),\ (3,5),\ (4,5) & \text{// From } P_2 \end{array} \right\}$$

  Above candidate pairs = 8 (Much less than $N^2 = 5^2 = 25$).

- $\mathbf{G_p}$:
  Group of Pairs for Paragraph p. This is the subset of candidate pairs from E_{probe} that were specifically generated because both nodes appeared together in paragraph P_p. This group G_p is used for batching the LLM calls — the text of paragraph P_p and the list of pairs G_p are sent to the LLM in a single query.

  Ex:

$$G_1 = [(1, 2), (1, 3), (2, 3)]$$

  (The pairs generated from paragraph P_1. These pairs are sent along with the text of P_1 to the LLM.)

$$G_2 = [(3, 4), (3, 5), (4, 5)]$$

  (The pairs generated from paragraph P_2. These pairs are sent along with the text of P_2 to the LLM.)

3.3 On-the-fly Mediator-Controlled Causal Prior Generation (C_{prior}) Generation

Our goal here is to create the sparse "answer key" C_{prior} by efficiently and intelligently querying the teacher LLM.

Our C_{prior} is kinda a high-quality "training dataset" that tlls us the true causal links in the document. Since we have no human labels, we must generate the dataset ourselves. This is a self-supervised process.

The core idea is to use a large, powerful "Teacher" LLM (e.g., GPT-4o) to perform complex causal reasoning. The output of this phase is the C_{prior} (Causal Prior).

C_{prior}

The C_{prior} is a sparse training dataset of direct causal effect scores.

- **Form:** It is a list of tuples (i, j, score), where i is a *subjectnodeid*, j is an *objectnodeid*, and score is a float $\in [0.0, 1.0]$.
- **Purpose:** It serves as the supervisory signal (or "ground truth") for the L_{causal} (Causal Loss) function.
- **Meaning:** A score of 1.0 means the "Teacher" LLM has determined there is a direct causal link $i \to j$. A score of 0.0 means there is no direct link.
- **Process:** Our CausGT-HS GNN (the "Student" model) is trained to replicate these scores. This process is known as **Knowledge Distillation**, where we distill the slow, complex, symbolic reasoning of the "Teacher" LLM into the fast, numerical, graph-based architecture of the "Student" GNN.

3.3.1 Active Candidate-Set Expansion (ACE):

The Core Problem: Now we have N nodes (for N entities) We cant ask the "Teacher" LLM to evaluate all $N \times N$ possible pairs of nodes. For a document with 50000 nodes, this is 2.5 billion queries, which is computationally and financially impossible.

We introduce **Active Candidate-Set Expansion (ACE)**, a multi-stage filtering cascade. The goal of ACE is to intelligently and efficiently prune the $O(N^2)$ search space down to a small, high-quality, high-recall list of candidate pairs, E_{prior}. This small list is what we actually send to the LLM for expensive causal reasoning. This process consists of 2 filtering stages:

- Structural Filter
- Semantic Filter

1. **Structural Filter (GAE):**

   Goal: Here our goal is to find all structurally plausible candidate pairs, including non-obvious, multi-hop pairs that were missed by our initial (local-only) graph extraction.

   Initial Problem: Our initial graph matrices A_W are "myopic" (local). They only contain 1-hop links found within the same paragraph. We need an unsupervised way to find pairs (i, j) that are strongly connected via multi-hop paths (e.g., $i \to k \to l \to j$), as these are highly plausible candidates for causal relationship.

Architecture: A GAE (Graph Autoencoder). GAE is an unsupervised GNN architecture ideal for this task. It consists of two components:

(a) **Encoder ($f_{encoder}$):** A GNN that compresses each node's structural neighborhood into a low-dimensional embedding.

(b) **Decoder ($f_{decoder}$):** A function (dot product) that reconstructs the graph by predicting links between nodes based on the similarity of their embeddings.

Justification for GAE training: The GAE is trained on a *pretext task*: reconstructing the "dumb," local, 1-hop graph $A_{\text{co-occur}}$. We do this not because we want to reconstruct $A_{\text{co-occur}}$, but because in order to succeed at this task, the Encoder (f_{enc}) is forced to learn a "smart" latent embedding space (Z).

In this space, nodes that are structurally related via multi-hop paths (e.g., $i \to k \to j$) are placed close together. We then exploit this embedding space Z to find the multi-hop links that $A_{\text{co-occur}}$ was missing.

Justification for 2-Layer GCN: We use a 2-layer GCN as our encoder f_{enc}. This is a deliberate hyperparameter choice to balance two opposing problems:

- **Myopia (1 layer):** A 1-layer GCN is "short-sighted." Its embedding Z_i only contains information from its 1-hop neighbors. It cannot discover the 2-hop path $i \to k \to j$.
- **Over-Smoothing (L layers, e.g., $L = 10$):** After many layers of message-passing, the embeddings of all nodes in a connected graph converge to the same value, "forgetting" all local structure.
- **Sweet Spot (2–3 layers):** A 2-layer (or 3-layer) GCN is the "sweet spot" — deep enough to capture crucial 2-hop and 3-hop paths, yet shallow enough to avoid over-smoothing.

