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

Shashank Tippanavar - IMT2022014
Kushal Jenamani - IMT2022057
Aditya Saraf - IMT2022067

International Institute of Information Technology Bangalore

Contents

1 Introduction

1.1 Github Link

Github Repo Link: https://github.com/MightyShashank/CausGT-HS-EBM-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.

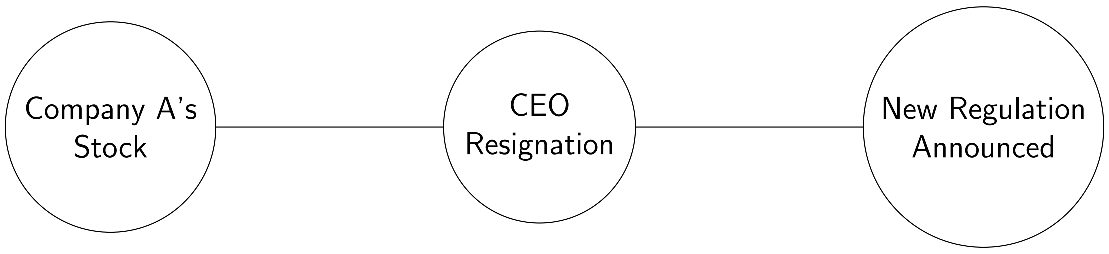
- latent = hidden
- meta-paths = path of classes and not instances
- causal = seeing the cause and not just structural similarity.

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:

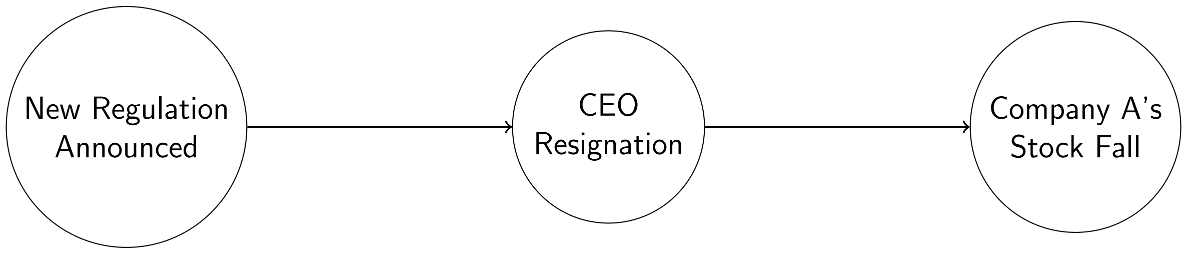


This graph is a "dumb" map. It is flat and undirected. It only tells you that these topics appeared together, but it offers no explanation.

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 ($A \to B$) 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 symetric (same in both directions of edges). They cannot, by design, differentiate between $A \to B$ (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 $A \to C \to B$) 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

* rows$_k[i]$ = source node index of edge i,
* cols$_k[i]$ = destination node index of edge i,
* 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

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

  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:

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

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

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

  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*:

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

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

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

– **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 $= O(N d_z log N)$

  (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 (Company A's Stock, 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.

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

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

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

  `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` $k_{\textsf{hypothetical}}$ `short, hypothetical sentences describing a plausibl`

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

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

  Each hypothesis is embedded to produce dense representations:

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

  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 $H = [h_1, h_2, \ldots, h_{k_{\textsf{hypothetical}}}]$, the goal is to identify a grounded subset $H_{\textsf{verified}}$ 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:

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

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.

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

  - 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 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 semanticsim(j, k) is high but 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:**

  [CLS] contextstring [SEP] nodeiname [SEP] nodejname [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:

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

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:

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

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:

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

– 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 $(i \rightarrow j)$. 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 $i \rightarrow j$ (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., $i \rightarrow k \rightarrow j$). A naive model, observing a strong correlation between i and j, may incorrectly add a "false shortcut" link $i \rightarrow j$, 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 will be clean, meaning it is already pruned of spuriours correlations (confounders).

This is achieved by fusing three distinct evidence components using a learned fusion mode. We use three sources of evidences here $f_{structural}$ (for a structural score based on the neighbourhood), f_{llm} (direct links counterfactual reasoning) and $f_{\texttt{llm\_confounder}}$ (a confounder check using LLMs).

For each pair (i, j) in our E_{prior}:

i. $f_{structural}$ **(The GNN intervention score):**
It sees the effect of absence of a neighbouring node on a node. Meaning "From a purely structural POV (based on our graph), how important is node i to the existence of node j?". We use a bootstrap ensemble of B (say B=5 as of now) pre-trained GNNs ($f_{GNN,1}$, $f_{GNN,2}$,....., $f_{GNN,5}$) to get a robust mean and variance.

For each model b in the ensemble:

A. Predict j's embedding using all its neighbors $\mathcal{N}_j$:

$$Z_{j,b}^{\text{normal}} = f_{\text{GNN},b}(\mathcal{N}_j, X)$$

B. Predict j's embedding without i's influence (i.e., effect of deleting i on j):

$$Z_{j,b}^{\text{intervened}} = f_{\text{GNN},b}(\mathcal{N}_j \setminus \{i\}, X)$$

C. The Calculated Effect

$$\text{score}_b = \text{distance}\big(Z_{j,b}^{\text{normal}}, Z_{j,b}^{\text{intervened}}\big)$$

D. Output is a distribution of B scores:

$$\mu_{\text{GNN}}(i, j) = \text{mean}\big(\{\text{score}_b\}\big)$$
$$\sigma^2_{\text{GNN}}(i, j) = \text{variance}\big(\{\text{score}_b\}\big)$$

ii. f_{llm} **(The LLM Counterfactual):**

Goal: To obtain a robust, interpretable semantic score for the causal link $i \to j$, based purely on textual reasoning while ignoring confounders (i.e., any intermediate causal paths). This component helps approximate the "direct effect" between i and j from static textual data.

Approach: We combine three modern strategies — Retrieval-Augmented Generation (RAG), Chain-of-Thought (CoT) prompting, and Semantic Entropy — into a unified LLM-based causal scoring framework. The pipeline proceeds as follows:

A. **Context Retrieval (RAG-HyDE-RAV):** For each candidate pair (i, j), we retrieve the $k = 3$ most relevant, diverse, and verified text snippets $\{\text{context}_1, \text{context}_2, \text{context}_3\}$ using a hybrid dense retrieval method (HyDE-RAV). These snippets form our input context contextstring that anchors the LLM reasoning.

B. **Counterfactual Reasoning (CoT Prompt):** We query the LLM with a controlled prompt designed to elicit a counterfactual causal explanation:

```
"Context: {...}
1. Reasoning: If i were NOT present, what is the impact on
j? Explain clearly.
2. Give a confidence score between 0.0 and 1.0."
```

This prompt is sampled $N_s = 5$ times at temperature 0.7 to capture reasoning variability. Each sampled response provides:

- A reasoning string r_p
- A causal strength score $s_p \in [0, 1]$

C. **Semantic Entropy and Coherence:** To ensure robustness and interpretability, we compute three additional metrics:

D. **Mean Causal Strength:**

$$\mu_{\text{LLM}}(i, j) = \frac{1}{N_s} \sum_{p=1}^{N_s} s_p$$

Represents the expected strength of causal effect predicted by the LLM.

E. **Variance of Scores (Uncertainty):**

$$\sigma^2_{\text{LLM}}(i, j) = \frac{1}{N_s} \sum_{p=1}^{N_s} (s_p - \mu_{\text{LLM}})^2$$

Captures uncertainty in causal reasoning. High variance indicates inconsistency or lack of confidence.

F. **Semantic Coherence (R_{coh}):** This is necessary because our LLM can be confidently wrong. Convert each reasoning text r_p into an embedding vector v_p using a sentence embedding model (e.g., Sentence-BERT). Compute pairwise cosine similarities and take the average:

$$R_{\text{coh}}(i, j) = \frac{2}{N_s(N_s - 1)} \sum_{p<q} \cos(v_p, v_q)$$

This measures how semantically consistent the reasoning explanations are across multiple LLM generations. A high R_{coh} implies stable reasoning under sampling noise.

G. **TBD: Provenance Storage (Auditability):** All generated reasoning samples $\{r_p\}$ and corresponding scores $\{s_p\}$ are stored for human interpretability and downstream analysis, ensuring full transparency of how each causal edge (i, j) was derived (I'm not sure if we'd be doing this also or not).

iii. f_{CI} **and** $f_{llmconfounder}$ **The Confounder Check:**

Here we solve the confounding. We compute a mathematical score that checks if the $i - j$ link is a "fake" (meaning a spurious correlation caused by a common cause k). So here we check for "common causes" k and re-evaluate the $i \to j$ link.

Confouder $\neq$ Mediator

- **Mediator:**
  $i \to k \to j$
  i causing j through k. Above $i \to j$ is hence a false shortcut. (Our Causal-DP algorithm in Step 1.b here is designed to find this path and delete this "false shortcut" $i \to j$).
  Ex: SortingAlgo (i) $\to$ CPU Usage (k) $\to$ System Crash (j).

- **Confounder:**
  $i \leftarrow k \to j$. This makes node k a common cause: k causes i, and k also causes j. This makes i and j look related, but they are just two puppets controlled by the same string.
  Ex: Ice Cream Sales (i) $\leftarrow$ Hot Weather (k) $\to$ Crime Rates (j). Here our 'Ice Cream Sales' does not cause 'Crime Rates'. The 'fake' link $i \to j$ is hence a spurious correlation. The goal of 1.b.iii is to delete these fake links.

Example:

- Spurious Link: 'Ice Cream Sales' (node i) $\to$ 'Crime Rates' (node j).
- 'Hot Weather' (node k) $\to$ 'Ice Cream Sales' (node i) and 'Hot Weather' (node k) $\to$ 'Crime Rates' (node j)

A. **Find Confounders ($M_{\text{set-confound}}$)**
Our goal is to generate a clean list of potential common causes $M_{\text{set-confound}}$ for the specific pair (i, j) being tested.
A Note on Directionality (Where it is Lost and Regained): The GAE training stage operates on a *symmetric*, undirected structural matrix and therefore loses all causal direction. Directionality is fully re-established in **Step 3.1**, when we run the LLM-based counterfactual queries that produce the asymmetric matrix $W_{\text{direct}}(i, j)$. This matrix becomes the foundation for all downstream causal traversal.

Step 1: Constructing the Directed Causal Graph $A_{\text{causaldir}}$

The directional structure used for confounder discovery comes from the asymmetric, causally meaningful score matrix W_{direct}, computed through the counterfactual LLM queries.
To obtain a sparse adjacency representation, we threshold W_{direct}:

$$A_{\text{causaldir}}(i, j) = \begin{cases} 1, & \text{if } W_{\text{direct}}(i, j) > \tau_{\text{path}} \\ 0, & \text{otherwise.} \end{cases}$$

Where:
- $W_{\text{direct}} \in \mathbb{R}^{N \times N}$: directed causal strengths - $A_{\text{causaldir}} \in \{0, 1\}^{N \times N}$: sparse directed adjacency - τ_{path} (e.g., 0.5) controls strictness of causal inclusion
This resolves the inconsistency: all downstream traversal is performed on a *directed*, causally-grounded graph—never on the symmetric GAE output.

Step 2: Directed Graph Traversal (Find Ancestors)

A node k qualifies as a confounder if it has a directed path leading into *both* i and j under the causal graph $A_{\text{causaldir}}$.

To find these ancestors, we perform a bounded forward BFS on the **transpose** graph $A^T_{\text{causaldir}}$:

$$\text{Ancestors}_i = \{\, k \mid \text{PathExists}(k \to \cdots \to i,\ A_{\text{causaldir}},\ \text{maxhops} = H)\,\}$$

$$\text{Ancestors}_j = \{\, k \mid \text{PathExists}(k \to \cdots \to j,\ A_{\text{causaldir}},\ \text{maxhops} = H)\,\}$$

where H (e.g., $H = 3$) enforces short causal chains and keeps the search computationally light.
The candidate confounders are simply the intersection:

$$M_{\text{set-confound}}(i, j) = \text{Ancestors}_i \cap \text{Ancestors}_j.$$

The resulting set contains node IDs (e.g., $\{12, 45, 101\}$) representing all discovered common ancestors that jointly influence both i and j.

Note

The only purpose of finding a confounder k is to invalidate the direct edge $i \to j$. We do *not* use k to create paths, enhance connectivity, or add new causal edges. Its sole role is to provide evidence that the apparent dependency between i and j is actually explained by a common cause.

Note

The only point of finding k is to prove that the $i \to j$ link is fake. It is not for finding a path from i to j. We find k only so we can use it to prove that the $i \to j$ link is a false correlation and safely delete it from our graph.

Step 3: Topological Regularization:
Objective: To enhance the reliability of the confounder set $M_{\text{setconfound}}$ by removing high-degree, semantically generic nodes that appear as spurious confounders.

Motivation: The raw list $M_{\text{setconfound}}$ often includes overly generic nodes such as "data," "research," or "model." These nodes tend to have very high graph connectivity and thus appear as ancestors of many nodes, even though they are not genuine confounders. To address this, a topological regularization step based on graph centrality measures is applied.

Methodology:

- **Centrality Computation:**
  For every node $k \in \mathcal{N}$, compute its PageRank score (or alternatively, its normalized degree centrality) using the smart adjacency matrix A_{smart}:

$$\text{PR}(k) = \text{PageRank}(k, A_{\text{smart}})$$

- **Hub Identification:**
  Determine the hub threshold τ_{hub} as the 95th percentile of all PageRank values:

$$\tau_{\text{hub}} = P_{95}\big(\{\text{PR}(k) \mid k \in \mathcal{N}\}\big)$$

  The nodes exceeding this threshold are considered graph "hubs":

$$\text{Hubs} = \{\, k \in \mathcal{N} \mid \text{PR}(k) > \tau_{\text{hub}}\,\}$$

- **Confounder Refinement:**
  Remove these high-centrality nodes from the original confounder candidate set:

$$M_{\text{setconfoundfinal}} = M_{\text{setconfound}}(i, j) \setminus \text{Hubs}$$

Result:

$$M_{\text{setconfoundfinal}} = \{\, k \in M_{\text{setconfound}}(i, j) \mid \text{PR}(k) \le \tau_{\text{hub}} \,\}$$

The resulting set contains only the most *plausible*, *specific*, and *non-generic* confounders.

Note: Why Topological Regularization Matters

High-degree nodes (e.g., "data," "study," "system") often create spurious correlations due to their ubiquity across the graph. By filtering these hubs, we ensure that detected confounders represent meaningful, localized causal interactions rather than graph-wide statistical noise.

B. The Dual Check:
Now that we have our clean list of confounders $M_{\texttt{set\_confound\_final}}$ (so we have our k, we use it now to debunk the fake $i \to j$ link), we run two parallel checks to get our final score.

- **The statistical check (f_{CI}):**
  Example: We run $\rho(i, j|k)$: This essentially asks "What is the correlation between "ice cream" and "crime" after we mathematically remove the effect of "Hot Weather"?". The answer is 0 so the fake link disappears.
  Above was a core of our solution but we need to extend it for multiple confounders. Above $\rho(i, j|k)$ takes care of only 1 confounder k.
  The goal of this module is to design a mathematically principled score for the causal link $i \to j$, while rigorously controlling for the influence of all possible confounders. Unlike the naive correlation, which captures both direct and indirect dependencies, f_{CI} isolates the *direct conditional dependence* between nodes i and j.
  - **Motivation:** A standard partial correlation $\rho(i, j \mid k)$ can only control for a *single* confounder variable k. However, real-world graphs often contain multiple confounding factors that jointly influence the observed association between i and j. To handle this, we move from scalar partial correlations to a **matrix-based formulation** using the *Precision Matrix* (Θ), which mathematically encodes conditional dependencies among all variables.
    Specifically:

$$\Theta = \Sigma^{-1}$$

    where Σ is the covariance matrix. The off-diagonal elements Θ_{ij} measure the conditional association between i and j given all other variables in the system.
  - **Step 1: Define Variable Set (V')** We start by defining the set of variables that will participate in the computation:

$$V' = \{i, j\} \cup M_{\text{setconfoundfinal}}$$

    Here:
    * $\{i, j\}$ are the nodes whose direct causal link we want to test.
    * $M_{\text{setconfoundfinal}}$ is the set of confounder nodes identified earlier in the pipeline.

The size of this variable set is denoted by $n' = |V'|$. For instance, if we have nodes $\{i, j, c_1, c_2, c_3\}$, then $n' = 5$.

– **Step 2: Extract Embeddings ($Z_{V'}$)** For every node in V', we retrieve its corresponding embedding vector from the Graph Autoencoder (GAE) (Those embeddings we calculated in the 2-layer Graph Autoencoder (GAE) during ACE). Suppose each node has an embedding of dimensionality d_z (e.g., $d_z = 64$). We organize these embeddings into a matrix:

$$Z_{V'} \in \mathbb{R}^{d_z \times n'}$$

Each column of $Z_{V'}$ corresponds to one node, and each row represents a dimension in the embedding space. We interpret each of the d_z embedding dimensions as independent "observations" across nodes, allowing us to compute covariance across nodes rather than across samples.
For example, if $d_z = 64$ and $n' = 5$, then:

$$Z_{V'} = \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{15} \\ z_{21} & z_{22} & \cdots & z_{25} \\ \vdots & \vdots & \ddots & \vdots \\ z_{64,1} & z_{64,2} & \cdots & z_{64,5} \end{bmatrix}$$

– **Step 3: Compute Covariance Matrix (Σ)** We now compute the empirical covariance matrix across the n' variables:

$$\Sigma = \mathsf{Cov}(Z_{V'}) = \frac{1}{d_z - 1}(Z_{V'}^\top Z_{V'} - \bar{Z}_{V'}^\top \bar{Z}_{V'})$$

where $\bar{Z}_{V'}$ represents the mean-centered embeddings. Thus, $\Sigma \in \mathbb{R}^{n' \times n'}$ and each element Σ_{pq} captures how strongly the embeddings of node p and node q co-vary across all embedding dimensions.
Example (for $n' = 3$ nodes):

$$\Sigma = \begin{bmatrix} \mathsf{Var}(i) & \mathsf{Cov}(i, j) & \mathsf{Cov}(i, c_1) \\ \mathsf{Cov}(j, i) & \mathsf{Var}(j) & \mathsf{Cov}(j, c_1) \\ \mathsf{Cov}(c_1, i) & \mathsf{Cov}(c_1, j) & \mathsf{Var}(c_1) \end{bmatrix}$$

– **Step 4: Compute Precision Matrix (Θ)** The precision matrix is obtained by inverting the covariance matrix:

$$\Theta = \Sigma^{-1}, \quad \Theta \in \mathbb{R}^{n' \times n'}$$

The key interpretation is:

∗ If $\Theta_{ij} = 0$, then nodes i and j are conditionally independent given all other nodes.

∗ The magnitude of Θ_{ij} reflects the strength of conditional dependency between i and j after removing all indirect (confounding) effects.

Conceptually, Θ acts like a "whitened" version of Σ, where all indirect influences (through confounders) have been algebraically canceled out.

- **Step 5: Compute Partial Correlation (f_{CI})** Once we have Θ, the partial correlation between nodes i and j controlling for all other variables in V' is computed as:

$$f_{CI}(i,j) = \rho(i,j \mid V' \setminus \{i,j\}) = -\frac{\Theta_{ij}}{\sqrt{\Theta_{ii}\,\Theta_{jj}}}$$

  The negative sign is conventional in partial correlation estimation when derived from the precision matrix. This produces a single scalar value in the range $[-1.0, 1.0]$.

- **Interpretation:**
  * $f_{CI}(i,j) \approx 0.0$: The link between i and j vanishes after controlling for confounders — implying that it was likely a spurious or indirect association.
  * $|f_{CI}(i,j)|$ close to 1.0: A strong conditional dependence remains, suggesting that i and j share a genuine, direct causal influence.
  * The sign of f_{CI} indicates the direction of the dependency (positive or inverse correlation). Positive partial correlation means i and j move together after conditioning on others; negative means they move oppositely.

- **Summary of Intuition:**
  * The covariance matrix Σ captures both direct and indirect associations.
  * The precision matrix $\Theta = \Sigma^{-1}$ isolates only the direct conditional dependencies.
  * The normalized entry $-\Theta_{ij}/\sqrt{\Theta_{ii}\Theta_{jj}}$ gives the cleanest estimate of the residual connection strength between i and j, after "subtracting out" all confounding paths.