Detailed overview:

- **Input:**

$$A_{\text{co-occur}} \in \{0,1\}^{N \times N}$$

  - Its a sparse, unweighted, symmetric "scaffolding" graph. It is the binary union of all K initial weighted matrices:

$$A_{\text{co-occur}}(i,j) = 1 \quad \text{if } \exists\, k \text{ such that } W_k(i,j) > 0$$

  - Dimensions: $N \times N$ (N = total nodes/entities)
  - Property: Its extremely sparse. Stored in a format like COO or CSR (i'll decide later (it happens to store only $O(|E|)$ memory (i.e., no. of edges))) (Allows fast row access, matrix-vector multiplication, and message passing operations — critical in GNNs and GTNs.). So above $|E|$ are the number of non-zero entries (edge) $|E| \ll N^2$.

$$X \in \mathbb{R}^{N \times d_{\text{in}}} :$$

  - Initial node features (pretrained embeddings of node names)
  - Dimensions: $N \times d_{in}$ (d_{in} are our initial embeddings).

- **GAE Encoder ($f_{encoder}$):**

  Layer 1:

$$H^{(1)} = \text{ReLU}\left(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}XW_0\right)$$

$$W_0 \in \mathbb{R}^{d_{\text{in}} \times d_h} :$$

Learnable weight matrix (e.g., [384 $\times$ 128]). d_h is the hidden dimension.

$$\hat{A} = A_{\text{co-occur}} + I :$$

Adjacency matrix with self-loops (sparse).

$$\hat{D} :$$

Diagonal degree matrix of $\hat{A}$.

$$\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2} :$$

Symmetrically normalized adjacency matrix.

Efficiency: The operation $\hat{A} \cdot X$ is not a dense $O(N^2)$ multiplication. It is a Sparse Matrix-Matrix Multiplication (SpMM) with complexity

$$O(|E| \cdot d_{\text{in}}),$$

which scales linearly with N (assuming $|E|$ scales linearly with N).

$$H^{(1)} \in \mathbb{R}^{N \times d_h} :$$

Matrix of 1-hop-aware node embeddings (e.g., [50,000 $\times$ 128]).
—

Layer 2 (Encoder Output):

$$Z = \hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}H^{(1)}W_1$$

$$W_1 \in \mathbb{R}^{d_h \times d_z} :$$

Learnable weight matrix (e.g., [128 $\times$ 64]). d_z is the final latent dimension.

Efficiency: This is also a sparse SpMM operation with complexity

$$O(|E| \cdot d_h).$$

$$Z \in \mathbb{R}^{N \times d_z} :$$

The final latent embedding matrix (e.g., [50,000 $\times$ 64]). The vector Z_i now encodes structural information about node i's 2-hop neighborhood.

- **GAE Decoder ($f_{decoder}$):**

  Equation:

$$\hat{A} = S_{\text{GAE}} = \sigma(ZZ^\top)$$

$$ZZ^\top \in \mathbb{R}^{N \times N} :$$

  This is the conceptual $N \times N$ matrix of predicted link probabilities. We do *not* compute this dense matrix explicitly, as it would require an $O(N^2 d_z)$ operation.

- **Training (Loss function):**

  Training Objective: We train the parameters W_0 and W_1 using a sampled binary cross-entropy loss. The loss is computed only on the *positive* links (from $A_{\text{co-occur}}$) and a random sample of *negative* links.

$$L = -\sum_{(i,j)\in\mathcal{P}} \log\left(\sigma(Z_i Z_j^\top)\right) - \sum_{(i,k)\in\mathcal{N}} \log\left(1 - \sigma(Z_i Z_k^\top)\right)$$

  The above equation is Binary cross entropy specialised for GAE. It measures how well the model reconstructs the observed links (positive samples) while penalizing incorrect predictions on non-existent links (negative samples).

  - $\mathcal{P}$: The set of positive links, where $A_{\text{co-occur}}(i,j) = 1$. $|\mathcal{P}| = |E|$.
  - $\mathcal{N}$: A random sample of negative links, where $A_{\text{co-occur}}(i,k) = 0$. We set $|\mathcal{N}| = |\mathcal{P}|$.
  - **Efficiency:** The dot products $Z_i Z_j^\top$ are computed *on-the-fly* only for these $2 \cdot |E|$ pairs. The total training complexity is $O(|E| \cdot d_z)$, which is linear and highly scalable.

Dual-Stream Causal-Transformer

Relation between $\hat{A}$ and S_{GAE}: The matrices $\hat{A}$ and S_{GAE} are *mathematically identical*:

$$S_{\text{GAE}} \equiv \hat{A} = \sigma(ZZ^\top)$$

Their distinction lies only in context:

- **During GAE Training:** Referred to as $\hat{A}$ (the *predicted adjacency matrix*). Purpose: Compared with the true graph $A_{\text{co-occur}}$ to compute the loss

$$L = \text{BCE}(\hat{A}, A_{\text{co-occur}})$$

- **After Training:** Referred to as S_{GAE} (the *the structural plausibility score matrix*). Purpose: Used to select the Top candidate set-1 most plausible new links

In summary, $\hat{A}$ is used for *training*, while S_{GAE} is used for *inference*.

- **Output (Candidate Set 1):** Now we need to find the best candidate pairs (i, j) to check for causal link. Based on above this would be N^2, this was technically represented in of S_{GAE} matrix. We cannot compute or store this matrix (like think if $N > 100000$).
  Hence its better to analyze only over the most important pairs from this logical matrix without actually computing the whole thing.
  Hence we use **Approximate Nearest Neighbors (ANN)** on the latent embedding space Z.
  A pair (i, j) will only have a high score in S_{GAE} if their embeddings (Z_i and Z_j) are very close in the 64-dimensional latent space.
  So the problem changes from instead of "find the top-Kpairs in an $N \times N$ matrix" the problem becomes: for each node i, find its Top-$k^{'}$ closest neighbours in the N-node embedding space.
  (a) We first build an ANN index (FAISS to be specific) on the N vectors in Z.
  Complexity $= O(Nd_z logN)$
  (b) For each node $i \in N$, query the index for its Top-$k^{'}$ nearest neighbours in the latent space. (Note: $k^{'}$ is a "small" hyperparam (maybe 50 idk)).

$$\text{Neighbours}_i = \text{ANNIndex.search}(Z_i, k')$$

  (c) Candidate Set 1 is the union of all these pairs:

$$C_1 = \bigcup_{i \in \mathcal{N}} \{(i, j) \mid j \in \text{Neighbors}_i\}$$

  Our candidate Set 1 C_1 is now our final list of K_1 pairs that are structurally plausible (i.e., "close" in the 2-hop embedding space).