- **The LLM Check ($f_{llmconfounder}$)** This looks for the confounder but in a semantic way. It finds semantic coincidences (Whereas our statistician found structural coincidences). It looks at the meaning of the words in the text snippets we give it. It uses its "world knowledge" to answer a "common sense" question. In short we need $f_{llmconfounder}$ despite the presence of f_{CI} (statistical check) because f_{CI} happens to be "semantically blind" and only finds structural confounders, whereas $f_{llmconfounder}$ uses textual context to perform a "smart" semantic check, catching non-causal links that our GNN may miss.
  We run a confidence-weighted CoT process.
  Process:

  Context Retrieval; We use our RAG-HyDE-RAV pipeline to retrieve k=5 snippets (contextstring) relevant to i,j and the names of the nodes in $M_{\text{setconfoundfinal}}$.
  CoT Prompt: Instead of asking the LLM for just a score, we force it to "show its work" first. We change the prompt to a 2-step task. So we are forrcing the LLM to perform a scientific intervention.

```
Context: "context_string"
Link to Check: '{node_i_name}' -> '{node_j_name}'
Potential Confounders: [ '{k1_name}', '{k2_name}', ... ]

1. Reasoning: First, provide a step-by-step rationale. If node i were NOT
↪  present, BUT all 'Potential Confounders' were HELD CONSTANT,
   what is the remaining, direct impact on node j? Explain why based only
   ↪  on the context.
   2. Score: Based on your reasoning, provide a final score from 0.0 (no
   ↪  impact) to 1.0 (strong impact).

   Output JSON: {"reasoning": "...", "score": 0.0}
```

Semantic Entropy (Sampling for Confidence): Aim here is to get a robust "confidence-weighted" score, not only s ingle noisy guess. We run our exact same CoT prompt N times (say N=5), with LLM's creativity aka temperature=0.7
We get a scores list

$$\text{Scoreslist} = [\, s_1, s_2, s_3, s_4, s_5 \,]$$

(Lower the spread/variance → higher the confidence; higher the variance → lower the confidence.)

$$\text{Reasoningslist} = [\, r_1, r_2, r_3, r_4, r_5 \,]$$

(We store them maybe not sure for provenance/auditability).
Score Aggregation and Confidence Computation:
Let each model in the ensemble provide a score. From these, we compute two key quantities:

$$\mu_{\text{score}} = \text{mean}(\text{Scoreslist})$$
$$C_{\text{confidence}} = 1.0 - \text{normalizedvariance}(\text{Scoreslist})$$

Here, μ_{score} represents the average prediction across ensemble members, and $C_{\text{confidence}}$ acts as an entropy-based reliability weight that decreases when variance across models is high.

Final Combined Score:

$$S_{\text{final}} = \mu_{\text{score}} \times C_{\text{confidence}}$$

This formulation ensures that even if the average score is high, high disagreement (variance) among ensemble members will reduce confidence and penalize the final score.

Illustrative Cases:

– **Case A (Confident "YES"):**
$\mu_{\text{score}} = \text{avg}(0.9, 0.95, \dots) = 0.92$
$C_{\text{confidence}} = 1.0 - \text{(low variance)} = 0.98$
Final Score: $0.92 \times 0.98 = \mathbf{0.90}$ — a high, trustworthy score.

– **Case B (Unsure / Weak):**
$\mu_{\text{score}} = \text{avg}(0.1, 0.7, \dots) = 0.34$
$C_{\text{confidence}} = 1.0 - \text{(high variance)} = 0.20$
Final Score: $0.34 \times 0.20 = \mathbf{0.068}$ — a low, "punished" score.

We combine these scores as:

Mean Causal Score: This is our primary semantic signal for the strength of the link.

$$\mu_{\text{LLMconfounder}}(i,j) = \frac{1}{N_s}\sum_{p=1}^{N_s} s_p$$

Variance Score: This is our "uncertainty" metric for the score. Low variance = high confidence. High variance = low confidence (the LLM is "unsure" and giving different scores).

$$\sigma^2_{\text{LLMconfounder}}(i,j) = \frac{1}{N_s}\sum_{p=1}^{N_s}(s_p - \mu_{\text{LLMconfounder}})^2$$

Semantic Coherence (R_{coh}): It measures the stability of the textual explanation (the Chain-of-Thought).

$$R_{\text{coh}}(i,j) = \frac{2}{N_s(N_s-1)}\sum_{p<q}\cos(v_p, v_q)$$

High Coherence means the LLM produced the same reasoning every time. Low coherence means the LLM produced contradictory reasoning every time, even if the final score was the same. (Bad! This is a "red flag"!)

iv. **Calibrated Learned Fusion (W_{direct}):**
Now we have to fuse all these signals to get one direct-guess score.
We vectorise and feed these 3 scores directly into our f_{fusion} (Calibrated Learned Fusion - See this in the EM refinement phase) model.

- **A feature vector using above scores:**

$$v_{ij} = \Big[\underbrace{\mu_{\text{GNN}},\, \sigma^2_{\text{GNN}},}_{\text{GNN Scores}}\ \underbrace{\mu_{\text{LLM}},\, \sigma^2_{\text{LLM}},\, \mu_{\text{LLMconfounder}},\, \sigma^2_{\text{LLMconfounder}},}_{\text{LLM Scores}}\ \underbrace{f_{CI}}_{\text{Stat Check}},\ \underbrace{R_{\text{cohllm}},\, R_{\text{cohcond}}}_{\text{Reasoning Confidence}}\Big]$$

(Note: We compute R_{coh} for both the naive f_{llm} and the $f_{llmcond}$ checks, giving us two reasoning confidence scores.)

- **Predicting a final score:**

$$W_{\text{direct}}(i,j) = f_{\text{fusion}}(v_{ij})$$

Our f_{fusion} model (trained in **EM refinement phase**) learns the crucial *"red flag"* rule: If Rcoh is low, then the score should be 0.0, even if μ_{LLM} is high.

Finally: A sparse $N \times N$ map: W_{direct}. This map is now already pruned of confounders because the f_{fusion} model has learned the rule as discussed above.

(b) **Find Best Indirect Paths I:** Above we calculated the direct link scores from node i to node j. Now we need to see for indirect paths from node i to node j. So here we are looking for **Mediation**.
Like between node i and node j we may have multiple indirect paths, we aggregate them in this method.
So $I(i,j) =$ Aggregated Indirect Evidence Score.
Ex: If $I(i,j)$ is much higher than $W_{direct}(i,j)$, then it means the direct link is a "false shortcut" and the relation between node i and node j is better explained by the indirect/detour paths. The objective of this step is to calculate $I(i,j)$, a single, robust score representing the total aggregated strength of all indirect ("detour") paths from node i to node j.

- **Path Enumeration and Encoding.**
  The goal of this component is to, for every candidate pair (i,j) in the "to-do" set E_{prior}, identify a set of credible indirect ("detour") paths from i to j and encode each such path into a fixed-dimensional vector $\vec{h}_p$ that can be consumed by the downstream Learned Path Aggregation (LPA) module.

  Induced Mini-Graph Construction and Log-Space Costs. Running path-finding on the full document graph with $N \approx 50{,}000$ nodes is computationally prohibitive. Instead, we construct a smaller induced subgraph $G' = (N_p, E', W')$ that focuses computation on relevant nodes and candidate mediators.

  First, we identify a small set of "mediator" or hub nodes N_{med} from the full co-occurrence graph $A_{\text{co-occur}}$, for example the top K nodes by PageRank or degree centrality. We then define the node set of the mini-graph as

$$N_p = \left(\bigcup_{(i,j)\in E_{\text{prior}}} \{i,j\} \right) \cup N_{\text{med}}.$$

  The resulting size $N'_p = |N_p|$ is much smaller than N (e.g., $N'_p \approx 10{,}000 + K$).
  The edge set E' consists of all directed pairs (u,v) such that $u,v \in N_p$ and there is a corresponding entry in the confounder-pruned direct map W_{direct} (from Step 3.1). The edge weights are inherited directly:

$$W'(u,v) = W_{\text{direct}}(u,v), \quad (u,v) \in E'.$$

  To enable numerically stable path scoring, we convert these probabilities into additive costs:

$$c(u,v) = -\log\big(W'(u,v) + \epsilon\big),$$

  where ϵ is a small positive constant (e.g., 10^{-9}) to avoid $\log(0)$. A high-confidence edge with $W'(u,v) \approx 0.9$ has a low cost, while a weak edge with $W'(u,v) \approx 0.1$ has a high cost. This transformation turns the problem of finding the strongest path (maximising the product of scores) into the equivalent problem of finding the cheapest path (minimising the sum of costs).

  Multi-Source Bounded k-Shortest Path Search. We seek, for each pair $(i,j) \in E_{\text{prior}}$, the top-$m$ most credible indirect paths from i to j, with path length bounded by a maximum

$L_{\max}$. To make this scalable, we adopt a multi-source dynamic programming approach over G'.

For each source node $i \in N_p$, we run a bounded k-shortest-path dynamic program, denoted

$$\text{BoundedkShortestPathDP}(G', \text{source} = i, K = m, L = L_{\max}),$$

which, for every target node j, maintains a list of up to m distinct paths from i to j with the smallest total cost and prunes any path whose length exceeds $L_{\max}$. This routine can be implemented as a modification of Dijkstra- or Bellman–Ford-style algorithms that keep multiple best paths per node.

The hyperparameters are:

- m: the number of top paths to retain per pair (i, j) (e.g., $m = 20$);
- $L_{\max}$: the maximum allowed path length (e.g., $L_{\max} = 5$), enforcing a bias toward short, causally plausible chains and ensuring fast termination.

The output of the multi-source DP is a lookup structure (PathTable) such that for every source i and target j we can retrieve the set of top-m paths found by the algorithm. For each candidate pair $(i, j) \in E_{\text{prior}}$, we then obtain its indirect paths via a constant-time lookup:

$$P_{\text{indirect}}(i, j) = \{p \in \text{PathTable}[i][j] \mid 2 \le |p| \le L_{\max}\},$$

where each path $p_l \in P_{\text{indirect}}(i, j)$ is a sequence of node IDs, e.g., $p_l = [i, k_1, \ldots, k_{L-1}, j]$.

Path Encoding via Transformer$_{\text{path}}$. Each path p_l is a variable-length sequence of nodes; we now encode it into a fixed-dimensional "path embedding" $\vec{h}_{p_l}$ using a pre-trained Transformer$_{\text{path}}$ model.

Let $p_l = [n_1, n_2, \ldots, n_L]$ be a path from source to target. We first obtain node embeddings from the GAE model trained in Phase 2:

$$Z \in \mathbb{R}^{N \times d_z}, \qquad Z_{n_1}, Z_{n_2}, \ldots, Z_{n_L} \in \mathbb{R}^{d_z}.$$

We then add positional encodings to inject order information. For each position $t \in \{1, \ldots, L\}$, let $PE_t \in \mathbb{R}^{d_z}$ be a standard sinusoidal positional embedding. The input sequence to the Transformer is

$$X_p = [Z_{n_1} + PE_1,\ Z_{n_2} + PE_2,\ \ldots,\ Z_{n_L} + PE_L] \in \mathbb{R}^{L \times d_z}.$$

The path Transformer processes this sequence and outputs a fixed-size representation:

$$\vec{h}_{p_l} = \mathrm{Transformer}_{\text{path}}(X_p) \in \mathbb{R}^{d_{\text{path}}},$$

where $\vec{h}_{p_l}$ may correspond, for example, to a dedicated `[CLS]` token or to a pooled representation over the output states.

To avoid recomputing embeddings for identical paths, we maintain a global cache keyed by the path signature (the ordered list of node IDs). Let sig(p_l) denote a canonical representation of the path. If sig(p_l) has already been encoded, we reuse the stored embedding; otherwise, we compute $\vec{h}_{p_l}$ and insert it into the cache:

$$\vec{h}_{p_l} = \begin{cases} \text{PathEmbeddingCache}[\text{sig}(p_l)], & \text{if sig}(p_l) \text{ is cached,} \\ \mathrm{Transformer}_{\text{path}}(X_p), & \text{otherwise (and then cached).} \end{cases}$$

For each pair (i, j), the final output of this stage is a set of at most m path embeddings

$$H_p(i, j) = \{\vec{h}_{p_1}, \ldots, \vec{h}_{p_m}\} \subset \mathbb{R}^{d_{\text{path}}},$$

where each $\vec{h}_{p_l}$ is a compact, semantically and structurally informed representation of an indirect path from i to j.

- **Learned Path Aggregation (LPA Model).** The objective of this component is to aggregate the set of indirect paths $P_{\text{indirect}}(i, j) = \{p_1, \ldots, p_m\}$ between a pair of nodes (i, j) into a single, robust, probabilistic score $I(i, j)$. This score is intended to capture the total strength of all indirect ("detour") mechanisms from i to j, by jointly considering both the structural strength and the semantic content of each path.

  The aggregation is performed by a pre-trained, pair-aware neural attention model f_{agg} that operates on:
  - the list of paths $P_{\text{indirect}}(i, j)$, where each p_l is a sequence of node IDs $[i, k_1, \ldots, j]$;
  - the corresponding path embeddings $H_p(i, j) = \{\vec{h}_{p_1}, \ldots, \vec{h}_{p_m}\}$, with each $\vec{h}_{p_l} \in \mathbb{R}^{d_{\text{path}}}$ obtained from the Transformer$_{\text{path}}$;
  - the node embeddings Z_i, Z_j from the GAE matrix $Z \in \mathbb{R}^{N \times d_z}$;
  - the edge cost matrix $c(u, v)$ defined on the induced mini-graph G'.

  The final score $I(i, j)$ is computed as an attention-weighted sum of per-path strength features f_{p_l}:

  $$I(i, j) = \sum_{l=1}^{m} a_l \, f_{p_l},$$

  where $f_{p_l} \in [0, 1]$ measures the numerical strength of path p_l, and $a_l \in [0, 1]$ is an attention weight reflecting the semantic importance of p_l for the specific pair (i, j).

  Per-Path Strength Features. For each path $p_l \in P_{\text{indirect}}(i, j)$ we compute a scalar strength $f_{p_l} \in [0, 1]$ in several steps.

  First, we define the total log-cost of the path p_l as the sum of its edge costs:

  $$c(p_l) = \sum_{(u,v) \in p_l} c(u, v),$$

  where $c(u, v) = -\log(W_{\text{direct}}(u, v) + \epsilon)$ is the edge cost from Step 3.2.1 and ϵ is a small constant.

  Next, we apply a learnable hop penalty that depends on the path length. Let $|p_l|$ denote the length (number of nodes) in p_l, and let $g(\cdot)$ be a small pre-trained MLP. We define

  $$\gamma(|p_l|) = g(|p_l|),$$

  so that the model can learn a non-linear penalty as a function of path length (e.g., two-hop paths may be preferred, while longer paths are penalized more strongly).

  We then combine these quantities into a log-strength:

  $$\text{logstr}(p_l) = -\big(c(p_l) + \gamma(|p_l|)\big),$$

  and finally squash this value into $[0, 1]$ using the sigmoid function:

  $$f_{p_l} = \sigma(\text{logstr}(p_l)) = \frac{1}{1 + e^{-\text{logstr}(p_l)}}.$$

  This produces a probability-like path strength that increases with higher edge probabilities and shorter, more plausible paths.

Pair-Aware Attention Weights. In parallel, we compute an attention weight a_l for each path p_l, based on its semantic embedding $\vec{h}_{p_l}$ and the specific node pair (i, j).

We first construct a pair context vector $\vec{q}_{ij}$ that captures the semantic context of the candidate link from i to j. Using the GAE embeddings $Z_i, Z_j \in \mathbb{R}^{d_z}$, we define:

$$\vec{q}_{ij} = \mathrm{MLP}_{\textsf{pair}}([Z_i; Z_j]),$$

where $[Z_i; Z_j] \in \mathbb{R}^{2d_z}$ denotes concatenation and $\mathrm{MLP}_{\textsf{pair}}$ is a pre-trained MLP. The resulting $\vec{q}_{ij} \in \mathbb{R}^{d_{\textsf{path}}}$ lies in the same space as the path embeddings.

For each path p_l, we concatenate $\vec{q}_{ij}$ and $\vec{h}_{p_l}$ and compute a scalar compatibility score:

$$\textsf{score}(p_l) = \mathrm{LeakyReLU}\big(\vec{w}^\top [\vec{q}_{ij}; \vec{h}_{p_l}]\big),$$

where $\vec{w} \in \mathbb{R}^{2d_{\textsf{path}}}$ is a learned weight vector and $[\vec{q}_{ij}; \vec{h}_{p_l}] \in \mathbb{R}^{2d_{\textsf{path}}}$ is their concatenation. This score reflects how well the semantic content of path p_l aligns with the specific pair (i, j).

We normalize these scores across the m paths using a softmax:

$$a_l = \frac{\exp(\textsf{score}(p_l))}{\sum_{k=1}^{m} \exp(\textsf{score}(p_k))},$$

so that $a_l \in [0, 1]$ and $\sum_{l=1}^{m} a_l = 1$. Paths that are more semantically relevant to the (i, j) context receive larger weights.

Aggregation. The final indirect score for the pair (i, j) is computed as the attention-weighted sum of the per-path strengths:

$$I(i, j) = \sum_{l=1}^{m} a_l \, f_{p_l}.$$

In this formulation, paths that are both structurally strong (high f_{p_l}) and semantically aligned with the pair context (high a_l) dominate the final score, while structurally strong but semantically irrelevant paths, or weak but relevant paths, contribute less.

Uncertainty Quantification. To obtain not only a point estimate but also a measure of uncertainty for the indirect score, we apply a simple bootstrap procedure. For each pair (i, j), we perform B bootstrap rounds (e.g., $B = 5$):

i. In round $b \in \{1, \ldots, B\}$, form a bootstrapped path set $P^{(b)}_{\textsf{indirect}}(i, j)$ by sampling m paths with replacement from $P_{\textsf{indirect}}(i, j)$.
ii. Run the full LPA pipeline described above (per-path strength and pair-aware attention) on $P^{(b)}_{\textsf{indirect}}(i, j)$ to obtain a bootstrap estimate $I_b(i, j)$.

This yields a collection of scores $\{I_1(i, j), \ldots, I_B(i, j)\}$. We then compute the mean and variance:

$$\mu_I(i, j) = \frac{1}{B} \sum_{b=1}^{B} I_b(i, j), \qquad \sigma_I^2(i, j) = \frac{1}{B-1} \sum_{b=1}^{B} \big(I_b(i, j) - \mu_I(i, j)\big)^2.$$

For each candidate pair $(i, j) \in E_{\textsf{prior}}$, this step therefore outputs two scalars:

  - $\mu_I(i,j)$: the mean aggregated strength of all indirect paths;
  - $\sigma_I^2(i,j)$: the estimated uncertainty (variance) of this indirect strength.

These quantities are then used in the subsequent pruning step (Step 3.3) to compare the direct evidence P_{direct} against an uncertainty-aware upper confidence bound for the indirect evidence.

- **Final Pruning (Build C_{prior})**

The goal of this final step is to decide whether each candidate link (i,j) in the set E_{prior} should be retained or pruned. Each candidate undergoes two sequential checks: a statistical significance test based on FDR control, and a mediation-aware causal pruning test. The output is the sparse, statistically rigorous "answer key" C_{prior}, containing only genuine direct causal links together with their full provenance.

Inputs.

  - E_{prior}: The candidate set of pairs (i,j).
  - $P_{\text{direct}}(i,j)$: Direct Evidence scores from Step 3.1.
  - $\mu_I(i,j)$, $\sigma_I^2(i,j)$: Mean and variance of indirect evidence from Step 3.2.
  - $\{D_{\text{null}}^{(b)}\}$: A stratified collection of null distributions (degree-based buckets).
  - Calibrator_D, Calibrator_I: Pre-trained calibration models.
  - Hyperparameters: α (FDR target), δ (tolerance), $\epsilon_{\min}$ (minimal effect size), z_α (confidence z-score), τ_{uncert} (uncertainty cap), λ_{med} (mediation penalty).

Joint Calibration

To enable a meaningful comparison between direct and indirect scores, both quantities are placed on a common calibrated probability scale. This is achieved through isotonic regression or temperature-scaling models trained offline on a held-out validation set.

$$P_{\text{direct}}^{\text{calib}}(i,j) = \text{Calibrator}_D(P_{\text{direct}}(i,j)), \qquad \mu_I^{\text{calib}}(i,j) = \text{Calibrator}_I(\mu_I(i,j)).$$

For clarity, the calibrated values will again be denoted by $P_{\text{direct}}(i,j)$ and $P_{\text{indirect}}(i,j)$ in subsequent steps.

Statistical Significance via FDR Control

To determine which links have statistically meaningful direct evidence, each candidate is compared against the null distribution corresponding to its structural bucket. For each (i,j), we identify its bucket b, then compute its empirical p-value:

$$p(i,j) = \frac{\left|\left\{ s \in D_{\text{null}}^{(b)} \;\middle|\; s \geq P_{\text{direct}}(i,j) \right\}\right|}{|D_{\text{null}}^{(b)}|}.$$

All K_2 p-values are sorted to obtain $p_{(1)} \leq p_{(2)} \leq \cdots \leq p_{(K_2)}$. We then find the largest index k satisfying

$$p_{(k)} \leq \frac{k}{K_2}\,\alpha.$$

The resulting Benjamini–Hochberg threshold is defined as

$$\alpha_{\text{BH}} = p_{(k)}.$$

Only candidates with $p(i,j) < \alpha_{\text{BH}}$ proceed to the next stage.

Pruning and Construction of C_{prior}
We initialize an empty list `Cpriorlist = [ ]`. Each candidate (i,j) is then processed as follows.

i. *Statistical Gate.* We require both statistical significance and a minimal effect size:

$$p(i,j) < \alpha_{\text{BH}} \qquad \text{and} \qquad P_{\text{direct}}(i,j) \geq \epsilon_{\min}.$$

Pairs that fail either condition are pruned.

ii. *Mediation Gate.* For surviving candidates, we compare the direct score against a robust upper confidence bound on the indirect score. The uncertainty is capped to prevent excessively large penalties:

$$P_{\text{indirect}}^{\text{UCB}} = P_{\text{indirect}}(i,j) + z_\alpha \cdot \min\Big(\sqrt{\sigma_I^2(i,j)},\ \tau_{\text{uncert}}\Big).$$

The direct link is accepted if

$$P_{\text{direct}}(i,j) > \left(P_{\text{indirect}}^{\text{UCB}} - \delta\right),$$

indicating that the direct explanation is stronger than any plausible indirect alternative. When this condition holds, we compute a mediation-aware residual score:

$$\text{finalscore} = \max\big(0,\ P_{\text{direct}}(i,j) - \lambda_{\text{med}} \cdot P_{\text{indirect}}^{\text{UCB}}\big).$$

Otherwise the candidate is pruned by setting finalscore $= 0$.

For each processed pair (i,j), we retrieve the associated `RAGcontext`, `CoTreasoning`, and the top three indirect paths from earlier stages, and append the record

$$(i, j,\ \text{finalscore},\ \texttt{RAGcontext},\ \texttt{CoTreasoning},\ \texttt{Topindirectpaths})$$

to `Cpriorlist`.

Output. The completed list C_{prior} contains all surviving direct causal links, each of which is:

– statistically significant relative to an appropriate stratified null,
– of meaningful effect size,
– free of confounding,
– not explainable by indirect mechanisms,
– evaluated with uncertainty-aware scoring,
– and fully auditable through stored contextual evidence.

This list serves as the final, high-fidelity answer key for training the main CausGT-HS model.

2. **EM-Refinement:**
This phase is a one-time, offline, self-supervised training loop that learns all specialized neural components used by the online CoCaD pipeline. It proceeds in three main steps: synthetic "gold data" generation, bootstrap pre-training of all modules, and an EM-style refinement loop on real documents with a Mean-Teacher architecture.

Learnable components in this phase.

- Sentence-level embedder f_{embed} with parameters θ_{embed}.
- Contextual plausibility classifier CPC_{model} (DeBERTa backbone + 3 heads) with parameters θ_{cpc}.
- Fusion model f_{fusion} (Wide&Deep MLP) with parameters θ_{fusion}.
- Path encoder Transformer$_{\text{path}}$ and path aggregator f_{agg} (jointly LPA_{model}) with parameters θ_{lpa}.
- Optional LLM-distillation model $f_{\text{LLM-student}}$ with parameters θ_{llm} used to approximate slow LLM scores in Phase 3.

All of these are neural and trained by gradient-based optimization on self-supervised objectives described below. In addition, a gradient-boosted tree model g_{GBT} is trained once on synthetic data and then distilled into f_{fusion}.

(a) **Synthetic Corpus & Ground-Truth Dataset Generation**

The objective of this stage is to construct a large-scale, richly annotated synthetic dataset D_{synth} that serves as the pseudo–ground truth for all downstream model pre-training. Because no human annotations are available in our setting, the entire dataset must be generated programmatically. This process consists of three tightly coupled stages: (i) sampling a ground-truth causal graph G_{true}, (ii) generating a synthetic document corpus that textually describes the structures contained in G_{true}, and (iii) executing the full inference pipeline on these documents to produce training files with exact labels drawn directly from G_{true}.

(1) Generation of the Ground-Truth Causal Graph G_{true}.
We construct a directed acyclic graph (DAG) $G_{\text{true}} = (V, E_{\text{true}})$ at large scale (e.g., $|V| \approx 10^5$), ensuring diversity in both structure and local causal mechanisms. The generative procedure consists of:

- *Topology Sampling.* The vertex set V is divided into several structural regimes and we sample subgraphs from a mixture of graph families, including scale-free networks, small-world graphs, and community-structured stochastic block models. The overall topology is obtained by combining these subnetworks and pruning edges to enforce acyclicity.
- *Structural Causal Mechanisms.* For each node $v_i \in V$, we define a functional assignment:

$$v_i = f_i(PA(v_i), \eta_i),$$

where $PA(v_i)$ denotes the parent set of v_i, and η_i is an exogenous noise term. The mechanism f_i is drawn from a mixture of families:

$$f_i^{\text{lin}}(PA(v_i), \eta_i) = \sum_{j \in PA(v_i)} c_{ij} v_j + \eta_i, \qquad f_i^{\text{mlp}}(PA(v_i), \eta_i) = \text{MLP}(v_{j_1}, \ldots, v_{j_k}) + \eta_i,$$

as well as saturating nonlinear forms $f_i^{\text{sat}} = \sigma(\sum_j c_{ij} v_j) + \eta_i$. Noise may be homoscedastic or heteroscedastic, e.g., $\eta_i \sim \mathcal{N}(0, \sigma_i^2(PA(v_i)))$.
- *Causal Motif Planting.* To guarantee broad coverage, we explicitly embed canonical motifs and record a motif tag for each: direct edges $i \to j$, mediated structures $i \to k \to j$, confounders $i \leftarrow k \to j$, colliders $i \to k \leftarrow j$, and hybrid cases combining direct and indirect paths.

(2) Generation of the Synthetic Text Corpus D_{corpus}.
We employ a text generator model f_{hyde} to produce document snippets describing the relationships in G_{true}. To mimic real-world corpora, the text is diversified across domains and enriched with realistic noise.

- *Domain-Conditioned Prompts.* Each node inherits a domain label (e.g., biomedical, finance). Prompts given to the generator are conditioned on this label, producing domain-appropriate narrative structures. For instance, for a mediated edge $i \to k \to j$ in a biomedical context:

  ``Write a clinical note explaining that i affects j by altering k."

- *Contrastive Snippets.* For every true relation, we additionally generate a contrastive (false) snippet that reverses or perturbs the causal direction. These serve as hard negatives in later training.
- *Generator–Critic Filtering.* A discriminator model evaluates whether a generated snippet faithfully reflects its intended causal structure. If the score falls below a threshold, the snippet is regenerated.
- *Structured Noise Augmentation.* To reduce the simulation–to–real gap, each snippet undergoes realistic perturbations such as hedging phrases, minor typographical errors, and style transfers across linguistic registers.

(3) Generation of the Fully Labeled Dataset D_{synth}.
After constructing D_{corpus}, we execute the full inference pipeline on it. Because the ground truth G_{true} is known exactly, all labels can be assigned without ambiguity.

- *Feature Extraction.* For each candidate pair (i, j) processed by the pipeline, we extract the multi-source feature vector v_{ij}, path embeddings $H_p(i, j)$, and path-level features obtained in Phases 1–3.
- *Ground-Truth Labeling.* Each pair (i, j) is matched to its true causal status by querying G_{true}. For example:
  $$\text{isdirectcause}(i, j) = \mathbf{1}\{(i, j) \in E_{\text{true}}\},$$
  and for indirect effects, we compute a normalized quantitative score by simulating the SCM under interventions and aggregating the path-specific contribution of i to j:
  $$\text{trueindirectscore}(i, j) = \frac{\partial\, \mathbb{E}[j \mid do(i)]}{\partial i} \text{ projected to } [0, 1].$$
- *Controlled Label Smoothing.* To avoid brittle overfitting to perfectly clean labels, the final training label is a smoothed version:
  $$y'_{ij} = y^{\text{gold}}_{ij}(1 - \epsilon) + \frac{\epsilon}{2},$$
  where ϵ is a small constant.
- *Hard Negative Sampling.* For every positive pair (i, j), we additionally select a structurally incorrect but semantically similar negative pair (i, k) such that $\text{sim}(Z_j, Z_k)$ is high but $(i, k) \notin E_{\text{true}}$.

The final output is the multi-file dataset D_{synth}, organized into several training sets used throughout Phase 4:

- `TrainingSetCPC`: contextual text with labels for plausibility, temporality, and mechanistic support.
- `TrainingSetFusion`: feature vectors v_{ij} paired with direct-causation labels.
- `TrainingSetLPA`: path embeddings and quantitative indirect-effect scores.
- `TrainingSetPath`: anchor, positive, and negative path tuples for contrastive training.
- `TrainingSetEmbedder`: query–positive–negative triplets for training the retrieval embedder.

(b) **Pre-train Models (Bootstrap)**

The objective of this step is to perform the initial "bootstrap" training for all specialized, learnable components required by the CoCaD inference pipeline. All models are trained on the synthetic, richly labeled datasets produced in Step 4.1 (e.g., `TrainingSetCPC`, `TrainingSetFusion`, `TrainingSetLPA`, `TrainingSetPath`, `TrainingSetEmbedder`). The resulting models form the initial versions $f^{(0)}_{\text{fusion}}$, $LPA^{(0)}_{\text{model}}$, $CPC^{(0)}_{\text{model}}$, and $f^{(0)}_{\text{embed-sota}}$, which are later refined in the EM-Refinement loop (Step 4.3).

Global training regime. All models in this step are trained under a shared curriculum and batching strategy:

- *Curriculum over regimes:* Training is staged using the regime metadata (`regimeid`) attached to each synthetic example in D_{synth}.
  - Stage A: train only on "easy" regimes (low in-degree, no hidden confounders).
  - Stage B: include "moderate" regimes (mediators, observed confounders).
  - Stage C: include "hard" regimes (latent confounders, high in-degree).
- *Domain-balanced batching:* Each mini-batch is stratified by the domain label (`domainlabel`, e.g., finance, biomed), so that all domains are represented during training.
- *Unified early stopping:* For each model, early stopping is based on performance on a held-out synthetic validation set that is stratified by regime and domain. Training stops when metrics on the hardest regimes saturate.

Within this regime, the following components are trained:

i. **Train $f_{\text{embed-sota}}$ (causality-aware embedder)**

Learnable components: A sentence-transformer backbone (e.g., `all-MiniLM-L6-v2`) and a small projection head $g(\cdot)$.

Model. For an input text u (query or snippet), the backbone produces a representation $h \in \mathbb{R}^{d_{\text{backbone}}}$, and a two-layer MLP projection head

$$g(h) = W_2\,\sigma(W_1 h) \in \mathbb{R}^{d_{\text{embed}}},$$

is applied, where W_1, W_2 are learnable weight matrices and σ is a non-linearity (e.g., GELU). The final embedding is L2-normalized:

$$\vec{z} = \frac{g(h)}{\|g(h)\|_2} \in \mathbb{R}^{d_{\text{embed}}}.$$

Data. `TrainingSetEmbedder` contains tuples

$$(q_{\text{query}}, \{s_{\text{pos}}\}, \{s_{\text{neg}}\}, \{s_{\text{hard-neg}}\}),$$

with multiple positive snippets, in-batch negatives, and additional hard negatives.

Objective. For each query q, we construct multiple stochastic views (e.g., via dropout/-paraphrase) of q and its positives. Let $\vec{z}_q$ be the embedding of a query view, $\mathcal{P}(q)$ the set of embeddings of all positive views, and $\mathcal{N}(q)$ the set of negative snippet embeddings in the batch. With cosine similarity

$$\text{sim}(\vec{a}, \vec{b}) = \frac{\vec{a}^\top \vec{b}}{\|\vec{a}\|_2 \|\vec{b}\|_2},$$

and a temperature parameter $\tau > 0$, the supervised contrastive loss for q is

$$\mathcal{L}_{\text{sup-contr}}(q) = -\frac{1}{|\mathcal{P}(q)|} \sum_{p \in \mathcal{P}(q)} \log \frac{\exp(\text{sim}(\vec{z}_q, \vec{z}_p)/\tau)}{\sum_{z \in \mathcal{P}(q) \cup \mathcal{N}(q)} \exp(\text{sim}(\vec{z}_q, \vec{z})/\tau)}.$$

The total loss is the average of $\mathcal{L}_{\text{sup-contr}}(q)$ over queries in the batch. Backpropagation updates both the backbone and projection head parameters.

Output. A causality-aware retrieval model $f^{(0)}_{\text{embed-sota}}$ that maps queries and evidence snippets to a shared embedding space.

ii. **Train CPC_{model} (semantic filter)**

Learnable components: A Transformer encoder (e.g., DeBERTa-v3), three classification heads, and an alignment projection head.

Model. Given an input sequence encoding a context and a candidate pair, the encoder produces a pooled representation $H_{\text{CLS}} \in \mathbb{R}^{d_{\text{enc}}}$. Three independent linear heads predict the plausibility, temporal, and mechanistic labels:

$$h_{\text{plaus}} = W_{\text{plaus}} H_{\text{CLS}} + b_{\text{plaus}}, \quad h_{\text{temp}} = W_{\text{temp}} H_{\text{CLS}} + b_{\text{temp}}, \quad h_{\text{mech}} = W_{\text{mech}} H_{\text{CLS}} + b_{\text{mech}}.$$

An additional projection head g_{align} maps H_{CLS} into the same embedding dimension as $f_{\text{embed-sota}}$:

$$\hat{e}_{\text{CPC}} = g_{\text{align}}(H_{\text{CLS}}) \in \mathbb{R}^{d_{\text{embed}}}.$$

Data. `TrainingSetCPC` contains tuples

$$(\text{context}, (i, j), y_{\text{plaus}}, y_{\text{temp}}, y_{\text{mech}}),$$

stratified by motif, regime, and domain.

Objective. For each task $t \in \{\text{plaus}, \text{temp}, \text{mech}\}$ we use label smoothing. Given a binary label $y_t \in \{0, 1\}$ and a smoothing parameter ϵ_{ls}, the smoothed label is

$$y'_t = y_t(1 - \epsilon_{\text{ls}}) + \frac{\epsilon_{\text{ls}}}{2}.$$

Let $p_t = \sigma(h_t)$ be the predicted probability for task t, where σ is the sigmoid. With Focal Loss parameters α_t and γ, the task loss is

$$\mathcal{L}_{\text{Focal}}(h_t, y'_t) = -\alpha_t (1 - p_t)^\gamma y'_t \log p_t - \alpha_t p_t^\gamma (1 - y'_t) \log(1 - p_t).$$

We use uncertainty-based weighting across tasks with learnable $\sigma_t > 0$:

$$\mathcal{L}_{\text{multi-task}} = \sum_{t \in \{\text{plaus,temp,mech}\}} \left(\frac{1}{2\sigma_t^2} \mathcal{L}_{\text{Focal}}(h_t, y'_t) + \log \sigma_t \right).$$

To align the semantic space with the retriever, we penalize the ℓ_2-distance between $\hat{e}_{\text{CPC}}$ and the embedding $e_{\text{embed-sota}}$ (obtained by encoding the same context with $f_{\text{embed-sota}}$):

$$\mathcal{L}_{\text{align}} = \lambda_{\text{align}} \left\| \hat{e}_{\text{CPC}} - e_{\text{embed-sota}} \right\|_2^2.$$

The total loss is

$$\mathcal{L}_{\text{CPC}} = \mathcal{L}_{\text{multi-task}} + \mathcal{L}_{\text{align}},$$

which updates the encoder, all three heads, the alignment head, and the task-uncertainty parameters.

Output. A pre-trained semantic filter $CPC^{(0)}_{\text{model}}$ that jointly predicts plausibility, temporal ordering, and mechanistic support.

iii. **Train f_{fusion} (fusion model)**

Learnable components: A gradient-boosted decision tree model and a scalar calibration function.

Model. For each candidate pair (i, j), the feature vector

$$v_{ij} = \left[\mu_{\text{GNN}}, \sigma^2_{\text{GNN}}, \mu_{\text{LLM}}, \sigma^2_{\text{LLM}}, \mu_{\text{LLM-cond}}, \sigma^2_{\text{LLM-cond}}, f_{\text{CI}}, S_{\text{GAE}}(i, j), k_{\text{disjoint}} \right] \in \mathbb{R}^{d_v}$$

is fed to a gradient-boosted tree model (e.g., LightGBM), which outputs a raw score $s_{ij} \in \mathbb{R}$ and a corresponding probability $p_{ij} = \sigma(s_{ij})$.

Data. `TrainingSetFusion` contains pairs (v_{ij}, y_{ij}), where $y_{ij} \in \{0, 1\}$ indicates whether (i, j) is a true direct cause in G_{true} (possibly with small label noise as defined in Step 4.1).

Objective. The primary classification loss is binary cross-entropy (LogLoss):

$$\mathcal{L}_{\text{LogLoss}} = -\sum_{(i,j)} \left[y_{ij} \log p_{ij} + (1 - y_{ij}) \log(1 - p_{ij}) \right].$$

In addition, we enforce ranking behavior within each source node i. For each i, we construct pairs of a positive (i, j^+) and a hard negative (i, j^-) and minimize a pairwise hinge loss:

$$\mathcal{L}_{\text{rank}} = \sum_i \sum_{(j^+, j^-)} \max\left(0,\, m - s_{ij^+} + s_{ij^-}\right),$$

where $m > 0$ is a margin hyperparameter. The total training objective is

$$\mathcal{L}_{\text{fusion}} = \mathcal{L}_{\text{LogLoss}} + \lambda_{\text{rank}} \mathcal{L}_{\text{rank}}.$$

Monotonic constraints. The boosted trees are trained with monotonicity constraints reflecting prior knowledge: scores should be monotonically increasing in features such as $\mu_{\text{LLM-cond}}$, f_{CI}, and $S_{\text{GAE}}(i, j)$, and monotonically decreasing in uncertainty features such as σ^2_{GNN} and σ^2_{LLM}.

Calibration. After training, we calibrate the predicted probabilities on a held-out validation set using isotonic regression. Given raw predictions p_{ij}^{raw} and true labels y_{ij}, the calibrator Calibrator_D fits a non-decreasing function

$$\hat{p}_{ij} = \text{Calibrator}_D\big(p_{ij}^{\text{raw}}\big),$$

which is stored for use in Phase 3.

Output. A robust, calibrated fusion model $f_{\text{fusion}}^{(0)}$ together with Calibrator_D.

iv. **Train LPA_{model} (learned path aggregator)**

Learnable components: A path encoder $Transformer_{\text{path}}$ and an attention-based aggregation network f_{agg}.

Model. For each path $p = [n_1, \dots, n_L]$, we construct a sequence of node embeddings plus positional encodings and pass it through $Transformer_{\text{path}}$ to obtain a path embedding $\vec{h}_p \in \mathbb{R}^{d_{\text{path}}}$. For a pair (i, j) with m paths $\{p_l\}_{l=1}^{m}$, the aggregator f_{agg} produces an indirect score $I_{\text{learned}}(i, j) \in [0, 1]$ based on path embeddings and path-level features (e.g., f_{p_l}).

Data. Two datasets are used:

- `TrainingSetPath`: triplets $(p_{\text{anchor}}, p_{\text{positive}}, p_{\text{hard-neg}})$ for contrastive pre-training of $Transformer_{\text{path}}$.
- `TrainingSetLPA`: tuples $(H_p(i,j), F_p(i,j), \text{trueindirectscore}_{ij})$ for training the aggregator and fine-tuning the encoder.

Two-phase schedule.

- Phase A: pre-train $Transformer_{\text{path}}$ on `TrainingSetPath` with contrastive and diversity losses.
- Phase B: first freeze $Transformer_{\text{path}}$ and train f_{agg} on `TrainingSetLPA` using a regression loss, then jointly fine-tune both components with the full objective.

Objective. The total loss is

$$\mathcal{L}_{\text{LPA}} = \lambda_{\text{path}}\mathcal{L}_{\text{path}} + \lambda_{\text{div}}\mathcal{L}_{\text{div}} + \lambda_{\text{regress}}\mathcal{L}_{\text{regress}}.$$

(i) Contrastive loss $\mathcal{L}_{path}$. Given an anchor path p_a, a positive path p_b for the same pair (i, j), and a set of negatives $\mathcal{N}(p_a)$, we compute normalized embeddings $\vec{h}_{p_a}, \vec{h}_{p_b}, \vec{h}_q$ and apply InfoNCE:

$$\mathcal{L}_{\text{path}} = -\log \frac{\exp(\text{sim}(\vec{h}_{p_a}, \vec{h}_{p_b})/\tau)}{\sum_{q \in \{p_b\} \cup \mathcal{N}(p_a)} \exp(\text{sim}(\vec{h}_{p_a}, \vec{h}_q)/\tau)}.$$

Negatives $\mathcal{N}(p_a)$ are drawn both from the current batch and from a momentum queue of past embeddings to enlarge the effective negative set.

(ii) Diversity loss $\mathcal{L}_{div}$. To prevent representational collapse, we employ variance–covariance regularization (VICReg) on a batch of path embeddings $H_p \in \mathbb{R}^{B \times d_{\text{path}}}$. Let $H_p^{(:,k)}$ denote the k-th feature dimension over the batch. The variance term enforces a minimum variance γ along each dimension:

$$\text{Var}(H_p) = \frac{1}{d_{\text{path}}} \sum_{k=1}^{d_{\text{path}}} \max\left(0, \gamma - \sqrt{\text{Var}(H_p^{(:,k)})}\right),$$

and the covariance term penalizes off-diagonal covariance entries:

$$\mathsf{Cov}(H_p) = \frac{1}{d_{\text{path}}^2} \sum_{k \neq \ell} \left(\Sigma_{k\ell}\right)^2,$$

where Σ is the empirical covariance matrix of H_p. The diversity loss is

$$\mathcal{L}_{\text{div}} = \lambda_{\text{vic}}\,\mathsf{Var}(H_p) + \mu_{\text{vic}}\,\mathsf{Cov}(H_p).$$

(iii) Regression loss $\mathcal{L}_{regress}$. For each pair (i, j) we have a ground-truth indirect score trueindirectscore$_{ij} \in [0, 1]$. Let

$$e_{ij} = I_{\text{learned}}(i, j) - \text{trueindirectscore}_{ij}.$$

We use the Huber loss with parameter $\delta > 0$:

$$\mathcal{L}_{\text{regress}}(i, j) = \begin{cases} \frac{1}{2}e_{ij}^2, & |e_{ij}| \leq \delta, \\ \delta|e_{ij}| - \frac{1}{2}\delta^2, & |e_{ij}| > \delta. \end{cases}$$

Averaging over all training pairs yields $\mathcal{L}_{\text{regress}}$.

Output. A learned path-aggregation model $LPA^{(0)}_{\text{model}}$ consisting of a path encoder and an attention-based aggregator capable of mapping sets of paths to calibrated indirect scores.

(c) **The EM-Refinement Loop**

Goal. The objective of this step is to refine the pre-trained (v0.1) models from Step 4.2 on the real, unlabeled document corpus by using their own predictions as a self-supervised signal. The refinement is formulated as an Expectation–Maximization (EM) style loop with a Mean-Teacher architecture to ensure stability.