  Our Brute force approach was $O(N^2)$ and our ANN approach is now $O(N.k^{'})$ (linear).

  Let K_1 be size of our resulting set, $K = |C_1|$. Its maximum size as inferred above is $N.k^{'}$. Linear.

2. **Semantic Filter (RAG-based with RAV verification):**

   Goal: Here our basic goal is to filter the large K_1 "structurally plausible" set down to a small, K_2 **"semantically plausible"** set. This pipeline is designed for maximum accuracy, speed, and robustness against hallucinations.

   Initial Problem: Our GAE (in previous step) is "semantically blind" and will find structurally plausible but nonsensical pairs.
   Example: Consider a knowledge graph where nodes represent entities and events:

$$\text{Company A's Stock},\ \text{News Article X},\ \text{Company B's Stock},\ \text{Government Policy}.$$

   Edges denote co-mentions in news reports. The Graph Autoencoder (GAE) observes the 2-hop path

$$(\text{Company A's Stock}) \rightarrow (\text{News Article X}) \rightarrow (\text{Company B's Stock})$$

   and predicts a high reconstruction score for the edge $(\text{Company A's Stock}, \text{Company B's Stock})$. Although this connection is structurally plausible, it is semantically nonsense. the two companies merely co-occur in the same article and may not be semantically or causally related. Hence, the GAE is said to be **semantically blind**—it captures structural proximity but ignores causal or semantic meaning.

Solution: We combine RAG-HyDE (Hypothetical Document Embeddings) for high-quality retrieval with RAV (Retrieval-Augmented Verification) to ground the LLM's generations and prevent hallucinations.

Architecture: Here we use a LLM for 3 roles:

- A Hypothetical generator ($f_{hypothetical}$): Generates hypothetical answer snippets.
- A verifier ($f_{verifier}$): Verifies it own generations against retrieved facts.

Detailed Overview:

- $S = [s_1, \ldots, s_M]$**:** This is the list of all M sentences from our document.

  $f_{\textbf{embed}}$**:** A pre-trained sentence embedding model (e.g., `all-MiniLM-L6-v2`, with embedding dimension $d_{\text{embed}} = 384$).

  $V_D \in \mathbb{R}^{M \times d_{\textbf{embed}}}$**:** The **Document Vector Store** — a dense matrix where each row corresponds to a sentence embedding:

  $$V_D(m) = f_{\text{embed}}(s_m)$$

  `idx_doc`**:** An Approximate Nearest Neighbor (ANN) index (e.g., FAISS) built on V_D for efficient k-NN search.

- **Filtering:**
  We do this below for every (i, j) in the Candidate set 1.

  (a) **Hypothetical Document Generation ($f_{hypothetical}$)**: We query the LLM with the following instruction to synthesize plausible hypotheses about the possible causal connection between nodes:

  prompt = `"Generate` $k_{\text{hypothetical}}$ `short, hypothetical sentences describing a plausibl`

  This produces a set of $k_{\text{hypothetical}}$ generated hypotheses:

  $$H = [h_1, h_2, \ldots, h_{k_{\text{hypothetical}}}]$$

  Each hypothesis is embedded to produce dense representations:

  $$[v_{k_1}, v_{k_2}, \ldots, v_{k_{\text{hypothetical}}}]$$

  To mitigate the risk of *hallucination drift*—wherein LLMs produce semantically coherent yet unsupported statements—we verify the generated hypotheses through retrieval-based grounding and semantic consistency estimation.

  Hypothesis Verification via RAG + Semantic Entropy Filtering:

  Given the set of hypothetical claims $H = [h_1, h_2, \ldots, h_{k_{\text{hypothetical}}}]$, the goal is to identify a grounded subset H_{verified} that is both factually supported by document evidence and semantically consistent under model uncertainty.
  Step 1: Initialization. Initialize an empty list to store verified hypotheses:

  $$H_{\text{verified}} \leftarrow [\,]$$

Step 2: Evidence Retrieval via RAG. For each $h_l \in H$, retrieve top-k_{RAG} supporting document snippets using the embedding-based retriever:

$$\vec{v}_{h_l} \leftarrow f_{\text{embed}}(h_l)$$

$$\text{Indices}_V \leftarrow \texttt{ANN\_Search}(\text{index} = \texttt{idx\_doc}, \text{query} = \vec{v}_{h_l}, k_{\text{RAG}} = 3)$$

$$\texttt{verification\_snippets} \leftarrow \text{Join}\big(\{S[\text{idx}] \mid \text{idx} \in \text{Indices}_V\}\big)$$

These snippets act as the local context to check factual support for the hypothesis.
Step 3: LLM Self-Check Verification. Each hypothesis h_l is paired with its retrieved evidence and passed to the LLM verifier f_{verify}:

$$\text{prompt}_{\text{Verify}} = "Claim :'$$

h$_l$'. Evidence: 'verificationsnippets'. Does the Evidence strongly support the Claim? YES or NO."

The verifier's response is converted into a probabilistic confidence score:

$$p_{\text{support}}(h_l) = f_{\text{verify}}(\text{prompt}_{\text{Verify}}) \in [0, 1]$$

This score measures the factual alignment between the claim and the retrieved evidence.
Step 4: Semantic Entropy Estimation ($\mathcal{H}_{semantic}$). To capture epistemic uncertainty in the verifier's reasoning, we perform multiple stochastic forward passes with different random seeds or temperature-scaled sampling:

$$P(Y \mid X, \theta_T) = f_{\text{verify}}(X; T)$$

Across R independent samples, let p_r represent the normalized probability assigned to each semantic outcome (e.g., "YES," "NO," or paraphrastic variants). Semantic entropy is defined as:

$$\mathcal{H}_{semantic}(h_l) = -\sum_{r=1}^{R} p_r \log p_r$$

A low $\mathcal{H}_{semantic}$ indicates internal consistency and strong model confidence, whereas high $\mathcal{H}_{semantic}$ signifies semantic disagreement or possible hallucination.
Step 4.1: Temperature–Entropy Relationship. The temperature parameter T in LLM decoding directly affects the entropy of generated responses:

$$P(y_i \mid x) = \frac{\exp\left(\frac{z_i}{T}\right)}{\sum_j \exp\left(\frac{z_j}{T}\right)}$$

where z_i denotes the unnormalized logit for token i. A lower temperature ($T \to 0$) sharpens the output distribution, leading to deterministic but potentially overconfident responses and thus reducing $\mathcal{H}_{semantic}$. Conversely, a higher temperature increases stochasticity and entropy, amplifying semantic variation and potential disagreement.
Empirical studies suggest that a **moderate temperature** ($T \in [0.6, 0.8]$) provides the best calibration—balancing semantic diversity with factual stability. This ensures that $\mathcal{H}_{semantic}$ reflects genuine model uncertainty rather than random sampling noise.

Step 5: Dual Filtering (Evidence + Entropy). A hypothesis is retained only if both factual support and semantic stability conditions are satisfied:

$$p_{\text{support}}(h_l) > \tau_{\text{support}} \quad \text{and} \quad \mathcal{H}_{semantic}(h_l) < \tau_{\text{entropy}}$$

$$\text{If satisfied: } H_{\text{verified}} \leftarrow H_{\text{verified}} \cup \{h_l\}$$

Step 6: Output. Return the final grounded and semantically consistent set of hypotheses:

$$\textbf{return } H_{\text{verified}}$$

This verification pipeline ensures that H_{verified} represents only those hypothetical causal statements that are (a) factually supported by the retrieved corpus and (b) semantically stable under model uncertainty. By integrating $\mathcal{H}_{semantic}$ as an intrinsic uncertainty estimator, the framework minimizes hallucination risk and promotes robust, evidence-aligned causal reasoning.

(b) **Adaptive Pooling and Multi-query Rank Fusion (RAG-MMR):**

Key Definitions and Parameters

- $k_{\textbf{pool}}$ — Adaptive candidate pool size, dynamically determined by verification confidence:
$$k_{\text{pool}} = k_{\text{base}} + (1 - \text{score}) \cdot k_{\text{expansion}}$$
where score is derived from semantic entropy or verification consistency.
- $k_{\textbf{RAG}}$ — Final number of top snippets chosen after MMR reranking.
- λ — Controls the trade-off between relevance and diversity in MMR.
- κ — Constant in RRF that smooths reciprocal rank influence.

i. **Adaptive Candidate Pooling:** For each verified hypothesis $h \in H_{\text{verified}}$ with a confidence score, dynamically set the pool size:

$$k_{\text{pool}}(h) = k_{\text{base}} + (1 - \text{score}_h) \cdot k_{\text{expansion}}$$

Higher uncertainty (lower score or higher entropy) results in larger pools, allowing the retrieval system to explore broader context.

ii. **MMR Reranking (Balancing Relevance and Diversity):** Retrieve k_{pool} nearest candidates using approximate nearest neighbor (ANN) search:

$$\text{Indices}_{h,\text{pool}} = \text{ANNSearch}(\text{index} = \texttt{idx\_doc}, \text{query} = \vec{v}_h, k = k_{\text{pool}}(h))$$

Then apply the Maximal Marginal Relevance (MMR) criterion:

$$\text{MMRScore}(d) = \lambda \cdot \text{sim}(\vec{v}_h, \vec{v}_d) - (1 - \lambda) \cdot \max_{j \in \text{Selected}} \text{sim}(\vec{v}_d, \vec{v}_j)$$

Select the top k_{RAG} items as the diverse final list for each hypothesis.

iii. **Rank Fusion (Reciprocal Rank Fusion — RRF):** After MMR, each hypothesis h yields a short ranked list of its most relevant snippets. We now combine these lists using RRF, a simple yet robust fusion technique that rewards consistency across hypotheses.

$$\text{RRFScore}(d) = \sum_{h \in H_{\text{verified}}} \frac{1}{\kappa + \text{rank}_h(d)}$$

- $\text{rank}_h(d)$ — Rank position of document d in hypothesis h's list (1 for top item, 2 for next, etc.).
- If d does not appear in list h, it is either ignored or assigned a large rank value.
- κ (typically 60–100) dampens small-rank dominance and prevents a single top item from overwhelming the score.