We maintain two sets of parameters for each learnable component (e.g., fusion model, path aggregator):

- *Student parameters θ_S*: actively updated by gradient-based optimization.
- *Teacher parameters θ_T*: updated as an exponential moving average (EMA) of θ_S and used to generate pseudo-labels.

For concreteness, let $\theta_{S,\text{fusion}}$ and $\theta_{T,\text{fusion}}$ denote the parameters of the student and teacher fusion models, and analogously for the LPA model.

We perform R refinement rounds:

$$\text{for } r = 1, \ldots, R.$$

A. E-Step (Expectation / Pseudo-Label Estimation).

Goal. For a fixed round r, the E-step uses the current teacher models to compute the best available estimate of the causal answer key $C^{(r)}_{\text{prior}}$ on the real corpus.

Procedure.

i. We run the complete CoCaD pipeline (Phase 3: Steps 3.1–3.3) on the real document corpus, using the teacher parameter sets from the previous round, $\theta_T^{(r-1)}$, for all learnable modules (fusion model, LPA model, CPC model, retriever, etc.). This yields, for each candidate pair $(i,j) \in E_{\text{prior}}$, a final score and associated statistics:

$$(i,j) \;\mapsto\; \big(\text{finalscore}_{ij}^{(r)},\, p(i,j),\, v_{ij},\, H_p(i,j),\, \dots\big),$$

where $p(i,j)$ is the empirical p-value from Step 3.3 and v_{ij} is the feature vector used by the fusion model.

ii. In round $r=1$, the conditional LLM component in Step 3.1 is executed explicitly to obtain scores $\mu_{\text{LLMcond}}(v_{ij})$ for all (i,j), and we store the pairs

$$\mathcal{D}_{\text{distill}} = \{(v_{ij},\, \mu_{\text{LLMcond}}(v_{ij}))\}.$$

iii. In the M-step of round 1 (see below), we train a student surrogate model f_{student} to approximate the LLM scores. In subsequent rounds $r>1$, the expensive LLM calls in Step 3.1 are deterministically replaced by $f_{\text{student}}(v_{ij})$, so that the E-step remains computationally feasible while preserving the interface of the pipeline.

The output of the E-step at round r is the sparse answer key

$$C_{\text{prior}}^{(r)} = \{(i,j,\text{finalscore}_{ij}^{(r)}, p(i,j), \dots)\}.$$

B. M-Step (Maximization / Student Update).

Goal. For fixed pseudo-labels $C_{\text{prior}}^{(r)}$, the M-step updates the student parameters $\theta_S^{(r)}$ by minimizing a combination of pseudo-label losses and consistency regularization on real data.

Soft, Confidence-Weighted Pseudo-Labels. For each pair (i,j) in $C_{\text{prior}}^{(r)}$ we define:

$$\begin{aligned} y_{ij}^{\text{soft}} &= \text{finalscore}_{ij}^{(r)} \in [0,1], \\ w_{ij} &= 1 - p(i,j), \end{aligned}$$

where w_{ij} acts as a confidence weight: highly significant links (small $p(i,j)$) receive weights close to 1, whereas non-significant links receive weights close to 0.

We also define a hard binary label for pruning decisions,

$$y_{ij}^{\text{bin}} = \begin{cases} 1, & \text{if finalscore}_{ij}^{(r)} > 0, \\ 0, & \text{otherwise.} \end{cases}$$

B.1. Fusion Model Update.

The student fusion model $f_{\text{fusion,S}}^{(r)}$ with parameters $\theta_{S,\text{fusion}}^{(r)}$ maps feature vectors v_{ij} to scores in $(0,1)$. We update $\theta_{S,\text{fusion}}^{(r)}$ by minimizing a confidence-weighted binary cross-entropy loss:

$$\mathcal{L}_{\text{fusion-real}} = \sum_{(i,j)\in C_{\text{prior}}^{(r)}} w_{ij} \cdot \text{BCE}\big(f_{\text{fusion,S}}^{(r)}(v_{ij}),\, y_{ij}^{\text{soft}}\big),$$

where $\text{BCE}(p,y) = -\left(y \log p + (1-y)\log(1-p)\right)$.

Gradients of $\mathcal{L}_{\text{fusion-real}}$ with respect to $\theta^{(r)}_{S,\text{fusion}}$ are computed and applied using standard stochastic gradient descent or Adam.

B.2. LPA Model Update.

The student LPA model $LPA^{(r)}_{\text{S}}$, with parameters $\theta^{(r)}_{S,\text{LPA}}$, outputs for each pair (i,j) an aggregated indirect score $I^{(r)}_{\text{learned,S}}(i,j) \in (0,1)$. In the refinement loop we treat this as a classifier for pruning and update it using the binary pseudo-labels:

$$\mathcal{L}_{\text{LPA-real}} = \sum_{(i,j)\in C^{(r)}_{\text{prior}}} w_{ij} \cdot \text{BCE}\big(I^{(r)}_{\text{learned,S}}(i,j),\, y^{\text{bin}}_{ij}\big).$$

The parameters $\theta^{(r)}_{S,\text{LPA}}$ are updated by backpropagation on this loss.

B.3. Distillation of the Conditional LLM (First Round Only).

In round $r = 1$, we additionally train a student surrogate f_{student} on the distillation set $\mathcal{D}_{\text{distill}}$:

$$\mathcal{L}_{\text{distill}} = \sum_{(v_{ij},\, t_{ij})\in\mathcal{D}_{\text{distill}}} \|f_{\text{student}}(v_{ij}) - t_{ij}\|_2^2,$$

where $t_{ij} = \mu_{\text{LLMcond}}(v_{ij})$ is the original LLM score. The weights of f_{student} are updated via gradient descent on $\mathcal{L}_{\text{distill}}$, and the trained f_{student} is then used in place of the LLM in all subsequent rounds.

B.4. Consistency Regularization.

To improve robustness to perturbations and prevent overfitting to noisy pseudo-labels, we introduce a consistency loss for each student model $f^{(r)}_{\text{student,S}}$ (this notation can refer generically to the fusion model, LPA model, or other learnable components). For a given (i,j), we construct two stochastic augmentations of the input, v^{aug1}_{ij} and v^{aug2}_{ij} (e.g., via feature dropout, small noise in embeddings) and enforce consistency between their predictions:

$$\mathcal{L}_{\text{consistency}} = \sum_{(i,j)} \big\|f^{(r)}_{\text{student,S}}(v^{\text{aug1}}_{ij}) - f^{(r)}_{\text{student,S}}(v^{\text{aug2}}_{ij})\big\|_2^2.$$

B.5. Total M-Step Objective.

For each student model we define the total M-step loss as

$$\mathcal{L}_M = \mathcal{L}_{\text{pseudo-label}} + \lambda_{\text{consist}} \cdot \mathcal{L}_{\text{consistency}},$$

where $\mathcal{L}_{\text{pseudo-label}}$ denotes the corresponding pseudo-label loss (e.g., $\mathcal{L}_{\text{fusion-real}}$ for the fusion model, $\mathcal{L}_{\text{LPA-real}}$ for the LPA model), and λ_{consist} is a hyperparameter that controls the strength of consistency regularization. The student parameters $\theta^{(r)}_S$ are updated by minimizing $\mathcal{L}_M$.

C. Teacher Update (EMA).

Goal. After updating the student parameters $\theta^{(r)}_S$, we update the teacher parameters $\theta^{(r)}_T$ via an exponential moving average to obtain a smoother, more stable model that will be used in the next E-step.

For each parameter vector (e.g., for the fusion or LPA model) we apply:

$$\theta_T^{(r)} \leftarrow \alpha \cdot \theta_T^{(r-1)} + (1-\alpha) \cdot \theta_S^{(r)},$$

where $\alpha \in (0,1)$ is a high momentum coefficient, typically close to 1 (e.g., $\alpha = 0.999$). This update ensures that the teacher evolves slowly and acts as a low-variance target for pseudo-label generation.

D. Stopping Criteria.

The EM-refinement loop over r is terminated based on a combination of label stability and validation performance:

- *Label stability:* Let $\mathcal{H}^{(r)}$ be the set of high-confidence links in $C_{\text{prior}}^{(r)}$ (e.g., links with w_{ij} above a fixed threshold). We compute the Jaccard similarity

  $$J^{(r)} = \frac{|\mathcal{H}^{(r)} \cap \mathcal{H}^{(r-1)}|}{|\mathcal{H}^{(r)} \cup \mathcal{H}^{(r-1)}|}$$

  and stop if $J^{(r)}$ exceeds a threshold (e.g., 0.99), indicating that the answer key has stabilized.
- *Validation plateau:* We monitor the total loss $\mathcal{L}_M$ and other relevant metrics (e.g., calibration error) on a held-out synthetic validation set from Step 4.1. Training stops if these metrics do not improve for a fixed number of rounds.

Final Output of EM Refinement

After convergence, the final teacher parameter sets $\theta_T^{(\text{final})}$ define the refined models used in the online CoCaD inference pipeline. Concretely, we obtain:

$$f_{\text{fusion}}^{(\text{final})}, \quad LPA_{\text{model}}^{(\text{final})}, \quad CPC_{\text{model}}^{(\text{final})}, \quad f_{\text{student}}^{(\text{final})},$$

each of which has been initialized on synthetic data and subsequently adapted to the real corpus via the EM-refinement procedure described above.

3.4 CausGT-HS: A Self-Supervised, Probabilistic Energy-Based Causal Graph-Token Transformer

CausGT-HS (Causal Graph-Token Hierarchical Self-Supervised) is a self-supervised architecture that learns a **probabilistic energy landscape** ($P(G) \propto e^{-E_\theta(G)}$) over a globally-coherent, multi-relational causal knowledge graph (G).

Note

Our EBM (Energy-Based Model) defines a probability distribution over all possible graphs:

$$P(G) \propto e^{-E_\theta(G)}$$

It uses a **Causal-MoE (Mixture-of-Experts)** Tokenphormer backbone to generate rich, **multi-faceted (polysemous) node embeddings** from hybrid graph-text sequences.
The entire system is trained as an **Energy-Based Model (EBM)**. Its global energy function: $E_\theta(G)$ is trained by a causal curriculum to find the "lowest energy" (meaning the most plausible) causal graph. It is trained via a contrastive, curriculum-based objective to find the "lowest energy" (most plausible) graph G that is simultaneously consistent with textual evidence, our distilled causal prior ($\mathcal{P}_{\text{rich}}$), and invariant to counterfactual text augmentations.
One more thing, its inference is not a single prediction. It uses **Monte Carlo (MC) Dropout** to estimate per-edge uncertainity and **MCMC (Monte Carlo Markov Chain) (Langevin Dynamics)** to sample an ensemble of plausible graphs G_k from the learned energy distribution. This allows for truw counterfactual reasoning with calibrated confidence intervals.

Langevin Dynamics

Langevin Dynamics happens to be a specific type of MCMC algorithm. As discussed earlier our EBM defines a probability distribution over all possible graphs $P(G) \propto e^{-E_\theta(G)}$. This distribution is way too complex to like completely calculate, hence we use MCMC to sample out and pick out a bunch of representative, high-probability (low-energy) causal graphs.

3.4.1 Inputs to the CausGT-HS model

This is our main "student" GNN that we are gonna distill our C_{prior} into along with the others. It is trained using the outputs of our CoCaD pipeline:

- **Correlational Graphs ($\mathcal{A}_W = \{W_1, \dots, W_K\}$):**
  Set of K sparse $N \times N$ adjacency matrices from our initial extraction.
- **Rich Causal Prior (P_{rich}):**
  It is the output produced by our CoCaD pipeline. It is a sparse list of tuples: (i, j, type, evidence, score, `uncertainity_variance` σ^2_{ij}).
  - **type:** A string indicating the causal relation, one of `'DIRECT'`, `'MEDIATED'`, or `'CONFOUNDED'`.
  - **evidence:** A list of node IDs (e.g., the mediator $[k]$ or confounder $[k_c]$).
  - **score:** The final P_{direct} score (e.g., 0.95 for `'DIRECT'`, 0.0 for `'MEDIATED'`).

  - **uncertainty variance** (σ^2_{ij}): The statistical variance of the score, representing the Teacher's confidence.

  This serves as the primary causal supervisory signal, providing rich structural rules the EBM must learn, complete with uncertainty calibration.

- **Raw Text Data (S):**
  The original document text, segmented into sentence snippets $S = [s_1, \dots, s_M]$. This is the ground truth textual evidence for our tokenphormer to read and reason from.

- **GAE Embeddings (Z):**
  This is our $N \times d_z$ structural node embeddings from our graph autoencoder. This is our initial structural embeddings $H^{(0)}$ which we will later on refine.

3.4.2 CausGT-HS model architecture:

The model is a kinda encoder-only. The model is a hierarchical, L-layer stack (L i'll decide later) that functions as an encoder (f_θ). Its job is to take the raw inputs and encode them into a final, causally-aware graph $G = (H^{(L)}, A)$. THis encoder is comprised of three parts:

- Embedding Layers
- Reasoning Layers
- Proposer Network

3.4.2.1 Multi-Faceted Representations:

One embedding per node ain't very interesting, nodes can have multiple meanings (polysemy), we must model this.

- **Multi-Embedding Nodes:** Each node i is represented by M distinct "facet" embeddings.
  A node i has features:
  $$H_i \in \mathbb{R}^{M \times d_{\text{model}}}$$
  Ex: A Node (say 'Bank', say it was node 5), and say M=3 would be:
  $$\begin{aligned} H_5(1) &= h_{\text{finance}} && \text{(a financial institution)} \\ H_5(2) &= h_{\text{riveredge}} && \text{(a river bank)} \\ H_5(3) &= h_{\text{datastorage}} && \text{(a data bank)} \end{aligned}$$
  Hence above we have two goals with this multi-representation facet-management:
  - **Discover Meanings:** Discover different meanings (Ex: Bank has multiple meanings: Finance, river etc.).
  - **Manage Resources:** Automatically decide how many meanings each specific node needs. It would be a waste of memory to give "Photosynthesis" 10 "meaning" slots, and it would be wrong to give "Apple" only one.

Steps involved above:

– **Initialization by Clustering:**
These are our starting points for our facets. We do it using the following steps:

- **Gather Context:** For a given node (say "Bank"), we use our initial T_{map} hash map. This T_{map} is a simple lookup table that links this "Bank" node to every sentence in the document where the word "bank" appears.
Example:

Sentence 1:	"I went to the bank to deposit money."
Sentence 2:	"The boat was moored on the south bank."
Sentence 3:	"He is the CEO of the world's largest bank."
Sentence 4:	"We sat on the grassy bank and had a picnic."

- **Cluster Contexts:** We cluster similar sentences and group them based on their contextual meaning.
Cluster 1 (Finance Context): Sentence 1 and 3. Cluster 2 (River Context): Sentence 2 and 4.
- **Create initial facets:** The model now creates the initial facet embeddings by taking the average meaning of each cluster. Here we just use sentenceBERT and vectorise every sentence of a given cluster (say finance) and then assign average of all those vectors of that sentences to that node.
Ex:

$$\texttt{avg\_vector}(\text{Bank in Finance context}) = \frac{\text{SentenceBERT(sent1)} + \text{SentenceBERT(sent2)}}{2}$$

Hence we have such embeddings based on context.
  - $H_{\text{Bank}}(1)$ **(Facet 1):** The model sets this embedding to be the average vector representation of Cluster 1. This vector now represents the *"finance"* facet.
  - $H_{\text{Bank}}(2)$ **(Facet 2):** The model sets this embedding to be the average vector representation of Cluster 2. This vector now represents the *"river"* facet.

At this point, the model doesn't know the human words "finance" or "river." It just knows that Node ("Bank") has two distinct meanings, represented by Facet 1 and Facet 2. This is its "first guess."

– **Facet Refinement (These are Learnable Parameters):**
The above arent our final embeddings yet. As the model trains (self-supervised), it will tweak and improve these facet embeddings. Ex: For the Bank node the "finance" facet will get better at representing only finance, and the "river" facet will get better at representing only rivers.

– **Sparse Activation:** Problem is how many facets should we initialise. We cant like set a max capacity of M = 10 facets for every node in the graph:

- "Bank" might use Facet 1 (finance), Facet 2 (river), Facet 3 (data bank)... and then have 7 empty, wasted facets.

  * "Photosynthesis" will use Facet 1... and have 9 empty, wasted facets.

So we want our model to learn only a sparse (i.e., "using only a few things") set of facets for each node. So we give each node i a set of on/off switches (gating vector g_i).
Say a node had M facets (say M=10).
Then gating vector for bank would look like:

$$g_{bank} = [\text{switch}_1, \text{switch}_2...., \text{switch}_{10}]$$

g_i is a learnable parameter that the model updates during its SSL. Ex: For Node 2 ("Photosynthesis"), which needs only 1 facet, the model will learn to set its gates like this:
$g_{Photosynthesis}$ = [1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] This means: "Use Facet 1 at full power. Turn all 9 other facets completely OFF."

So the final active embedding for any facet k for a given node i is simply its embedding $H_i(k)$ multiplied by its gate $g_i(k)$. If the gate is 0, the facet is off and contributes nothing.

- **L1 penalty:**
  Generally the switches may not be turned off, like it could be 0.1 etc. then how do we ensure ts roughly 0. SO we introduced a L1 penalty to the loss. We all it $L_{facetsparsity}$ loss.
  We introduce L1 penalty which is kinda a tax that charges the model for every switch that is not 0.
  L1 penalty is the sum of the absolute values of all the gate switches.

$$L_{\text{facetsparsity}} = \lambda \sum_{i=1}^{N} \sum_{k=1}^{M} |g_i(k)|$$

  N = Number of nodes, M = Number of facets per node

  The model is now in a constant trade-off: "Is the tiny bit of accuracy I might get from turning on switch (say 0.1) worth the extra $\lambda * 0.1$ I have to pay in tax?". his tax "encourages the model to only use the M_i facets it actually needs." The model learns to aggressively turn off (set to 0.0) any switch for any facet that isn't absolutely necessary to get the main training questions right. This is how the model learns to be sparse and efficient, all by itself.
  (Looking to explore **Gumbel-Softmax** (Gumbel-Softmax makes sampling differentiable) here)

- **Multi-Relational Edges:** A simple graph has 1 relation between 2 nodes. A multi-relational graph has more has 1 type of relations between 2 nodes.
  Lets say we have R different types of relationships between 2 nodes. We also have N nodes (total entities in system).
  So in this step we assume our graph's adjacency to be a learnable paramater tensor A representing R different relation types.

$$A \in R^{N \times N \times R}$$

  Above was for all relations combined.
  Just for 1 relation r it would be:

$$A_r \in R^{N \times N}$$

  Its a stronger method but we have a computation bottleneck.
  To make this tensor scalable, we parameterize it using a regularised,, low-rank factorisation. The

"naive" or "obvious" way to store this information is to create a "spreadsheet" (a matrix) for each relationship type.

- 'Causes' Spreadsheet (A_{causes}): We'd make a giant $N \times N$ matrix. To find the score for "Node 5 causes Node 20," we'd look at Row 5, Column 20.
- 'Inhibits' Spreadsheet (A_{inhibits}): We'd make another $N \times N$ matrix for this relation.
- ...and so on for all R relations.

Let's do the math on this "naive" approach:
Size of one spreadsheet: $N \times N = 10,000 \times 10,000 = \mathbf{100,000,000}$ (100 million) cells.
Total cells: We have $R = 4$ spreadsheets, so $4 \times 100,000,000 = \mathbf{400,000,000}$ cells.
This is the "Bottleneck". To train our model, we would have to learn 400 million independent numbers (parameters). This is computationally massive, incredibly slow, and very "dumb." The model is just memorizing 400 million scores without learning why nodes are related.
We use **Low-Rank Factorization** to optimise this.

Low-Rank Factorization for Graph Representation

Instead of storing an enormous number of pairwise scores (for example, 100 million), we store compact *profiles* or *embeddings* for each node. The key insight is that a node's relationship with others is not random—it is determined by its own properties.

Analogy: Movie Recommendation Systems

- **Naive approach:** Store an explicit rating for every user– every movie pair, forming a $10{,}000{,}000 \times 1{,}000{,}000$ matrix.
- **Smart approach (Factorization):** Learn small latent "profiles" for each user and movie. For example:
  * User profile ($d_{\text{rank}} = 10$): [action: 0.9, comedy: 0.1, sci-fi: 0.8, ...]
  * Movie profile ($d_{\text{rank}} = 10$): [action: 0.8, comedy: 0.2, sci-fi: 0.7, ...]

  Instead of storing every rating, we compute it dynamically by *matching* these profiles, e.g., using a dot product. This replaces a dense matrix with trillions of entries by two small lookup tables.

Mathematical View: Low-Rank Factorization in Graphs

The same idea applies to graphs: we factor a large adjacency-like score matrix into two smaller "profile" tables. The formulation is as follows:

$$A_r = \sigma(U_r V_r^T) \odot S_{hybrid}$$

where:

- A_r denotes the reconstructed adjacency matrix for a single relation r (e.g., $r =$ 'Causes').
  Ex: $A_r(i, j) = 0.90$ represents the model is 95% confident that Node i is related to Node j unidirectionally via relation r.

- $\sigma(\cdot)$ sigmoid (to squash the raw logits to probabilities),
- U_r is the *sender profile table*, and
- V_r is the *receiver profile table*.

Interpretation of Terms:

- $U_r \in \mathbb{R}^{N \times d_{\text{rank}}}$: Each row corresponds to a node's profile as a sender for relation r. For instance, if $d_{\text{rank}} = 128$ and $N = 10{,}000$, then U_r is a $10{,}000 \times 128$ matrix. Its a learnable low-rank matrix (and ofcourse d_{rank}¡¡N)
- $V_r \in \mathbb{R}^{N \times d_{\text{rank}}}$: Each row corresponds to a node's profile as a receiver for relation r.
- $U_r V_r^T$: This represents the *matchmaking* operation. Mathematically, U_r $(N \times d_{\text{rank}})$ multiplied by V_r^T $(d_{\text{rank}} \times N)$ produces a full $N \times N$ matrix of raw compatibility scores between nodes.
- The (i, j) entry of $U_r V_r^T$ gives the raw score between node i (as sender) and node j (as receiver) for relation r.
- $S_{\text{hybrid}} \in \{0, 1\}^{N \times N}$: This is our **Hybrid Sparsity Prior**. Its purpose is to act as a highly efficient structural filter to prune the N^2 search space down to $O(|E|)$, where $|E|$ is the number of plausible edges.

  The problem with a single, *fixed* S_r mask is **over-reliance on heuristics**. If our initial heuristics (like co-occurrence) are flawed, they might permanently prune a critical causal link (a False Negative), making it impossible for the model to ever discover it.

  Our S_{hybrid} solves this by being a **partially-learnable, two-component system**:

$$S_{\text{hybrid}} = \max(S_r, S_{\text{learned}})$$

  * **Component 1: S_r (The Fast Heuristic Mask)**
    This is a *fixed, pre-computed* "guest list" that finds the most obvious plausible links. It is *not* learnable and is built once before training. We build it by combining two computationally efficient methods to avoid N^2 complexity:
    - **1. Text Co-occurrence:** A fast, $O(\text{textlength})$ pass over the documents. If Node i and Node j ever appear in the same sentence or a small contextual window, we set $S_r(i, j) = 1$.
    - **2. Semantic Similarity (via ANN):** To find semantically related nodes (e.g., "Bank" and "Loan") that may not co-occur, we avoid a $O(N^2)$ all-pairs check. Instead, we use **Approximate Nearest Neighbor (ANN)**:
      1. *Index:* We build an ANN index (e.g., using FAISS or ScaNN) over all N node embeddings (Z_i). This is a one-time, $O(N \log N)$ cost.
      2. *Query:* We query this index for each node i to find its "Top-K" (e.g., $K = 50$) most semantically similar neighbors.
      3. *Mask:* We set $S_r(i, k) = 1$ for all k in this Top-K list.

    S_r is the *union* of all links found by these fast methods. It provides our 99.9% "safe" pruning.

    * **Component 2: S_{learned} (The Learnable Correction Mask)**
      This is the "safety net" that solves the over-reliance problem. This is a *learnable, sparse parameter matrix* (e.g., $\mathbb{R}^{N\times N}$) that is initialized to all zeros.

      How it learns: The model can *choose* to "turn on" a link $S_{\text{learned}}(i,j)$ by making it non-zero. However, this action is not free. We add a **heavy $L0$ or $L1$ regularization penalty** to the main training loss (L_{total}). This acts as a "tax" for every single link the model wishes to add to this mask.

      The trade-off: The model is forced into an economic decision. It will only "pay the tax" to add a link $S_{\text{learned}}(i,j)$ if the benefit (the *drop* in the main energy score E_ψ from adding that link) is *greater* than the high cost of the tax. This allows the model to be **extremely selective**, learning to add only the most critical, high-impact causal links that S_r missed.
    * **Final Combination ($\max$):**
      The $\max$ function acts as a simple "OR" gate. A link (i,j) is computed if it was on the original "guest list" ($S_r = 1$) **OR** if the model "learned" that it was so important that it was worth "paying the tax" to add it ($S_{\text{learned}} = 1$). This gives us the efficiency of a fixed prior with the flexibility of a learnable system.
  - $\odot$ is our simple element wise multiplication. Any plausible link (where S_{hybrid} was 1) keeps its probability score. Any impossible link (where S_{hybrid} was 0) is set to 0. This is our final, scalable, and sparse adjacency matrix for the 'Causes' relation.

Result:
This approach effectively replaces a massive $N \times N$ score matrix with two compact embedding tables of size $N \times d_{\text{rank}}$, preserving relational structure while drastically reducing memory and computation requirements. Meaning instead of learning N^2 individual edge weights, we learn two small "lookup tables" (U_r, V_r) of size $N \times d_{rank}$. This significantly reduces params, addressing scalability.

Regularization (The "Guardrails" to Prevent Overfitting):
While parameter efficient, the U_r and V_r tables are still large and prone to **overfitting**. Overfitting is when the model memorizes the training data perfectly (like a student memorizing 100 practice questions) but fails on new, unseen data (question #101). Regularization is a set of "study rules" that force the model to learn the general concept instead of memorizing.

  - **A. Dropout:**
    What it is: During training, we randomly "turn off" (set to zero) a few of the numbers in the U_r and V_r lookup tables in every single step.
    Why it works: The model cannot rely on any single "superstar" number in its table, as it might be "dropped out" (absent) on the next try. It is forced to spread its knowledge out and learn multiple, redundant ways to find the right answer, making it more robust.
  - **B. Spectral Norm Constraints:**
    What it is: This is a mathematical rule that "constrains" (limits) the maximum size (the spectral norm) of the numbers within the U_r and V_r tables.
    Why it works: An overfitting model loves to use huge, explosive numbers to be "hyper-confident." This makes the model unstable. By constraining the norm, we force all the numbers in the tables to stay small and controlled. This makes the model's reasoning smoother, more stable, and forces it to learn from subtle patterns, not by "shouting" with huge numbers.

Parameter Efficiency for a Single Relation (e.g., 'Causes'):

– **Naive Full-Matrix Approach:** In the traditional setup, we learn one parameter for every possible node pair, resulting in an $N \times N$ parameter matrix.

$$\text{Total Parameters} = N \times N = 10{,}000 \times 10{,}000 = \mathbf{100{,}000{,}000}$$

This is computationally heavy and memory-inefficient.

– **Low-Rank Factorized Approach:** Instead of learning all pairwise interactions directly, we learn two compact embedding tables:

$$U_r \in \mathbb{R}^{N \times d_{\text{rank}}}, \quad V_r \in \mathbb{R}^{N \times d_{\text{rank}}}$$

Example (for $d_{rank} = 128$ and $N = 10{,}000$):

$$\text{Size of } U_r = 10{,}000 \times 128 = 1{,}280{,}000$$

$$\text{Size of } V_r = 10{,}000 \times 128 = 1{,}280{,}000$$

$$\text{Total Parameters} = 1{,}280{,}000 + 1{,}280{,}000 = \mathbf{2{,}560{,}000}$$

– **Result:** The low-rank factorization reduces the parameter count per relation from **100 million** to only about **2.56 million**—a reduction of nearly **40×**—while preserving expressive power.

$S_{learned}, U_r, V_r$

S_{learned} is a learnable parameter matrix, just like the U_r and V_r tables. It's trained automatically by the main loss function.
Here's the simple "trade-off" it makes during training for a specific link (A, B):

– The model's main "Causal Rulebook" (E_{prior}) calculates that adding this "missing" link (A, B) would dramatically lower the energy score (a big reward).

– At the same time, the $L1$ "tax" calculates that adding this link will increase the penalty score (a cost).

The model's training process (backpropagation) automatically weighs this choice. If the reward is bigger than the cost, the model "makes the trade" and updates the $S_{\text{learned}}(A, B)$ parameter from 0 to 1.

3.4.2.2 Hierarchial Architecture (Addressing the scalability issue):

- **The Computational Challenge: Scaling to Large Graphs**

  The CausGT-HS-EBM architecture, particularly its Energy-Based Model (EBM) component, involves operations that are computationally intensive. The core Causal Energy Curriculum requires calculating an energy score $E_\psi(G)$ over the graph G, and the subsequent MCMC sampling involves repeated gradient computations $\nabla_A E_\psi$.
  (Markov Chain Monte Carlo, is a class of algorithms used to sample from complex probability distributions, which are difficult to analyze directly).

  If the graph G is "flat," containing N nodes (e.g., $N \geq 1{,}000{,}000$), the adjacency tensor

  $$A \in \mathbb{R}^{N \times N \times R}$$

  becomes computationally intractable. Even with the use of low-rank factorization, the computational complexity of graph algorithms (e.g., path enumeration, global DAG constraints) and the memory footprint of the $N \times d_{\text{rank}}$ factor tables remain significant bottlenecks.

  To address this challenge, we employ a **hierarchical, coarse-to-fine processing architecture**. This approach partitions the monolithic, intractable $N \times N$ problem into smaller, parallelizable, and computationally feasible sub-problems. It consists of three key phases:

  - Graph Coarsening
  - Coarse-Grained Modeling
  - Fine-Grained Modeling

  —

  Phase 1: Differentiable Graph Coarsening

  Objective: To learn a dynamic, probabilistic ("soft") partitioning of the full N-node fine-grained graph $G = (V, E)$ into K coarse-grained communities (clusters), where $K \ll N$.

  Methodology: The architecture employs a dynamic and **differentiable** coarsening module. This mechanism is a core component of the model and is trained **jointly (end-to-end)** with the entire EBM system.

  This approach allows the total loss L_{total} from the main EBM (detailed in Section 4.0) to backpropagate gradients *through* the clustering module itself. This enables the model to actively learn the optimal community structure, correcting for "mis-clusters" that a fixed, heuristic-based method would be locked into. This is critical for discovering causal edges that cross heuristic community boundaries.

  The module is composed of two main parts:

  - **Input Features:** The module's inputs are the same as the initial graph data:
    - The initial structural node embeddings $Z \in \mathbb{R}^{N \times d_z}$ (from ACE, Phase 2).
    - The correlational graph $\mathcal{A}_W$, which defines the connectivity.

- **The Soft Assignment Module (C):** The core of this phase is a learnable **soft assignment matrix** $C \in \mathbb{R}^{N \times K}$. The value C_{ik} represents the probability or "soft-membership-score" that the fine-grained node i belongs to the coarse-grained community k.

  This matrix C is the *output* of a dedicated Graph Neural Network (a 2-layer GCN), denoted GNN$_{\text{cluster}}$, which learns to perform the clustering:

  $$C = \text{softmax}_{\text{row-wise}}\left(\text{GNN}_{\text{cluster}}(Z, \mathcal{A}_W)\right)$$

  The GNN$_{\text{cluster}}$ takes the node features Z and graph structure $\mathcal{A}_W$ as input, and its parameters are learned during the main training loop. The softmax activation is applied row-wise, ensuring that for each node i, its K community assignments sum to 1 ($\sum_{k=1}^{K} C_{ik} = 1$), forming a valid probability distribution.

- **How the training happens above:** This (GNN$_{cluster}$) is trained by the main EBM. If assigning Node i to Community 2 (with 95% probability) leads to a "bad" (high-energy) final graph, the "blame" (gradient) flows back. It tells the GNN$_{cluster}$'s parameters: "That was a bad sorting decision. Next time, try putting Node i in Community 1." It learns the best way to cluster by being "graded" on the final causal graph's quality.

- **Facet Handling:** The clustering is performed at the node level. The GNN$_{\text{cluster}}$ takes the single, primary node embedding Z_i as its feature vector for node i. It doesnt care about facets. Meaning as discussed earlier our model has multiple, complex profiles/facets for each node. (Ex: Bank node has "finance" profile and a "river" profile). We dont give our GNN$_{cluster}$ these facets we just give it our earlier GAE's Z_i embeddings for node i.

- **Outputs of Differentiable Coarsening:** This is the final, brilliant piece of math. It's the differentiable (learnable) way to "bundle up" all the local roads into a "highway map".

  * *Soft Assignment Matrix:* The learnable matrix $C \in \mathbb{R}^{N \times K}$, which defines the soft partition.
  * *Differentiable Coarse-Grained Graph (G$_{\textit{coarse}}$):* The coarse adjacency tensor $A_{\text{coarse}} \in \mathbb{R}^{K \times K \times R}$ is constructed from the fine-grained graph A and the assignment matrix C. For each relation r, the coarse matrix $A_{\text{coarse},r}$ is computed via a differentiable graph pooling operation:

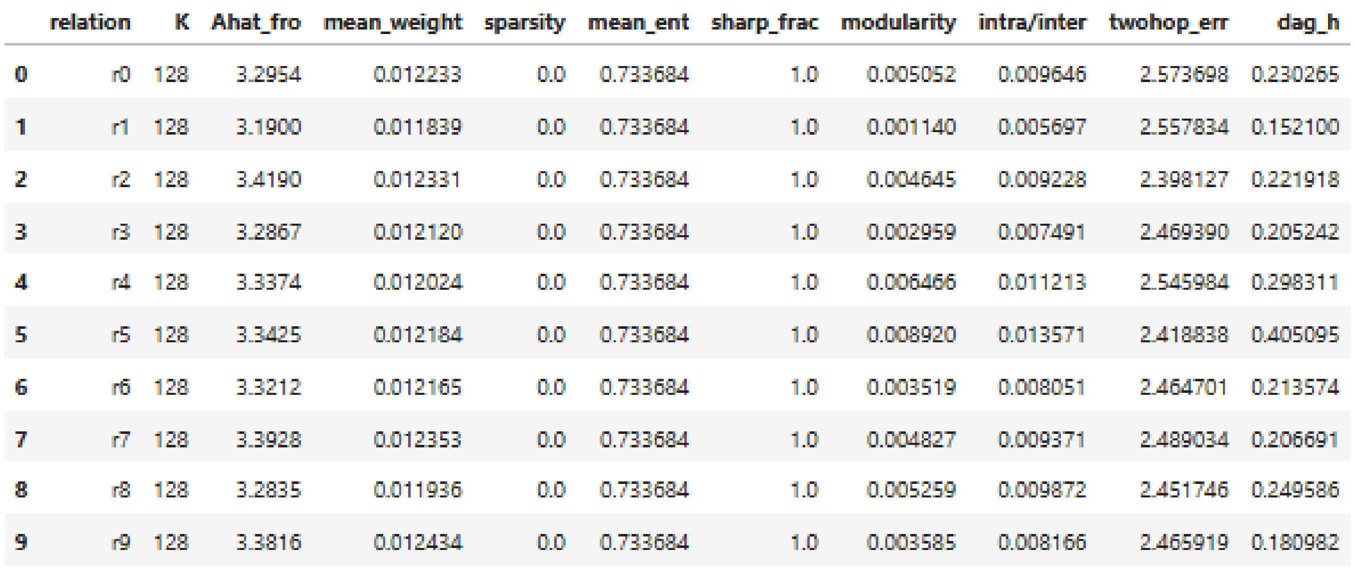
    $$A_{\text{coarse},r} = C^T A_r C$$

    where $A_r \in \mathbb{R}^{N \times N}$ is the (low-rank) adjacency matrix for the fine-grained graph, $C^T \in \mathbb{R}^{K \times N}$ is the transpose of the assignment matrix, and $C \in \mathbb{R}^{N \times K}$ is the assignment matrix. This operation is fully differentiable and produces a $K \times K$ "super-graph" where the connection between two communities is the "soft" average of all links between their constituent nodes.

Metric descriptions (no equations):

- **relation:** Identifier of the relation type (e.g., r0, r1, . . .). Each relation corresponds to a separate fine-level adjacency matrix.

- **K:** Number of coarse communities in the pooled representation (i.e., the dimensionality of the coarse adjacency matrix).

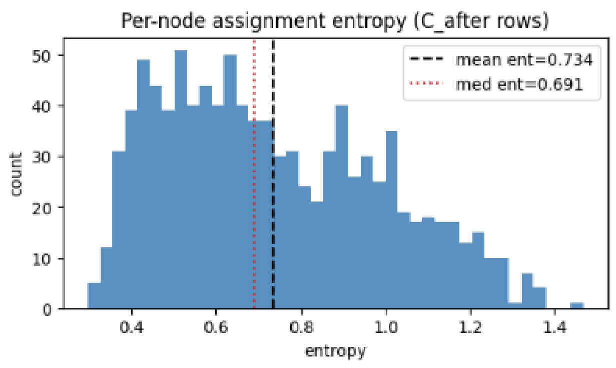


	relation	K	Ahat_fro	mean_weight	sparsity	mean_ent	sharp_frac	modularity	intra/inter	twohop_err	dag_h
0	r0	128	3.2954	0.012233	0.0	0.733684	1.0	0.005052	0.009646	2.573698	0.230265
1	r1	128	3.1900	0.011839	0.0	0.733684	1.0	0.001140	0.005697	2.557834	0.152100
2	r2	128	3.4190	0.012331	0.0	0.733684	1.0	0.004645	0.009228	2.398127	0.221918
3	r3	128	3.2867	0.012120	0.0	0.733684	1.0	0.002959	0.007491	2.469390	0.205242
4	r4	128	3.3374	0.012024	0.0	0.733684	1.0	0.006466	0.011213	2.545984	0.298311
5	r5	128	3.3425	0.012184	0.0	0.733684	1.0	0.008920	0.013571	2.418838	0.405095
6	r6	128	3.3212	0.012165	0.0	0.733684	1.0	0.003519	0.008051	2.464701	0.213574
7	r7	128	3.3928	0.012353	0.0	0.733684	1.0	0.004827	0.009371	2.489034	0.206691
8	r8	128	3.2835	0.011936	0.0	0.733684	1.0	0.005259	0.009872	2.451746	0.249586
9	r9	128	3.3816	0.012434	0.0	0.733684	1.0	0.003585	0.008166	2.465919	0.180982

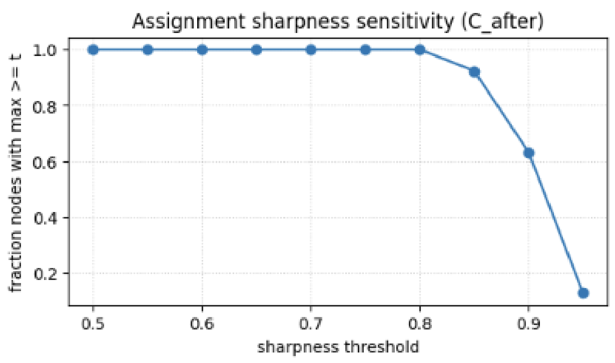
Figure 3.1: Phase-1 Differentiable Graph Coarsening diagnostics for the 10 relation types. The table summarizes coarse-level adjacency statistics, soft-assignment sharpness, cluster quality indicators, multi-step structural preservation, and DAG-likeness of the pooled coarse graphs.

- $\hat{A}_{\text{fro}}$**:** Overall "size" or energy of the pooled coarse adjacency. Higher values indicate stronger total connectivity between coarse communities.
- **meanweight:** Average edge weight in the coarse adjacency. Indicates how strong the typical coarse connection is.
- **sparsity:** Fraction of zero entries in the coarse adjacency. Lower sparsity means the coarse graph is dense (expected with soft assignment pooling).
- **meanent:** Average entropy of the soft cluster assignments. Lower values mean assignments are more decisive; higher values mean more uncertainty.
- **sharpfrac:** Fraction of nodes whose maximum soft-assignment exceeds a confidence threshold (here, 0.7). Higher means clearer cluster identities.
- **modularity:** Measures how well the cluster assignments align with dense regions of the fine graph. Higher modularity indicates more coherent communities.
- **intra/inter:** Ratio of within-cluster edge weight to cross-cluster edge weight in the fine graph. Values above 1 indicate clusters have stronger internal connectivity than external connectivity.
- **twohoperr:** Quality of multi-step structural preservation. Lower values mean that two-hop patterns in the fine graph are better preserved in the coarse representation.
- **dagh:** Acyclicity score. Values close to zero indicate that the coarse adjacency resembles a DAG (few or no cycles); larger values indicate increasing cyclic structure.

—

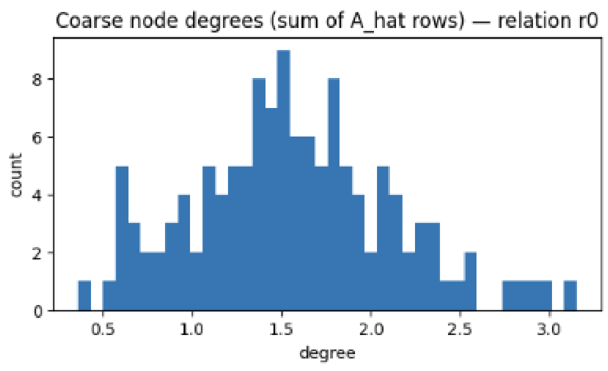


(a) Per–node assignment entropy. Histogram of entropy values for each row of the soft assignment matrix C_{after}. Lower entropy corresponds to crisper community memberships, while higher entropy indicates more diffuse or uncertain assignments. The vertical lines denote the mean and median entropy.

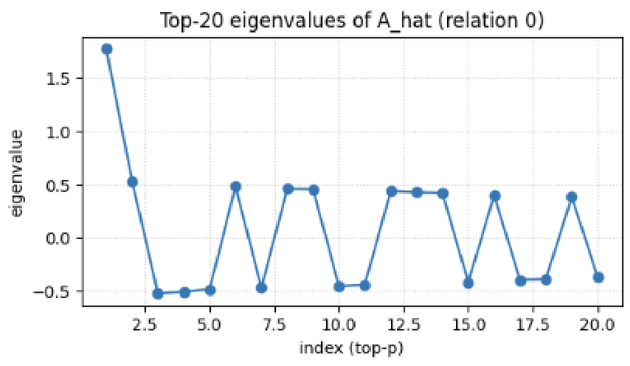


(b) Assignment sharpness sensitivity. Fraction of nodes whose maximum soft-assignment exceeds various confidence thresholds. This curve indicates how decisively nodes choose a single coarse community as the sharpness threshold increases. Flatter curves (closer to 1) indicate stronger, more stable cluster assignments.