Intuition: Snippets that appear consistently near the top across multiple hypotheses accumulate higher scores, producing a consensus-driven ranked list that is both relevant and stable across query variations.

iv. **Final Fusion and Deduplication:** Combine all hypothesis-specific lists into one unified set:

$$\text{Indices}_{\text{total}} = \bigcup_{h \in H_{\text{verified}}} \text{FinalIndices}_h$$

Sort by $\text{RRFScore}(d)$ and select the top entries for downstream RAG context.

Where Semantic Entropy Exactly Plugs In:

- The verification score score_h can be derived from semantic entropy inversion, or equivalently:

$$\text{score}_h = p_{\text{support}} \times (1 - \text{normalized variance})$$

- It controls the adaptivity of the retrieval system:
  - **Lower confidence (high entropy):** increase k_{pool}, raise MMR diversity by reducing λ, or increase sampling temperature.
  - **Higher confidence (low entropy):** shrink k_{pool}, tighten λ toward 1 for more focused retrieval.

Pseudocode:

```
# Adaptive Pooling and Multi-query Rank Fusion (RAG-MMR)

# Step 1: Adaptive Candidate Pooling
all_ranked_lists = []
for (h, score) in H_verified_scores.items():
    # Dynamically adjust pool size based on confidence
    k_pool = int(k_base + (1 - score) * k_expansion)

    # Retrieve top-k_pool candidates via ANN search
    pool_indices, pool_vectors = ANN_Search(idx_doc, v_h, k=k_pool)

    # Step 2: MMR Reranking (Relevance{Diversity balance)
    final_list_h = MMR_rerank(
        query_vec=v_h,
        candidates=pool_vectors,
        k=k_RAG,
        lambda_value=lambda_mmr
    )

    all_ranked_lists.append(final_list_h)

# Step 3: Rank Fusion (Reciprocal Rank Fusion | RRF)
RRF = dict()
for list_h in all_ranked_lists:
    for rank, doc in enumerate(list_h, start=1):
        RRF[doc] = RRF.get(doc, 0.0) + 1.0 / (kappa + rank)
```

```
# Step 4: Final Context Selection
final_sorted = sort_by_value_desc(RRF)
Indices_total = final_sorted[:k_FINAL]

# Output: Adaptively pooled, diverse, and consensus-fused context snippets
```

Complexity and Efficiency Notes:

- Over-fetching large pools for many hypotheses is computationally heavy; adaptive pooling helps but worst-case cost scales with total hypotheses.
- Cache results — similar hypotheses often retrieve overlapping candidates.
- Early de-duplication (by document ID) reduces downstream RRF computation.
- Apply a global retrieval budget to prevent explosion in total retrieved snippets.

Pitfalls and Practical Considerations:

- **Over-expansion:** Very low verification scores may inflate k_{pool} excessively — cap it at a maximum threshold.
- **Noisy verification scores:** Entropy-based scores can fluctuate; use exponential moving averages for stability.
- **MMR tuning:** Too small λ yields irrelevant "diverse" snippets; too large λ kills diversity.
- **RRF κ sensitivity:** κ too small $\rightarrow$ rank-1 items dominate. κ too large $\rightarrow$ all contributions flatten, weakening discrimination.

Result: A final ranked set of snippets that are adaptively pooled, semantically diverse, confidence-weighted, and consensus-fused — forming the optimal retrieval substrate for downstream RAG generation.

MMR (Maximal Marginal Relevance)

Problem: Retrieved passages in RAG are often redundant, repeating similar information.

Idea: MMR balances *relevance* to the query with *diversity* among selected passages.

Formula:

$$\text{MMR} = \arg\max_{d_i \in D \setminus S} \left[\lambda \cdot \text{Sim}(d_i, q) - (1 - \lambda) \cdot \max_{d_j \in S} \text{Sim}(d_i, d_j) \right]$$

Where:

- D: set of all retrieved documents/snippets
- S: set of already-selected documents
- d_i: candidate document not yet selected
- q: query embedding
- $\text{Sim}(a, b)$: cosine similarity between a and b
- λ: trade-off parameter ($0 \leq \lambda \leq 1$) — higher values favor relevance, lower values favor diversity

Interpretation: Select each new passage that is highly relevant to the query but minimally similar to what's already chosen, ensuring diverse and informative retrieval.

Outcome: Produces a complementary, non-redundant set of retrieved passages for richer RAG context.

(c) **Build Dynamic Context and Run Causal Plausibility Classifier (CPC):**

i. **Executive summary.** This section describes the offline workflow to construct a high-quality, multi-label dataset and to train a fast, deployable Causal Plausibility Classifier (CPC). The workflow has two major parts: (A) *Dataset construction (for CPC) and Teacher LLM labeling* and (B) *Model training and calibration*. The CPC is designed as a discriminative cross-encoder (DeBERTa-v3 base) with three task-specific heads (Plausible, Temporal, Mechanistic). The trained CPC is followed by post-hoc calibration so outputs become reliable probabilities used for downstream filtering rules.

Note: We are not yet doing the unidirectional causal check. This step is intentionally bidirectional. The CPC model's input "`[CLS] {context} [SEP] {nodei} [SEP] {nodej} [SEP]`" will produce the same logits as "`[CLS] {context} [SEP] {nodej} [SEP] {nodei}`".
In this Active Candidate Expansion (ACE) phase, the ultimate goal is **plausibility**, not directionality. It only reduces, say, 1M pairs to 10K pairs — like asking, "Are i and j related in a way worth checking for causality?"
The unidirectional check ($i \to j$ vs. $j \to i$) is the work of the next phase: **Mediator-Controlled Counterfactual Querying (MCC-LME)**.