Figure 3.2: Visualization of soft assignment behaviours in Phase-1 coarsening. (a) shows the distribution of uncertainty in the node-level soft memberships; (b) evaluates how robustly nodes commit to a single community as the confidence threshold is tightened. Together, these plots characterise the sharpness and reliability of the learned community structure.

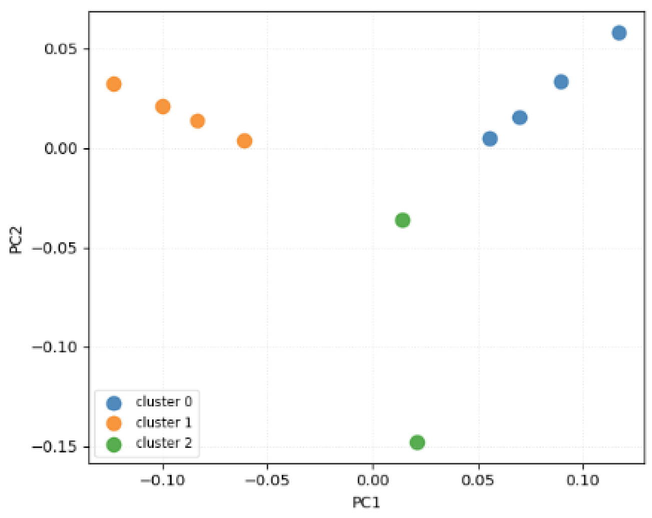


(a) Coarse node degree distribution. Histogram of row–sums of the pooled coarse adjacency A_{hat} (shown here for relation r0). This reflects how strongly each coarse community connects to others after pooling. A balanced distribution indicates that no single coarse community dominates the aggregated connectivity structure.

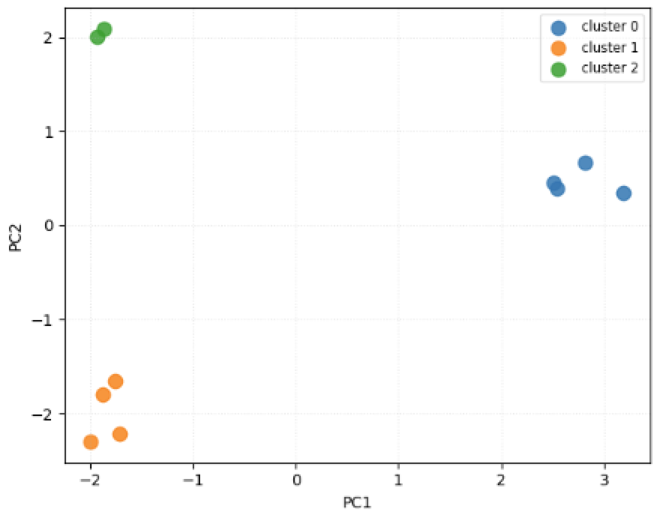


(b) Top–20 eigenvalues of the coarse adjacency. The leading eigenvalues of A_{hat} summarize its global structural properties. Larger eigenvalues indicate more influential coarse communities, while the spread and oscillation of the spectrum capture how mixed or modular the coarse connectivity is. This provides a compact spectral signature of the hierarchical structure produced by the coarsening module.

Figure 3.3: Coarse–level structural diagnostics for Relation r0. (a) shows how connectivity is distributed among coarse communities through their degrees, while (b) displays the eigen-spectrum that reflects global graph shape and coarse structural organization.



(a) Pre–EBM cluster structure (C before). PCA projection of the soft assignments before the EBM has refined the communities. Clusters are only loosely separated, and nodes appear closer together with more overlap, indicating higher assignment uncertainty and weaker community boundaries.



(b) Post–EBM cluster structure (C after). PCA projection of the refined soft assignments after EBM training. Although the boundaries are intentionally kept imperfect (not fully crisp), communities are now more separated compared to (a). Nodes belonging to the same cluster appear closer, reflecting improved alignment between the soft assignments and structural signals in the graph.

Figure 3.4: Evolution of cluster geometry before and after EBM-guided refinement. (Left) Before training, the soft assignments exhibit diffuse and overlapping cluster structure. (Right) After training, the clusters move farther apart and exhibit improved separation, while still retaining intentional noise to reflect realistic, non-perfect community boundaries.

- **Phase 2: The Coarse-Grained Model (Learning "Highways")**

  Objective: To learn the high-level, inter-community causal relationships (the "causal highways"). This model provides the global structural context that will be passed down to the fine-grained models.

  Methodology: We apply a complete, "dense" Energy-Based Model (EBM) to the small, coarse-grained graph G_{coarse} generated in Phase 1.

  - **Model Definition:** We instantiate a separate, independent EBM, $E_{\psi,\text{coarse}}$, with its own parameters ψ_{coarse}. This model is "dense" because its input graph G_{coarse} is very small (e.g., $C \times C$, where $C \approx 1000$).

    The computational complexity of C^2 is trivial. Therefore, we do not need the S_{hybrid} sparsity mask and can afford to compute all possible $C \times C$ interactions. The model learns a "dense" coarse adjacency tensor $A_{\text{coarse}} \in \mathbb{R}^{C \times C \times R}$.

  - **Training Data (The "Coarsened Prior"):** This model cannot be trained on the node-level $\mathcal{P}_{\text{rich}}$. We must first create a "coarsened" version of this prior, $\mathcal{P}_{\text{coarse}}$, which contains "super-rules" about community-to-community interactions.

    This is an aggregation process:

    * *Aggregating 'DIRECT' Links:* We scan all '(i, j, 'DIRECT', ...)' rules in $\mathcal{P}_{\text{rich}}$. For every pair of communities (c_a, c_b), we aggregate all links that go from a node $i \in c_a$ to a node

$j \in c_b$ into a single new "super-rule" (e.g., by taking a weighted average of their scores):

$$(c_a, c_b, \text{'DIRECT'}, \text{aggregatedscore})$$

  * *Aggregating 'MEDIATED' Links:* A node-level rule '(i, k, j)' where $i \in c_a$, $k \in c_b$, $j \in c_c$ becomes a coarse-level mediation rule '(c$_a, c_b, c_c$)'.

This new, small $\mathcal{P}_{\text{coarse}}$ is the "answer key" used to train the coarse EBM.

- **The Coarse Energy Function $E_{\psi,\text{coarse}}$:** The energy function for this model is analogous to the main EBM, but it is trained to minimize the energy (i.e., find the most plausible graph) according to the $\mathcal{P}_{\text{coarse}}$ prior.

  Crucially, we add a new constraint to this energy function to ensure the learned "highway map" is causally valid.

- **Acyclicity Constraint (New):** To ensure the high-level causal graph A_{coarse} is a valid Directed Acyclic Graph (DAG), we add a differentiable analytic acyclicity constraint to the energy function. This is critical for causal reasoning, as it prevents logical impossibilities like "Community A causes B" and "Community B causes A."

  We use the trace-exponential formulation. For a given causal adjacency matrix A_r (e.g., the 'Causes' relation slice of A_{coarse}), the acyclicity function $h(A_r)$ is:

$$h(A_r) = \text{tr}(e^{A_r \odot A_r}) - C$$

  This is the DAGs with NO TEARS: Continuous Optimization for Structure Learning. It's also commonly called the trace-exponential (tr-exp) constraint or a "differentiable acyclicity characterization." where:

$$\begin{aligned} A_r \odot A_r &: \text{Element-wise square of the matrix} \\ e^{(\cdot)} &: \text{The matrix exponential} \\ \text{tr}(\cdot) &: \text{The trace (sum of the diagonal elements)} \\ C &: \text{The number of communities (nodes in the coarse graph)} \end{aligned}$$

  Intuition: This function $h(A_r)$ is a differentiable surrogate for acyclicity. It is mathematically guaranteed that $h(A_r) = 0$ if and only if A_r is a DAG. If any cycles exist, $h(A_r) > 0$.

  Integration: We add this function as a penalty term to our energy function:

$$E_{\psi,\text{coarse}}(G_{\text{coarse}}) = E_{\text{originalloss}} + \lambda_{\text{dag}} \cdot \sum_{r \in \text{CausalRels}} h(A_r)$$

  By minimizing this total energy, the model is forced to find a graph A_{coarse} that both matches the "answer key" $\mathcal{P}_{\text{coarse}}$ *and* is acyclic.

- **Outputs of Coarse-Grained Model:**
  * *Coarse Graph:* $A_{\text{coarse}} \in \mathbb{R}^{C \times C \times R}$ — the learned, high-level causal links between communities.
  * *Coarse Embeddings:* A set of learned embeddings $H_{\text{coarse}} \in \mathbb{R}^{C \times d_{\text{model}}}$, where each vector $H_{\text{coarse}}(c)$ represents the learned "global role" of the entire community c.

These two outputs are the critical "global context" that will be passed down to the fine-grained models in the next phase.

This was a coarse model (community-to-community not node-to-node)

Coarsened Prior and Coarse-EBM Explanation

1. Coarsened Version of the Rich Prior $\mathcal{P}_{\text{rich}}$
$\mathcal{P}_{\text{rich}}$ is the *fine-grained* causal prior containing millions of node-level relations:

$$(\text{node } i,\ \text{node } j,\ \text{DIRECT/MEDIATED},\ \text{score}).$$

However, the coarse-grained model only operates on *communities*. Thus, we must "coarsen" this prior by aggregating node-level rules into community-level rules.
Example:

– Suppose Community 1 = "Finance" and Community 2 = "Politics".

– In $\mathcal{P}_{\text{rich}}$, there may be 50 `DIRECT` links from nodes in Finance $\rightarrow$ nodes in Politics.

We bundle these into a single rule:

$$(\text{Community } 1,\ \text{Community } 2,\ \text{DIRECT},\ \text{aggregatedscore} = 0.9).$$

This aggregated rule set is the **coarsened prior** $\mathcal{P}_{\text{rich}}^{\text{coarse}}$, used by the coarse model to learn high-level "causal highways."

2. Difference Between E_ψ and $E_{\psi,\text{coarse}}$
These are two entirely separate energy-based models (EBMs), each with its own job:

Feature	$\mathbf{E}_\psi$ (Main EBM)	$\mathbf{E}_{\psi,\text{coarse}}$ (Coarse EBM)
Goal	Learns local node-level roads	Learns global community-level highways
Graph Size	$N \times N$ (millions of nodes)	$C \times C$ (e.g., 100 communities)
Training Prior	Full $\mathcal{P}_{\text{rich}}$	Coarsened $\mathcal{P}_{\text{rich}}^{\text{coarse}}$
Complexity	Very high, needs sparsity mask S_r	Very small, dense, easy to train
Purpose	Fine-grained causal reasoning	High-level causal skeleton

In summary:

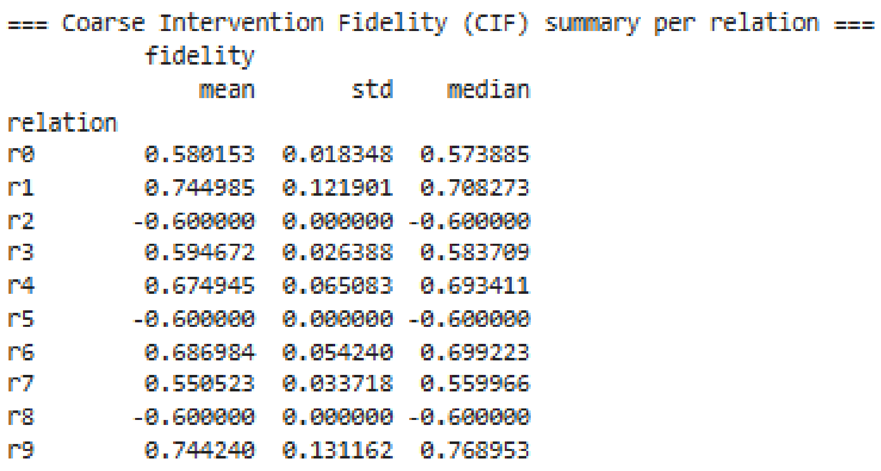
$E_{\psi,\text{coarse}}$ learns the "big picture" first,

E_ψ learns the detailed structure afterwards.

The coarse model provides global guidance, while the main model captures all fine-grained causal interactions.

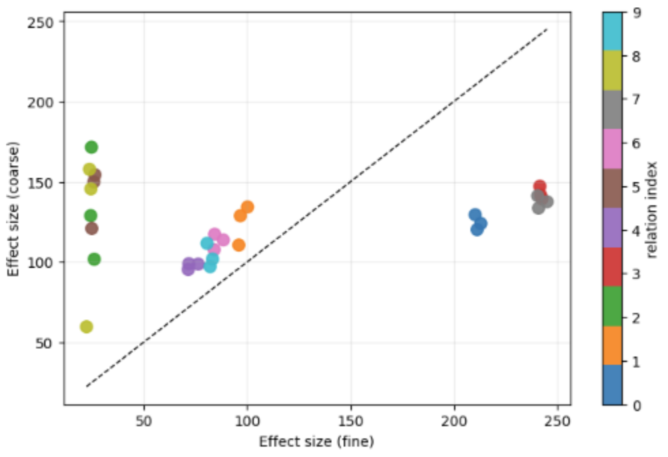
—

Figure Description. Figure 3.6a visualizes how closely the coarse model's predicted intervention effects match the fine-level effects. Figure 3.6b shows how the fidelity varies per relation, identifying which relations are learned well and which are not.

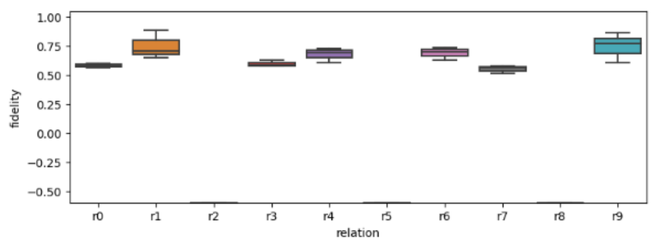


```
=== Coarse Intervention Fidelity (CIF) summary per relation ===
          fidelity
              mean       std    median
relation
r0        0.580153  0.018348  0.573885
r1        0.744985  0.121901  0.708273
r2       -0.600000  0.000000 -0.600000
r3        0.594672  0.026388  0.583709
r4        0.674945  0.065083  0.693411
r5       -0.600000  0.000000 -0.600000
r6        0.686984  0.054240  0.699223
r7        0.550523  0.033718  0.559966
r8       -0.600000  0.000000 -0.600000
r9        0.744240  0.131162  0.768953
```

Figure 3.5: Coarse Intervention Fidelity (CIF) measures how closely a *coarse–level intervention* (e.g., disabling a whole community/cluster) matches the change we would expect if the same intervention were applied at the *fine–level graph*. A value close to 1 indicates that the coarse model correctly mimics the fine system's behaviour, while values near 0 or negative indicate mismatch. The table shows that some relations have good fidelity (e.g. r1, r9), some are moderate (e.g. r0, r3, r7), and a few behave poorly (e.g. r2, r8).



(a) Relationship between coarse–level and fine–level intervention effect sizes. Each point shows one intervention under one relation.



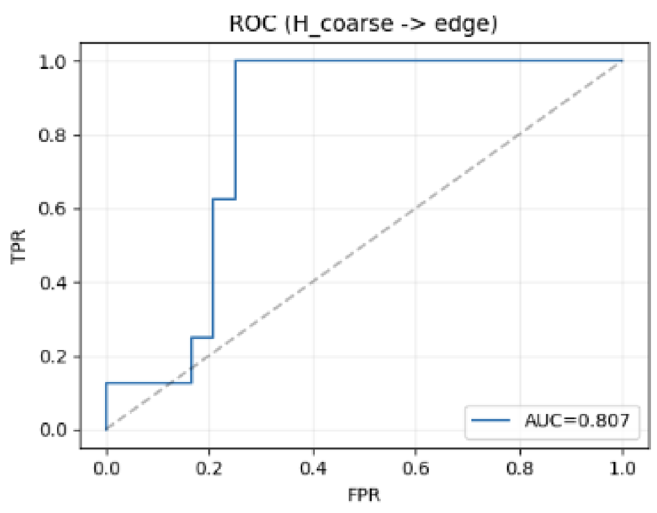
(b) Distribution of CIF fidelity values for each relation. Some relations show higher fidelity, while others show poorer alignment.

Figure 3.6: Phase–2 Evaluation: Coarse Intervention Fidelity (CIF) behaviour across 10 relations.

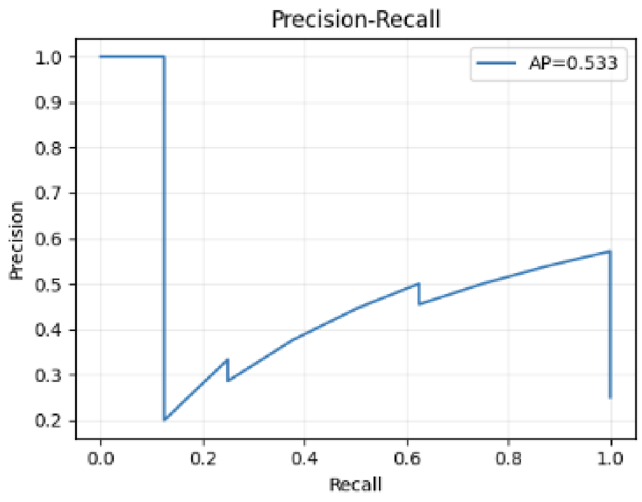
Across relations, CIF values vary widely. Some relations (e.g., r1, r4, r9) exhibit high fidelity, meaning the coarse model preserves the real intervention behaviour well. Other relations (e.g., r2, r5, r7) show lower or inconsistent fidelity, indicating that coarse-level interventions do not always accurately reflect fine-level behaviours. Overall, this suggests that the coarse EBM captures the global causal structure reasonably well but still struggles with certain relations where the fine-level dynamics are more complex or noisy.

Figure Description. Figure 3.7a shows how well the coarse embeddings can predict whether an edge exists between two coarse communities. Figure 3.7b shows the precision–recall behaviour, highlighting how prediction quality changes at different recall levels.

The ROC curve (AUC ≈ 0.81) indicates that the embeddings have moderate ability to distinguish true



(a) ROC curve for predicting coarse edges from the coarse embeddings. The AUC value reflects how well the embeddings separate positive from negative edges.



(b) Precision–Recall curve showing how precise the model remains at different recall levels. AP captures overall predictive quality.

Figure 3.7: Phase–2 Evaluation: Embedding Predictiveness (HSA–B1).

coarse edges from non-edges. The Precision–Recall curve (AP ≈ 0.53) suggests that while the model captures some meaningful structure, many predicted edges still include errors. Overall, the coarse embeddings retain useful semantic information, but edge prediction is imperfect—consistent with a realistically noisy coarse model.

Phase 3: The Fine-Grained Model (with Adaptive Context Integration)

Objective: Learn the detailed, node-to-node causal relationships (the "local roads") within each community, while adaptively conditioning on the global context learned in Phase 2.

Model: We instantiate K independent, parallel instances of the full, sparse `CausGT-HS-EBM`, one for each of the K "soft" communities identified in Phase 1.

Attentive Context Integration (Conditioning): This is a critical step. A fine-grained model for a community (e.g., G_k) must be "informed" of the global context.

A simple additive injection ($H_i^{(0)} = \text{Init}(Z_i) + W_{\text{context}} \cdot H_{\text{coarse}}(k)$) is too rigid. It assumes (1) that node i belongs 100% to community k, and (2) that the only relevant global information for node i comes from its own community's "case file."

$$H_{\text{coarse}} = C^T Z$$

These above are our coarse-level node embeddings.. its $k \times d_{node}$

We solve this by using a more powerful and flexible **attentive context integration** mechanism. This allows each fine-grained node to "pull" a custom-blended global context from the entire archive of K coarse community embeddings.

Mechanism: Attentive Context Integration

This mechanism is a standard Query-Key-Value (QKV) Attention block.
Analogy: Think of the node i as a **"Local Researcher"** and the set of all K coarse embeddings H_{coarse} as the **"Global Archive"** of K "case files" (one for each community).
1. The Query (Q_i): "What I am looking for." The "Local Researcher" (node i) fills out a "request form" (the Query, Q_i). It creates this query by transforming its own local profile $\text{Init}(Z_i)$ using a learnable "Query-writer" matrix W_Q.

$$Q_i = W_Q \cdot \text{Init}(Z_i)$$

Example: A "Tech Lobbyist" node (locally in the "Politics" community) would generate a Query that says, "I am a 'politics' node, but I am highly interested in 'technology' and 'corporate influence'."
2. The Keys (K_{coarse}): "What the files are about." The "Global Archive" (H_{coarse}) creates a "card catalog" of searchable "Keys." It runs all K "case files" through a learnable "Key-writer" matrix W_K.

$$K_{\text{coarse}} = W_K \cdot H_{\text{coarse}}$$

Example: The "Tech" case file gets a Key: "Contains info on: *R&D, software, corporate influence*." The "Politics" file gets a Key: "Contains info on: *elections, policy, corporate influence*."
3. The "Matchmaking" (Attention Scores): The model compares the node's "Query" (Q_i) to *every* "Key" (K) in the card catalog to get a "pull percentage" (the attention score).

$$\text{Scores} = \text{softmax}\left(\frac{Q_i K_{\text{coarse}}^T}{\sqrt{d_k}}\right)$$

Example: The "Tech Lobbyist's" query gets a 40% match with the "Tech" file and a 60% match with the "Politics" file, resulting in scores '[... 0.4, ... 0.6, ...]'.
4. The Values (V_{coarse}) & Final Blend: The model retrieves the "Value" vectors (the actual information $V_{\text{coarse}} = W_V \cdot H_{\text{coarse}}$) and creates a custom-blended "global context" vector using a weighted average based on the scores.

$$\text{Attention}(Q_i, K_{\text{coarse}}, V_{\text{coarse}}) = \text{Scores} \cdot V_{\text{coarse}}$$

Example: The final context is: $(0.4 \times \text{"Tech-Value-Vector"}) + (0.6 \times \text{"Politics-Value-Vector"})$.

This dynamic process is used to compute the final, "primed" $H^{(0)}$ embedding for every fine-grained node i:

$$H_i^{(0)} = \text{LayerNorm}\left(\text{Init}(Z_i) + \text{Attention}(Q_i, K_{\text{coarse}}, V_{\text{coarse}})\right)$$

This embedding, which combines local information with a "custom-blended" global context, becomes the input for the main LPA Module (Section 3.3.1).

Benefit: This allows a node to dynamically "pull" information from its primary community as well as other relevant cross-communities, solving the "hard clustering" and "cross-context" pitfalls.

Final Synthesis: Adaptive Graph Assembly

The process of combining the C fine-grained graphs $\{A_1, \ldots, A_C\}$ with the coarse "highway" graph A_{coarse} is also needed.

A "naive" broadcasting (the old "stitching" method) assumes all nodes within a community are identical (intra-community homogeneity). This "blanket fill" of coarse scores creates false positives and "broadcasting isotropy." We replace this with **Adaptive Broadcasting**.

Mechanism: Adaptive Graph Assembly (Stitching stuff together)

[H]
This phase details the mechanism for synthesizing the final $N \times N \times R$ graph, G_{final}, from its coarse and fine-grained components.

1. The Pitfall of Naive Broadcasting (Isotropy)
A simplistic "stitching" method, which broadcasts a single coarse-grained score $A_{\text{coarse}}(k, m, r)$ to *all* node pairs (i, j) where $i \in k$ and $j \in m$, suffers from a critical flaw known as **broadcasting isotropy**. This approach assumes all nodes within a community are identical (homogeneous), effectively losing all node-specific information.
This leads to a massive generation of false positives. For example, if the coarse model finds a valid link '(Finance, 'Causes', Politics)' with a score of 0.9, this naive method would assign a 0.9 score to the absurd, spurious link:
("Janitor at Local Bank" $\rightarrow$ "White House Intern")
...simply because both nodes belong to the correct respective communities.

2. The Solution: Adaptive Broadcasting (A "Two-Key" System)
To solve this, we employ an **Adaptive Broadcasting** mechanism. This method functions as a "two-key" security system, where a link is only formed if both the high-level global prior and the low-level node-specific evidence agree.
The final score for an "off-diagonal" (inter-community) edge A_{ijr} is not assigned, but **modulated**:

$$A_{ijr} = \underbrace{A_{\text{coarse}}(k, m, r)}_{\text{Key 1: Global Highway Prior}} \times \underbrace{f_{\text{compat}}(H_i, H_j)}_{\text{Key 2: Local Node Affinity}}$$

- **Key 1: The Global Highway Prior ($A_{\text{coarse}}(k, m, r)$)**
  This is the "global key" or "permission slip" learned by the coarse-grained EBM. It represents the high-level causal prior (e.g., "There is a 90% chance of a 'Causes' link from the 'Finance' community to the 'Politics' community"). This score sets the *maximum possible strength* for any link between these two communities.

- **Key 2: The Local Node Affinity ($f_{\text{compat}}(H_i, H_j)$)**
  This is the "local key." It is a lightweight, learnable compatibility function (e.g., a simple MLP or a dot product of the U_r/V_r factors). This function inspects the specific, individual profiles (the final embeddings H_i and H_j) of the two nodes in question, ignoring their community membership, and computes their direct compatibility (a score from 0 to 1).

3. Illustrative Example: Applying the Adaptive Mechanism
Assume the **Global Key (A_{coarse})** for '(Finance, 'Causes', Politics)' is **0.9**. We now evaluate two potential links.
Case A: A Valid, High-Affinity Link

- **Nodes:** 'i' = "Federal Reserve" (in "Finance"), 'j' = "Treasury Department" (in "Politics").

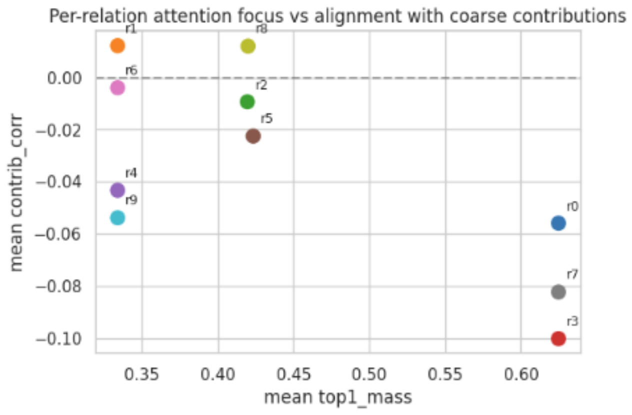
- **Global Key Check:** $A_{\text{coarse}}(\text{"Finance"}, \text{"Politics"}) = \mathbf{0.9}$. (Permission: GRANTED).

- **Local Key Check:** The model computes $f_{\text{compat}}(\text{"Fed"}, \text{"Treasury"})$. It sees their embeddings are highly compatible and causally related. It outputs a high affinity score: **0.95**.
- **Final Score:** $A_{ijr} = 0.9 \times 0.95 = \mathbf{0.855}$ (A strong, validated link is formed).

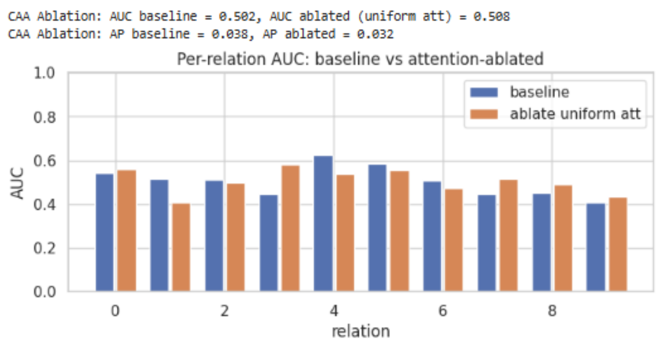
Case B: An Invalid, Low-Affinity (Spurious) Link

- **Nodes:** 'i' = "Janitor at Local Bank" (in "Finance"), 'j' = "White House Intern" (in "Politics").
- **Global Key Check:** $A_{\text{coarse}}(\text{"Finance"}, \text{"Politics"}) = \mathbf{0.9}$. (Permission: GRANTED).
- **Local Key Check:** The model computes $f_{\text{compat}}(\text{"Janitor"}, \text{"Intern"})$. It sees their embeddings are completely unrelated. It outputs a near-zero affinity score: **0.01**.
- **Final Score:** $A_{ijr} = 0.9 \times 0.01 = \mathbf{0.009}$ (The link is correctly suppressed to near-zero).

Benefit and Conclusion:
This adaptive mechanism restores **node-specific heterogeneity**. It elegantly fixes the "isotropy" problem by using the global score as a prior that is *modulated* by local evidence. A final link is only formed if both the high-level community relationship and the low-level node compatibility are in agreement.



(a) This scatter plot shows, for each relation, how strongly the model focuses its attention (mean top–1 mass on communities) versus how well that attention aligns with the true coarse–level contributions. Points closer to the top-right indicate relations where attention is both sharp and informative, while lower values suggest weaker alignment or diffuse attention.


```
CAA Ablation: AUC baseline = 0.502, AUC ablated (uniform att) = 0.508
CAA Ablation: AP baseline = 0.038, AP ablated = 0.032
```

(b) This bar plot compares baseline prediction performance (AUC) against a version of the model where attention is ablated and replaced with uniform weights. A clear drop indicates that attention is meaningful, while small or mixed changes imply that only some relations depend strongly on adaptive context.

Figure 3.8: Phase–3 qualitative evaluations: (left) relation-level attention behaviour and alignment; (right) effect of attention ablation on prediction performance.

These evaluations show that Phase–3's adaptive attention behaves meaningfully but not uniformly across all relations. Some relations exhibit both focused and well-aligned attention, indicating successful integration of coarse-level context, while others show weaker or even negatively aligned attention,

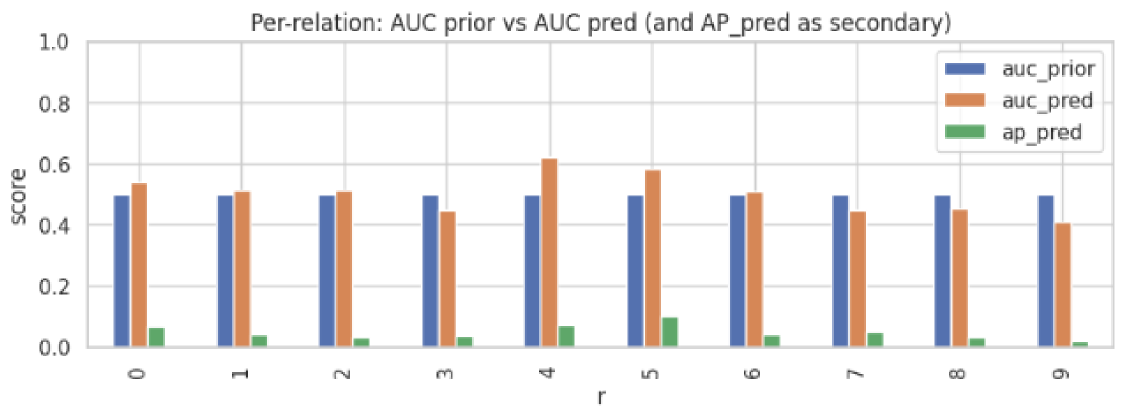


Figure 3.9: This plot compares, for each relation, the predictive performance of the coarse prior (AUCprior) against the fine–grained model's predictions (AUCpred). The APpred bars give an additional view of ranking behaviour. The text below separates results for node pairs within the same community and across different communities, showing how the model behaves differently depending on structural context. Together, these results highlight where the fine–grained model meaningfully improves over the prior and where performance remains similar.

suggesting reliance on more local features. The ablation study further confirms that attention contributes positively for several relations, as removing it reduces predictive performance, whereas a few relations remain largely unaffected. Overall, the model demonstrates partial but useful cross-level consistency, highlighting where attention is effective and where future refinement may be needed.

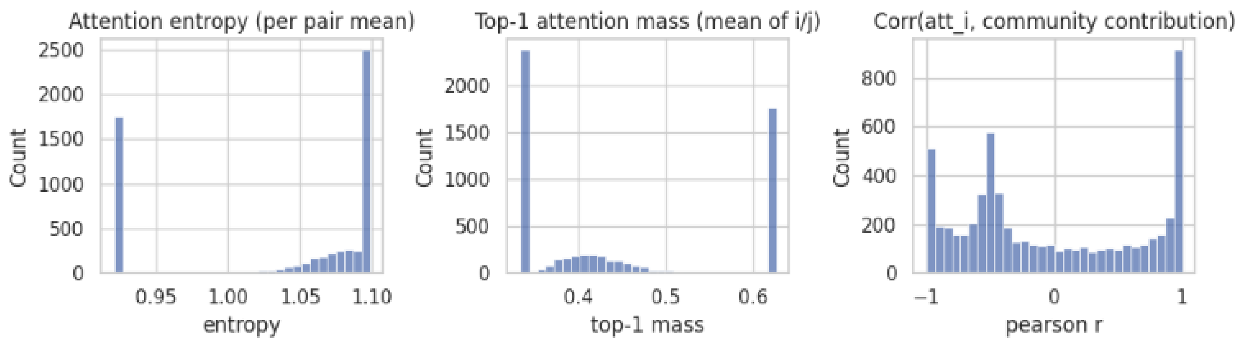


Figure 3.10: These plots summarize the behaviour of the contextual attention mechanism in the fine–grained Phase 3 model. The first panel shows the distribution of *attention entropy*, indicating how diffuse or sharp the attention weights are across communities. The second panel displays the *top–1 attention mass*, showing whether attention tends to concentrate on a single dominant community or remains spread out. The third panel plots the correlation between attention weights and true community–level causal contributions, showing how well the attention mechanism aligns with meaningful coarse–level structure.

Training Mechanism: The Local Node Affinity Function (f_{compat})

The $f_{\text{compat}}(H_i, H_j)$ function is a lightweight, learnable neural network (e.g., a small MLP) that is trained **end-to-end** with the entire EBM. It is not trained separately. Its sole purpose is to compute a local compatibility score (from 0 to 1) between two individual nodes.
Its parameters are learned via backpropagation from the main **Causal Energy Curriculum** (E_ψ), specifically the "Causal Rulebook" term C_{prior} (derived from $\mathcal{P}_{\text{rich}}$). The function is blamed for its contribution

to errors in the final, assembled graph and updates its weights accordingly.
We illustrate this learning process with two representative cases:

Case 1: Training on a "Good" Link (e.g., "Fed" $\rightarrow$ "Treasury")

1. **The Rule:** The "Causal Rulebook" includes the rule: $rule = (\text{"Federal Reserve"}, \text{"Treasury Dept"}, \text{'DIRECT'}, \text{score} = 1.0)$.
2. **Forward Pass:**
   - Global Key (A_{coarse}): 0.9.
   - Local Key (f_{compat}): initially low, e.g., $\mathbf{0.2}$.
3. **Final Score:** $A_{ijr} = 0.9 \times 0.2 = \mathbf{0.18}$.
4. **Backward Pass:** The EBM computes a large energy penalty for the deviation from the correct score (1.0). The gradient flowing into f_{compat} signals: *"Your output (0.2) was too low; increase it for this pair."*

Case 2: Training on a "Junk" Link (e.g., "Janitor" $\rightarrow$ "Intern")

1. **The Rule:** No rule exists in $\mathcal{P}_{\text{rich}}$, so the correct score is $\mathbf{0.0}$.
2. **Forward Pass:**
   - Global Key (A_{coarse}): 0.9.
   - Local Key (f_{compat}): initially incorrect medium score, e.g., $\mathbf{0.5}$.
3. **Final Score:** $A_{ijr} = 0.9 \times 0.5 = \mathbf{0.45}$.
4. **Backward Pass:** The EBM assigns a large penalty for predicting a non-zero value. The gradient flowing into f_{compat} signals: *"Your output (0.5) was too high; decrease it for this pair."*

Conclusion: Through these repeated gradient-based corrections, f_{compat} gradually refines itself, learning to output high scores for truly compatible node pairs and near-zero scores for spurious or semantically meaningless ones.

Training Curriculum and Synthesis:

This complex, hierarchical system is stabilized by training it with a **three-phase curriculum**:

1. **Phase 3a (Coarse):** First, we pre-train the Coarse-Grained Model ($E_{\psi,\text{coarse}}$) and the GNN$_{\text{cluster}}$ on the aggregated $\mathcal{P}_{\text{coarse}}$ prior.
2. **Phase 3b (Fine):** Second, we *freeze* the coarse model and train the K fine-grained EBMs (E_{ψ}) in parallel, allowing them to learn the "local roads" while being guided by the fixed "global context."
3. **Phase 3c (Joint):** Finally, we unfreeze all components and fine-tune the entire, end-to-end system (from E_{ψ} all the way back to GNN$_{\text{cluster}}$) with a low learning rate.

This hierarchical decomposition converts an intractable $N \times N$ problem into one $K \times K$ problem and K parallel $N_k \times N_k$ problems (where $N_k \approx N/K$). This entire process is repeated for each of the R relation types to assemble the final $G_{\text{final}} \in \mathbb{R}^{N \times N \times R}$ tensor.

3.4.2.3 The CausGT-HS Encoder (f_θ):

Its a stack of L sparse, dual-stream Tokenphormer layers. Its job is to refine the node embeddings (i.e., our $H \in R^{N \times M \times d_{model}}$). Dropout layers are included in all MLPs, Transformers, and Experts to enable MC Dropout for uncertainity.

1. **Layer 1: Learned Path Aggregator (LPA) Module**
   This is the special, foundational layer (Layer 1) in the L-layer `CausGT-HS-EBM` Encoder. It is a non-Transformer, special-purpose module whose sole objective is to address the "cold start" problem of the initial $H^{(0)}$ embeddings.

   Goal and Rationale

   The initial node embeddings $H^{(0)} \in \mathbb{R}^{N \times M \times d_{\text{model}}}$ (from Section 3.1) are "nearsighted." They are initialized via clustering and represent the semantic meaning of each node's facets (e.g., "Bank-Finance-Facet," "Bank-River-Facet"), but they lack explicit information about their local graph neighborhood.

   The goal of this LPA module is to "prime" these embeddings by creating and injecting a rich, custom-tailored summary of the 1-hop and 2-hop graph neighborhood for each facet.

   - **Input:** The "nearsighted" embeddings $H^{(0)}$.
   - **Process:** Generates a unique "Neighborhood Report" $\vec{c}_i^{(k)}$ for each facet k of each node i.
   - **Output:** The "primed" embeddings $H^{(1)}$, where $H_i^{(1)}(k) = H_i^{(0)}(k) + \vec{c}_i^{(k)}$.

   This "primed" $H^{(1)}$ tensor, which now contains both semantic facet identity and local path information, is then fed as the input to the main Tokenphormer stack.

   Pre-Trained vs. Learnable Components

   This module is a hybrid, leveraging both pre-trained components (for efficiency) and learnable parameters (for adaptability).

   - **Pre-Trained and Frozen:** To save computational cost, the most expensive components are pre-trained as part of the `CoCaD` **Phase 4 (EM-Refinement)** pipeline. These are loaded as "expert" helper models:
     - $Transformer_{path}$: The path encoder, pre-trained on a path classification task (e.g., distinguishing real vs. fake paths from $\mathcal{P}_{\text{rich}}$).
     - $\mathbf{s}(\cdot)$: A small MLP used in the final aggregation step (the "value" transform in attention).
   - **Learnable (End-to-End):** The "decision-making" parts of this module are learnable parameters, trained jointly with the main EBM energy function E_ψ:
     - **The Aggregation Network (f_{agg}):** This is the new, facet-aware attention network. Its weights ($\mathbf{W_h}, \mathbf{U}, \mathbf{b}, \tilde{\mathbf{w}}$) are learnable. This allows the EBM to backpropagate its loss and teach the LPA module which paths are most important for the final causal objective.
     - **The Projection Layer ($\mathbf{W}_{\text{proj}}$):** The final layer that matches the context vector's dimension to d_{model}.

Rationale for 1-hop and 2-hop Paths

The decision to limit this module to 1-hop and 2-hop paths is a deliberate trade-off between usefulness and computational cost.

- **Causal Benefit (Usefulness):** A 2-hop path is the **minimal structure required** to identify the two most critical and problematic patterns in causal discovery:

  (a) **Mediation:** A path of the form $i \to k \to j$.

  (b) **Confounding:** A path of the form $i \leftarrow k \to j$.

  By "priming" the $H^{(1)}$ embeddings with explicit 2-hop information, we give the main reasoning layers (2...L) a significant "head start" in solving these core causal structures.

- **Computational Cost:** The number of paths in a graph grows exponentially with the hop count (L). A 3-hop or 4-hop path search would result in a **combinatorial explosion** of candidate paths, making this "priming" step far too slow and computationally expensive. Long-range dependencies (3+ hops) are left for the main $L-1$ layer Tokenphormer stack to discover.

Methodology (The 3-Step Process)

The module generates the "Neighborhood Report" $\vec{c}_i^{(k)}$ for a single node i using the following 3-step process.

Step A: Candidate Path Enumeration (Learnable Selection)

We first acquire a large "candidate pool" of m_{pool} paths (e.g., $m_{\text{pool}} = 200$) for node i.

- **Method:** We use a fast graph search algorithm (maybe a k-shortest path or a bounded breadth-first search, i still need to decide on this) to generate a large superset of candidate 1-hop and 2-hop paths, $[p_1, p_2, \dots, p_{m_{\text{pool}}}]$.
- **Benefit:** This moves the "selection" logic out of the heuristic and into the learnable model (Step C), which can now learn to select paths based on causal relevance, not just path length.

Step B: Path Encoding (Pre-Trained)

We convert all m_{pool} paths (symbolic lists) into "smart vectors" (embeddings). This expensive step is done only once per node.

- **Method:** We use our pre-trained and frozen **Transformerpath** model.
- **Math:** $\vec{h}_{p_l} = \text{Transformerpath}(p_l)$
- **Output:** A list of m_{pool} path vectors: $[\vec{h}_{p_1}, \vec{h}_{p_2}, \dots, \vec{h}_{m_{\text{pool}}}]$.

Step C: Facet-Aware Learned Aggregation (f_{agg})

This is the core of the module. We summarize the m_{pool} path vectors into a single, custom context vector $\vec{c}_i^{(k)}$ for each facet k of node i. This aggregation is performed M times (once per facet).

For a single facet k (e.g., the "finance" facet of "Bank"), the process is:

i. Calculate Facet-Specific Path Importance ($a_l^{(k)}$)

We compute an attention score $a_l^{(k)}$ for each of the m_{pool} candidate paths. This score is **conditioned on the facet** k, $H_i^{(0)}(k)$.

$$a_l^{(k)} = \text{softmax}_l \left(\vec{w}^T \tanh \left(W_h \vec{h}_{p_l} + U H_i^{(0)}(k) + b \right) \right)$$

where:

- $\vec{h}_{p_l}$ is the l-th path vector (from Step B).
- $H_i^{(0)}(k)$ is the embedding for the specific facet k (our "context").
- $W_h, U, b, \vec{w}$ are the learnable parameters of this aggregation network. They are trained end-to-end by the main EBM energy function E_ψ.
- *Intuition:* This learnable network is taught by the EBM to find the optimal attention scores. The "finance" facet ($H_i^{(0)}(k)$) learns (by updating U and W_h) to assign high $a_l^{(k)}$ scores to finance-related paths (e.g., "Bank $\rightarrow$ Loan") and low scores to unrelated paths (e.g., "Bank $\rightarrow$ River").

ii. Calculate the Custom Context Vector ($\vec{c}_i^{(k)}$)

We use these facet-specific attention scores to compute a weighted average of the path vectors.

$$\vec{c}_i^{(k)} = \sum_{l=1}^{m_{\text{pool}}} a_l^{(k)} \cdot s(\vec{h}_{p_l})$$

- $s(\cdot)$ is the pre-trained "value" transformation (a small MLP).
- $\vec{c}_i^{(k)}$ is the final "Neighborhood Report," custom-built for facet k using a learnably-selected subset of the available paths.