Part A: Dataset construction (for CPC) and Teacher LLM labeling

A.1 Motivation and high-level design

The goal of Part A is to produce a large, balanced, and robust multi-label dataset of tuples

$$(\text{node}_i, \text{node}_j, \text{highqualitycontext})$$

together with multi-task labels and a short rationale per example. This dataset will be used to train the CPC cross-encoder. High-quality contexts are retrieved with a full retrieval pipeline (RAG-HyDE-RAV) and structural plausibility candidates are proposed by a Graph Auto-Encoder (GAE) + ANN index.

A.2 Key label definitions (report-style explanation)

Below we define the three label axes used by the CPC. These appear as independent binary targets and are the core of the multi-task design.

Plausibility (Adjective: Plausible). Formal definition: Plausibility measures the logical and semantic coherence of a potential relationship between two nodes in the given context. It answers: "Is it possible and sensible that a connection exists between these two concepts, given the context?"
In simple terms: "Does this link make sense at all, or is it nonsense?" This is the primary keep/discard filter.
Examples:

- *Plausible:* ("XGBoost", "accuracy") — an algorithm vs. a performance metric; the relation is sensible.
- *Implausible:* ("XGBoost", "18th Century France") — normally implausible unless the context is extremely specific.

Temporality (Adjective: Temporal). Formal definition: Temporality measures whether the text provides evidence of a time-based order between the two nodes. True causality requires cause preceding effect.
In simple terms: "Is there a time-order? Does the text say one thing led to or followed another?"
Examples:

- Context: "The implementation of XGBoost (Node i) led to a 95% accuracy (Node j)."
  Label: `labeltemporal = YES` (phrase "led to" indicates order).
- Context: "We discuss XGBoost (Node i) and accuracy (Node j)."
  Label: `labeltemporal = NO` (no order implied).

Mechanism (Adjective: Mechanistic). Formal definition: Mechanism measures whether the text describes a process, pathway, or means by which one node influences the other (the "how").
In simple terms: "Does the text explain how i affects j or through what means?"
Examples:

- Context: "XGBoost (Node i) achieves high performance (Node j) by using gradient boosting principles."
  Label: `labelmechanistic = YES`.

– Context: "XGBoost (Node i) achieved high performance (Node j)."
Label: `labelmechanistic = NO` (no explanation of the mechanism).

A.3 Step 1 — Generate weak labels (unlabeled dataset)

Objective: Build a massive pool of tuples (nodei, nodej, context) from a diverse corpus (e.g., 10k PDFs) that will be annotated by a Teacher LLM.

Procedure:

i. **Corpus collection:** Collect heterogeneous documents across domains (scientific articles, technical docs, web corpora), target scale: tens of thousands of files.
ii. **Run ACE Stage 1 (GAE pass):** For each document, extract node lists N and initial scaffolding adjacency $A_{\text{co-occur}}$. Train or apply the Graph Auto-Encoder (GAE) and build an ANN index over learned structural embeddings.
iii. **Candidate generation:** Use the ANN index to propose millions of structurally plausible (i, j) candidate pairs, prioritizing structural plausibility while ensuring semantic coverage.
iv. **Context retrieval (RAG-HyDE-RAV):** For each candidate pair, run the full retrieval + HyDE reranking pipeline to fetch the highest-quality supporting context string.
v. **Output:** Produce a large unlabeled file of tuples:
$(\text{nodeiname}, \text{nodejname}, \text{highqualitycontextstring})$. Example scale: $O(10^7)$ tuples.

A.4 Step 2 — Teacher LLM labeling and rationale augmentation

Objective: Use a high-capability Teacher LLM to annotate each tuple with a one-sentence rationale and three binary labels (plausible, temporal, mechanistic).

Prompt and output schema: Provide the Teacher LLM with the retrieved context and pair, and request a structured JSON like:

```
{"rationale": "...", "label_plausible": "YES"|"NO",
 "label_temporal": "YES"|"NO", "label_mechanistic": "YES"|"NO"}
```

Quality controls:

– Enforce instruction-following: single-sentence rationale, exactly the JSON schema, conservative labeling guidelines.
– Sample-based human validation: periodically validate Teacher outputs to ensure accuracy and reduce systematic bias.

Offline output: `labeleddata.jsonl` — each line is a JSON object containing the context, the pair, the rationale, and the three labels.

A.5 Example `labeleddata.jsonl` (three samples)

The following verbatim entries are representative lines from the `labeleddata.jsonl` file:

Example Causal-Plausibility Data Points

Datapoint 1

```
{
  "context": "We evaluated XGBoost on the ImageNet dataset (IMG). The
  ↪  performance achieved...",
  "pair": ["performance", "accuracy"],
  "rationale": "The context states that 'performance' (the subject) 'achieved'
  ↪  the 'accuracy'...",
  "label_plausible": "YES",
  "label_temporal": "YES",
  "label_mechanistic": "NO"
}
```

Datapoint 2

```
{
  "context": "The XGBoost model (XGB) utilizes gradient boosting (GB)
  ↪  principles.",
  "pair": ["XGBoost", "gradient boosting"],
  "rationale": "The text explicitly states that XGBoost 'utilizes' gradient
  ↪  boosting...",
  "label_plausible": "YES",
  "label_temporal": "YES",
  "label_mechanistic": "YES"
}
```

Datapoint 3

```
{
  "context": "Both XGBoost and LightGBM are popular GBDT frameworks...",
  "pair": ["XGBoost", "LightGBM"],
  "rationale": "The text only *compares* these two entities as peers...",
  "label_plausible": "NO",
  "label_temporal": "NO",
  "label_mechanistic": "NO"
}
```

A.6 Step 3 — Generate adversarial (hard) negatives

Goal: Create hard negatives to prevent trivial shortcuts and to increase classifier robustness.

Definition (Adversarial Hard Negative): A pair (i, k) that is semantically similar to a positive example but structurally distant from i in the GAE embedding space. Example: for positive $(\text{XGBoost}, \text{accuracy})$, a hard negative might be $(\text{XGBoost}, \text{LightGBM})$ if LightGBM is semantically similar to XGBoost but not causally linked.

Procedure:

i. For each positive (i, j), find candidate node k such that $\text{semanticsim}(j, k)$ is high but $\text{structuralsim}_{GAE}(i, k)$ is low (e.g., < 0.1).

ii. Retrieve context for (i, k) using RAG-HyDE.

iii. Label the adversarial candidate with the Teacher LLM; expect `labelplausible = NO` in most cases.

iv. Construct a final balanced training dataset by sampling:
   - 50% Positive examples (Teacher says `YES`),
   - 25% Easy negatives (random Teacher `NO`),
   - 25% Hard negatives (adversarial, Teacher-labeled).

Offline output: `traindataset.jsonl` (e.g., 1M examples, balanced as above).

Part B: Model training, calibration and final deliverables

B.1 Step 4 — Train the Causal Plausibility Classifier (CPC)

Architecture overview (Discriminative Cross-Encoder).

- **Encoder body:** DeBERTa-v3 (or similar transformer).
- **Input format:**

$$[\text{CLS}]\ \text{contextstring}\ [\text{SEP}]\ \text{nodeiname}\ [\text{SEP}]\ \text{nodejname}\ [\text{SEP}]$$

- **Shared representation:** The encoder produces a pooled token embedding H_{CLS} representing the entire input.
- **Three independent heads:** Each head is a linear classifier producing a scalar logit:

$$\begin{aligned}\text{logit}_{\text{plausible}} &= W_{\text{plausible}} \cdot H_{CLS} + b_{\text{plausible}},\\ \text{logit}_{\text{temporal}} &= W_{\text{temporal}} \cdot H_{CLS} + b_{\text{temporal}},\\ \text{logit}_{\text{mechanistic}} &= W_{\text{mechanistic}} \cdot H_{CLS} + b_{\text{mechanistic}}.\end{aligned}$$

Multi-task loss and training. Each head is supervised with a binary cross-entropy loss; the total loss is the sum:

$$L_{\text{total}} = L_{\text{BCE}}\big(\text{logit}_{\text{plausible}}, y_{\text{plausible}}\big) + L_{\text{BCE}}\big(\text{logit}_{\text{temporal}}, y_{\text{temporal}}\big) + L_{\text{BCE}}\big(\text{logit}_{\text{mechanistic}}, y_{\text{mechanistic}}\big).$$

Why multi-task? Joint training forces the shared encoder to learn representations sensitive to plausibility, temporal cues, and mechanism clues. This cross-task signal improves generalization and makes $\text{logit}_{\text{plausible}}$ more informative for downstream filtering.

Training steps (practical notes):

- Fine-tune with AdamW, learning-rate scheduler, mixed precision where available.
- Use stratified shuffling respecting the Positive / EasyNeg / HardNeg ratios.
- Hold out a 10% validation set for calibrator training (see B.2).
- Save best checkpoint by validation loss and per-head AUROC.

B.2 Step 5 — Train calibrators (Isotonic Regression)

Goal: Convert CPC raw outputs (logits or raw probabilities) into well-calibrated probabilities that reflect real-world frequencies. This is essential because the CPC's raw confidences are often miscalibrated (over- or under-confident).

B.2.1 The problem: uncalibrated scores Modern neural classifiers produce logits or probabilities that do not reliably correspond to empirical accuracy. Example: $p_{\text{raw}} = 0.9$ does not necessarily mean "90% chance that the label is YES". Without calibration, thresholding becomes unreliable.

B.2.2 The solution: Isotonic Regression Isotonic Regression is a non-parametric, order-preserving calibrator. It learns a monotonic mapping $f(\cdot)$ from raw scores to calibrated probabilities:

$$p_{\text{calibrated}} = f(p_{\text{raw}}), \quad f \text{ monotone non-decreasing.}$$

Advantages:

- No parametric assumption (flexible).
- Preserves ordering (if scoreA $>$ scoreB then calibratedA $\geq$ calibratedB).

B.2.3 The process:

i. **Hold-out set:** Reserve 10% of the labeled training set as a hold-out (never used to update CPC).

ii. **Get raw outputs:** Run the trained CPC on the hold-out inputs to obtain raw logits (or raw sigmoid probabilities) for each head:

$$\text{logitsplausibleholdout} = [s_1, s_2, \ldots, s_n].$$

iii. **Prepare labels:** Extract the true binary labels,

$$Y_{\text{holdoutplausible}} = [y_1, y_2, \ldots, y_n].$$

iv. **Fit calibrator:** For each head, fit an isotonic regression model:

$$\text{Calibrator}_{\text{plausible}} = \text{IsotonicRegression}().\text{fit}(\text{logitsplausibleholdout}, Y_{\text{holdoutplausible}}).$$

Repeat for temporal and mechanistic heads.

v. **Save:** Serialize calibrator objects (e.g., `Calibrators.pkl`).

B.2.4 Intuition and examples Isotonic regression constructs a stepwise mapping such as:

- If CPC outputs raw scores in $[0.7, 0.8]$, the true empirical frequency may be 0.75, so map that whole segment to 0.75.
- If CPC outputs raw scores in $[0.9, 1.0]$, empirical frequency may be 0.92, so map accordingly.

B.2.5 Using calibrators in the online inference pipeline At inference time:

i. Produce raw scores:

$$\text{logits} = \text{CPCModel}(\text{input}) \rightarrow (\text{logit}_{\text{plausible}}, \text{logit}_{\text{temporal}}, \text{logit}_{\text{mechanistic}}).$$

ii. Convert raw logits to raw probabilities (if logits were used):

$$p_{\text{raw}} = \sigma(\text{logit}), \quad \text{or pass logits directly to calibrator depending on implementation.}$$

iii. Apply calibrators:

$$p_{\text{plausible}} = \text{Calibrator}_{\text{plausible}}.\text{predict}(p_{\text{raw,plausible}}),$$

and similarly for temporal and mechanistic heads.

iv. Use these calibrated probabilities for decision rules: **Threshold Selection:** The thresholds $\tau_{\text{plausible}}$, τ_{temporal}, and $\tau_{\text{mechanistic}}$ are tunable *hyperparameters* that control the strictness of our filtering rule. They determine how confident the model must be before accepting a causal link as valid. Typical empirically chosen values are: $\tau_{\text{plausible}} = 0.5$, $\tau_{\text{temporal}} = 0.3$, and $\tau_{\text{mechanistic}} = 0.3$.

```
# Filtering Rule

if p_plausible > tau_plausible and (
    p_temporal > tau_temporal or
    p_mechanistic > tau_mechanistic
):
    # Keep example: add (i, j) to E_prior
    keep_example(i, j)
else:
    # Discard example
    discard(i, j)
```

B.3 Final deliverables and runtime assets

After training and calibration we produce the following offline assets that are deployed with the live pipeline:

- `CPCModel.bin` — the trained DeBERTa-v3 cross-encoder checkpoint (weights config).
- `Calibrators.pkl` — serialized isotonic regression calibrators for plausible, temporal, and mechanistic heads.
- `traindataset.jsonl` — final balanced training dataset used to train the CPC.
- `validationholdout.jsonl` — hold-out data used for calibrator training and final evaluation.
- `trainingreport.pdf` — metrics (per-head AUROC, calibration curves, reliability diagrams, confusion matrices).

B.4 Some checks for later on here

- **Sanity checks:** Verify that calibrator monotonicity holds and that reliability diagrams improve after calibration.
- **Human-in-the-loop:** Spot-check Teacher LLM labels, especially for adversarial negatives.
- **Edge cases:** Explicitly test on rare domains to ensure the CPC is not overly biased toward common contexts.
- **Monitoring:** In production, log calibrated probabilities and flagged examples for periodic manual review and dataset refresh.

Important equations and macros

- Multi-task BCE loss:

$$L_{\text{total}} = \sum_{t \in \{\text{plausible, temporal, mechanistic}\}} L_{\text{BCE}}\big(\text{logit}_t, y_t\big).$$

– Calibrator training:

$$\text{Calibrator}_t = \text{IsotonicRegression}().\text{fit}\big(\text{scores}_{t,\text{holdout}}, \text{labels}_{t,\text{holdout}}\big).$$

The output":

Its a final K_2-sized list E_{prior}, which is now of extremely high quality and ready for the "Teacher" CoCaD (Counterfactual Causal-DP) causal reasoning.

3.3.2 CoCaD (Counterfactual Causal-DP)

Here our goal is to pick these E_{prior} list tuples and create our C_{prior} (Which happens to be a sparse list of direct causal links $(i, j, score)$ which serves as the answer key for training our main CausGT-HS GNN model).

Actually a fundamental goal of automated knowledge discovery from unstructured text is to move beyond simple correlational graphs $(i - j)$ to the extraction of directed, causal mechanisms $(i \to j)$. This is the difference between an "information map" and an "explanation map". However, extracting causal graphs from static, observational text happens to be notoriously difficult, ill-posed, as it is plagued by two primary forms of statistical error:

1. **Confounding (Spurious Correlation):** Two variables, i and j, may appear strongly correlated because they both are influenced by a third, often unobserved, common cause k (called a confounder).
   Example: Ice Cream Sales (node i) and Crime Rates (node j) are strongly correlated, but neither causes the other; both are caused by Hot Weather (node k). A naive model will incorrectly learn the link $i \to j$ (Hence making it look as if Ice cream sales caused the rise in crime rates).
2. **Mediation (Indirect Causation):** A variable i may have no direct causal effect on j, but may be linked by a chain of true causes (e.g., $i \to k \to j$). A naive model, observing a strong correlation between i and j, may incorrectly add a "false shortcut" link $i \to j$, which masks the true, explanatory mechanism.

Below we propose CoCaD (counterfactual Causal-DP), a novel self-supervised pipeline. CoCaD is specifically designed to solve both the confounding and mediation problems by systematically generating and pruning a set of causal hypotheses.

We perform this in 2 steps:

- **CoCaD core:** The core engine for reasoning. Its a non-recursive, DP-based pipeline that robustly calculates direct and indirect causal scores.
- **EM-Refinement:** An unsupervised loop to refine the model's parameters.

1. **CoCaD:** Here our goal is to pick these E_{prior} list tuples and create our C_{prior} (Which happens to be a sparse list of direct causal links $(i, j, score)$ which serves as the answer key for training our main CausGT-HS GNN model).

   We do the above in 3 steps:

   (a) **Build W_{direct} (A direct guess map)**: Here our goal is to generate a single, robust, and calibrated score, $W_{direct}(i, j)$, for the direct causal strength of every pair (i, j) in E_{prior}. This score