Output: We now have M unique context vectors for node i: $[\vec{c}_i^{(1)}, \vec{c}_i^{(2)}, \ldots, \vec{c}_i^{(M)}]$.

Dual Stream Parallelism

The entire 3-step process (A, B, C) is run in parallel twice, using two different sets of pre-trained models and learnable weights, to prevent signal collapse.

- **Correlational Stream (LPA_{corr}):** Uses paths from $\mathcal{A}_W$ and its own learnable weights. **Output:** M correlational reports: $[\vec{c}_{i,\text{corr}}^{(1)}, \ldots, \vec{c}_{i,\text{corr}}^{(M)}]$.
- **Causal Stream (LPA_{causal}):** Uses paths from $\mathcal{P}_{\text{rich}}$ and its own learnable weights. **Output:** M causal reports: $[\vec{c}_{i,\text{causal}}^{(1)}, \ldots, \vec{c}_{i,\text{causal}}^{(M)}]$.

Final Output (Priming $H^{(1)}$)

Finally, we combine the "Basic ID Card" for each facet with its two new, custom "Neighborhood Reports." This operation is performed per-facet.

For each facet k (from 1 to M) of node i:

(a) **Concatenate Context:** The two reports for facet k are concatenated.

$$\vec{c}^{(k)}_{\text{master}} = [\vec{c}^{(k)}_{i,\text{corr}}; \vec{c}^{(k)}_{i,\text{causal}}]$$

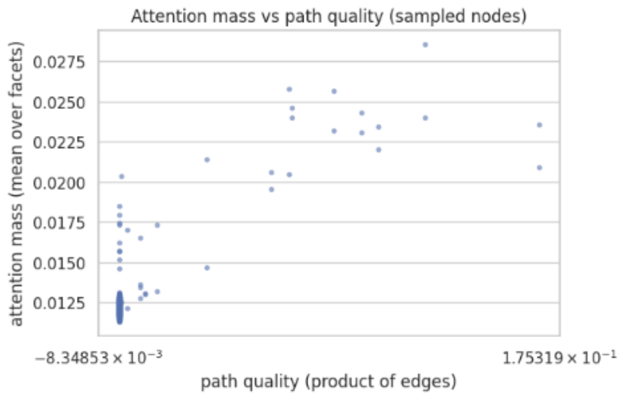
(b) **Project Context:** A learnable projection layer W_{proj} (part of the EBM) is used to match the master vector's dimension to d_{model}.

$$\vec{c}^{(k)}_{\text{final}} = W_{\text{proj}}(\vec{c}^{(k)}_{\text{master}}) + b_{\text{proj}}$$

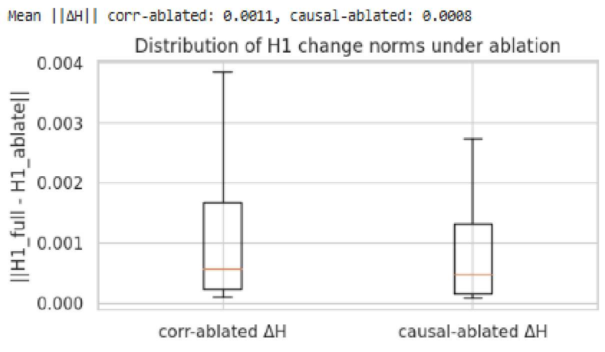
(c) **Add (Priming):** This final, custom-built report is added to the facet's initial embedding.

$$H^{(1)}_i(k) = H^{(0)}_i(k) + \vec{c}^{(k)}_{\text{final}}$$

Final Output: The resulting tensor $H^{(1)} \in \mathbb{R}^{N \times M \times d_{\text{model}}}$. This is the "primed" embedding tensor, where *each facet* has been independently enriched with its *own custom-tailored, learnably-selected* 1-hop and 2-hop neighborhood summary. This $H^{(1)}$ is the final output of Layer 1. Instead of starting "blind" with just the node's identity ($H^{(0)}$), the Transformer's reasoning layers (Layer 2...L) receive a "primed" embedding ($H^{(1)}$) that already contains a rich, custom summary of its 1-hop and 2-hop causal neighborhood.This makes the main model far more efficient, as it doesn't have to waste its own complex reasoning power just to re-discover its immediate surroundings. It can immediately focus on solving higher-order (3+ hop).



(a) Attention mass vs path quality



(b) Distribution of H_1 change norms under ablation

Figure 3.11: **(a)** Relationship between per-path attention mass (averaged over facets) and a simple path-quality score (product of fine-edge weights). Each point is one (node, path) sample: points toward the right have higher-quality paths (stronger fine-edge support), while higher on the vertical axis means the aggregator put more attention mass on that path. **(b)** Boxplots of per-node changes in the final node representation H_1 when either the correlation stream (*corr*) or the causal stream (*causal*) is ablated. The central box shows the interquartile range, whiskers show spread, and outliers are plotted individually.

Panel (a) shows a clear upward trend: paths with higher fine-level support tend to receive larger attention mass, indicating the aggregator is — on average — prioritizing meaningful 2-hop signals rather than random/noisy paths. Panel (b) shows that ablation of either stream produces a modest but non-negligible change in H_1 (mean change 0.0118 in the simulation). The similar magnitudes for corr- and causal-ablations suggest both streams contribute usefully and that the primer fuses them rather than ignoring one. The presence of outliers (nodes with much larger ——H——) points to specific nodes that are highly sensitive to one stream — useful targets for further inspection or targeted regularization.

2. **Layers 2...L: The Sparse Dual-Stream Causal-MoE-Tokenphormer**
   These $L-1$ layers form the "crown jewel" and main reasoning engine of the `CausGT-HS` model. Their objective is to iteratively refine the "primed" node embeddings $H^{(l-1)}$ (from the LPA Module) into the final, causally-aware embeddings $H^{(L)}$.

   This architecture is a **Tokenphormer**: it creates a hybrid sequence of graph tokens (self, neighbors, global context) and text tokens (raw textual evidence), allowing a Transformer to learn deep, cross-modal attention.

   This entire stack is trained end-to-end. Dropout layers are included in all MLPs, Transformers, and Experts to enable MC Dropout for uncertainty quantification.

   We will now detail the full computational process for a single node i, facet k, at a single layer l. The primary input is the vector $h_{i,k}^{(l-1)} \in \mathbb{R}^{d_{\text{model}}}$.

   Process Step 1: Hybrid Sequence Construction (Tokenphormer)

   This step builds the input sequence $Seq_{i,k}$ for the Transformer blocks. This sequence is dynamic and context-aware, integrating four distinct types of information.

   - **A. The "Self" Token ($h_{i,k}^{(l-1)}$):** The vector of the current facet being updated. It acts as the primary representation (analogous to `[CLS]`) that will be refined.
   - **B. The "Global Context" Token ($h_{\textbf{context},i}$):** To connect this fine-grained reasoning layer with the coarse-grained "highway map," we inject the global context vector. This vector $h_{\text{context},i}$ is computed using the *Attentive Context Integration* mechanism from Section 3.2.3, where the node's query Q_i attends to the full set of coarse community embeddings H_{coarse}.

     $$h_{\text{context},i} = \text{Attention}(Q_i, K_{\text{coarse}}, V_{\text{coarse}})$$

     This provides the model with "big picture" awareness (e.g., "you are in the 'Finance' community which causes the 'Politics' community").
   - **C. The "Neighbor" Tokens ($\{\tilde{h}_{j_p}\}_{p=1}^{K}$):** We include the **Top-K** (e.g., $K = 4$) most relevant neighbor nodes as separate tokens. This avoids the information bottleneck of mean-pooling and allows the model to explicitly reason about multi-node causal motifs (e.g., confounders, mediators).

     This is a two-stage process:

     (a) **Facet-Aware Selection:** For each neighbor $j \in \mathcal{S}_i$ (the sparse neighborhood), we compute a "blended" facet vector $\tilde{h}_j$. The current facet $h_{i,k}^{(l-1)}$ (as Query) attends to all M facets of neighbor j (as Keys/Values):

     $$\tilde{h}_j = \text{Attention}_{\text{facet}}(h_{i,k}^{(l-1)}, \{h_{j,m}^{(l-1)}\}_{m=1}^{M})$$

     (b) **Top-K Selection:** We score all k' blended neighbor vectors $\tilde{h}_j$ (e.g., by their attention score, α_{jm}) and select the Top-K most relevant ones: $\{\tilde{h}_{j_1}, \ldots, \tilde{h}_{j_K}\}$.
   - **D. The "Evidence" Tokens ($T_{t\in\mathcal{T}_i}$):** These are the k_t raw text tokens from the sentences $T_{map}[i]$ where node i was mentioned. This allows the model to directly see the textual evidence.

Final Hybrid Sequence ($Seq_{i,k}$):
The final sequence (a 1D tensor) is the concatenation of these tokens, resulting in a sequence of length $(2 + K + k_t)$:

$$Seq_{i,k} = \text{Concat}\left([h_{i,k}^{(l-1)}], [h_{\text{context},i}], [\tilde{h}_{j_1}, \ldots, \tilde{h}_{j_K}], [t_1, \ldots, t_{k_t}]\right)$$

This sequence $Seq_{i,k} \in \mathbb{R}^{(2+K+k_t)\times d_{\text{model}}}$ is the input for the dual-stream processor.

Process Step 2: Dual-Stream Processing (Strong Separation)

The model processes $Seq_{i,k}$ using two parallel, specialized Transformer blocks. This separation is crucial: it allows the model to learn observational patterns (correlations) without "polluting" its understanding of true causal mechanisms, and vice-versa.

We enforce a Strong Loss Decomposition: the two streams are trained by different objective functions.

A. The "Correlational Stream" (α^C)

- **Purpose:** To learn "observational" and "correlational" patterns. It models what co-occurs and what is textually consistent, without regard for causal invariance.
- **Methodology:** This is a standard Multi-Head Self-Attention (MHSA) block, MHSA$_C$, followed by an Add & Norm.

$$H_{\text{attncorr}} = \text{MHSA}_C(Seq_{i,k})$$
$$H_{\text{normcorr}} = \text{LayerNorm}(Seq_{i,k} + H_{\text{attncorr}})$$

- **Output:** We extract the representation of the "Self" token (the first token in the sequence):

$$h_{\text{corr},i,k}^{(l)} = H_{\text{normcorr}}[0] \in \mathbb{R}^{d_{\text{model}}}$$

- **Learning (Crucial):** The parameters of this stream (W^C) are **only** updated by the gradient from the $E_{\textbf{local}}$ loss term (the text/graph reconstruction "game").

B. The "Causal Stream" (Mixture-of-Experts)

- **Purpose:** To learn the true, underlying, and *invariant* causal mechanisms.
- **Methodology:** This stream replaces the standard MHSA block with a sparse, context-aware Mixture-of-Experts (MoE).

  1. Context-Aware Router: The router "previews" the sequence to make an intelligent routing decision.

  - A cheap pooled context vector is computed: $c_{i,k} = \text{MeanPool}(Seq_{i,k})$.
  - The routing logits $s \in \mathbb{R}^E$ are computed from the concatenation of the "self" token and this context:

  $$s = \text{Concat}\left(h_{i,k}^{(l-1)}, c_{i,k}\right) W_r$$

  This allows the router to see both the node's identity and the sequence's content (e.g., "this sequence has strong causal words").

2. Softly Sparse Top-2 Gating: We use sparse Top-2 gating with a temperature τ to select $E = 2$ experts.

$$G_{i,k} = \text{TopK}\left(\text{softmax}(s/\tau), k = 2\right)$$

This is a sparse vector (e.g., '[0, 0.65, 0.35]') that allows gradient flow to both experts but is computationally sparse, as only two experts are run. A load-balancing loss ($L_{\text{loadbalance}}$) is added to L_{total} to prevent router collapse.

3. Specialized Experts: These are E (e.g., $E = 3$) different, specialized MHSA blocks (MHSA_e). To **enforce specialization** and prevent expert collapse, we add auxiliary loss heads (Head_e) to each expert, which are only used during training.

For each expert e, we compute its output:

$$H_{\text{norme}} = \text{LayerNorm}(Seq_{i,k} + \text{MHSA}_e(Seq_{i,k}))$$

The auxiliary losses are then computed from this output:

– **Expert 1 (Semantic):** Head_1 is trained on a text-based task.

$$L_{\text{sem}} = \text{Loss}_{\text{BCE}}(\text{Head}_1(H_{\text{norm}}), \text{hascausalkeyword})$$

– **Expert 2 (Structural):** Head_2 is trained to find graph motifs.

$$L_{\text{struct}} = \text{Loss}_{\text{CE}}(\text{Head}_2(H_{\text{norm}}), \text{v-structureexists})$$

– **Expert 3 (Interventional):** Head_3 is trained to be stable.

$$L_{\text{int}} = \text{Variance}(\text{Head}_3(H_{\text{norm}})_{\text{env1}}, \text{Head}_3(H_{\text{norm}})_{\text{env2}})$$

These auxiliary losses are added to the main training objective.

4. Sparse Aggregation: The final output is the gated sum of the expert outputs.

$$H_{\text{aggregated}} = \sum_{e=1}^{E} G_{i,k}[e] \cdot H_{\text{norme}}$$

Since $G_{i,k}$ is sparse, this is a computationally cheap sum of $k = 2$ vectors.

5. Output: We extract the representation of the "Self" token:

$$h^{(l)}_{\text{causal},i,k} = H_{\text{aggregated}}[0] \in \mathbb{R}^{d_{\text{model}}}$$

6. Learning (Crucial): The parameters of this entire Causal Stream (Router W_r, Experts W^e) are **only** updated by the gradients from the causal/invariant losses: $E_{\textbf{prior}}$ and $L_{\textbf{inv}}$.

Process Step 3: Dynamic Gating & Final Layer Update

Finally, we fuse the outputs of the two streams and pass them through the FFN.

- **Dynamic Gating (λ):** We use a learnable "slider" $\lambda_{i,k}$ to decide how much to trust each stream for this specific facet. The gate is computed from the input embedding:

$$\lambda_{i,k} = \sigma\left(h^{(l-1)}_{i,k} W_{\text{gate}} + b_{\text{gate}}\right) \quad (\sigma = \text{sigmoid})$$

- **Stream Fusion:** The two proposals are fused via a weighted average:

$$h_{\text{fused}} = (\lambda_{i,k}) \cdot h^{(l)}_{\text{corr},i,k} + (1 - \lambda_{i,k}) \cdot h^{(l)}_{\text{causal},i,k}$$

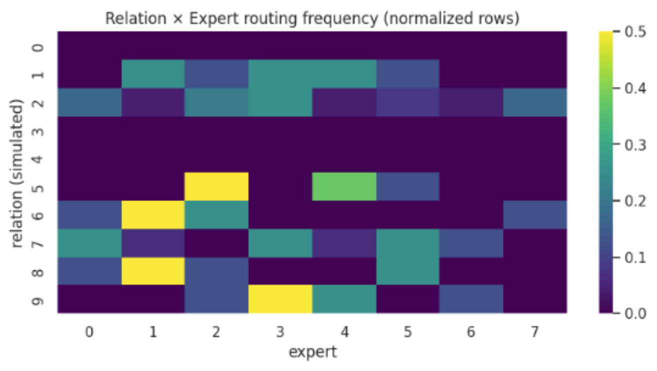
- **Feed-Forward Network (FFN) & Final Update:** This is the second sub-layer of a standard Transformer block. The h_{fused} vector (which is the result of the first sub-layer) is now passed through a shared Feed-Forward Network and normalized with a residual connection.

$$h_{\text{FFN}} = \text{FFN}(h_{\text{fused}}) = \text{ReLU}(h_{\text{fused}} W_1 + b_1) W_2 + b_2$$
$$h^{(l)}_{i,k} = \text{LayerNorm}(h_{\text{fused}} + h_{\text{FFN}})$$

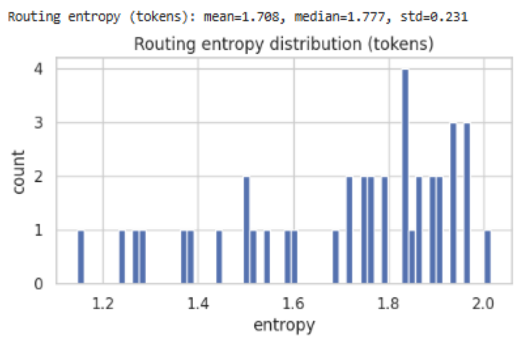
Output of Layer l

The output is the final, refined node embedding tensor $H^{(l)} \in \mathbb{R}^{N \times M \times d_{\text{model}}}$, which is the collection of all $h^{(l)}_{i,k}$ vectors.

This tensor is then passed up to the next layer $(l+1)$, where this entire process repeats, allowing the model to build an ever-deeper and more complex understanding of the graph. The final output of the last layer, $H^{(L)}$, is the ultimate causally-aware embedding used by the Proposer Network (Q_ϕ) and the EBM (E_ψ).



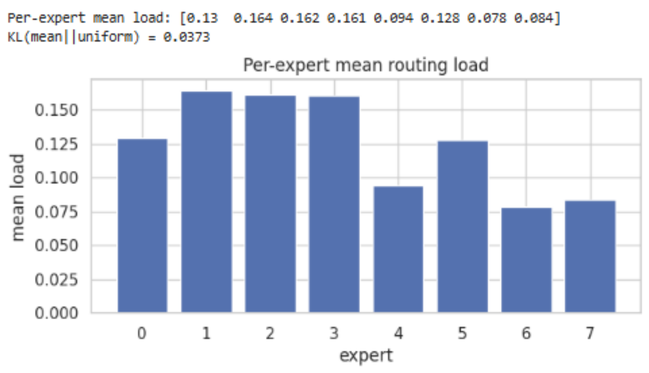
Relation–Expert Routing Map:
This heatmap shows how often tokens from different relations are routed to each expert in the Sparse Causal-MoE block. Rows represent relations and columns represent experts. Brighter cells indicate higher routing frequency. The varied patterns suggest that some relations rely heavily on specific experts, while others distribute their load across several experts.



Routing Entropy Distribution:
This histogram shows the entropy of expert-routing distributions across tokens. Low entropy means the router makes sharp decisions (selecting a few experts strongly), while higher entropy indicates softer, more uniform routing. The spread reflects a mix of confident and uncertain routing behaviours.

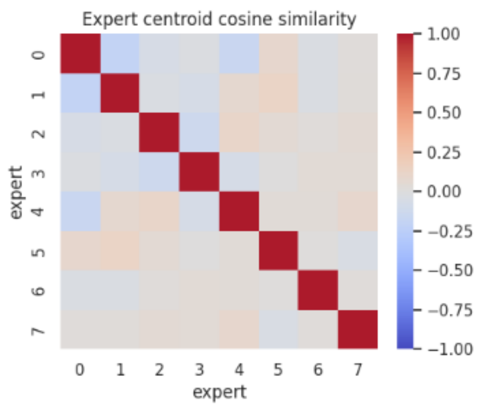
Figure 3.12: Layer–2 MoE diagnostics: expert-routing behaviour across relations (left) and routing uncertainty measured via entropy (right).

The routing matrix shows that relations activate different experts in distinct ways, meaning the MoE block is learning relation-specific specialisation. Meanwhile, the routing entropy histogram indicates a balanced mixture of sharp and diffuse routing, suggesting that some tokens confidently select a small set of experts while others require broader context aggregation. Together, these results imply that the Layer–2 Sparse Causal-MoE successfully captures both specialised and general interaction patterns.



```
Per-expert mean load: [0.13  0.164 0.162 0.161 0.094 0.128 0.078 0.084]
KL(mean||uniform) = 0.0373
```

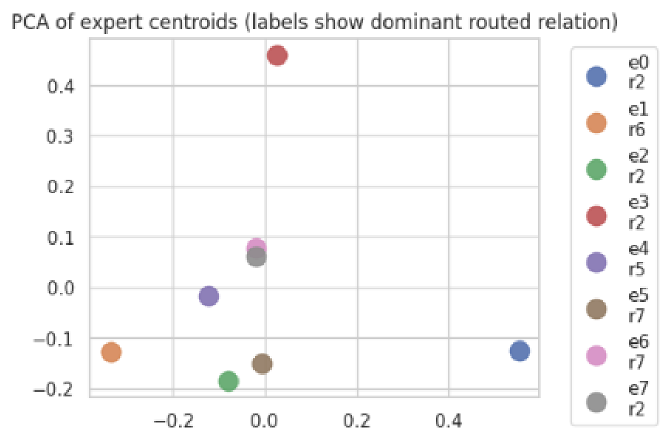
This plot shows the *mean routing load* for each expert in the Sparse Causal-MoE. A perfectly balanced system would assign equal probability mass to all experts. Here, routing is moderately uneven: some experts (e.g., 1–3) carry significantly higher load, while others (e.g., 6–7) are used less often. The accompanying KL divergence value quantifies deviation from a uniform distribution, with higher KL indicating stronger imbalance.

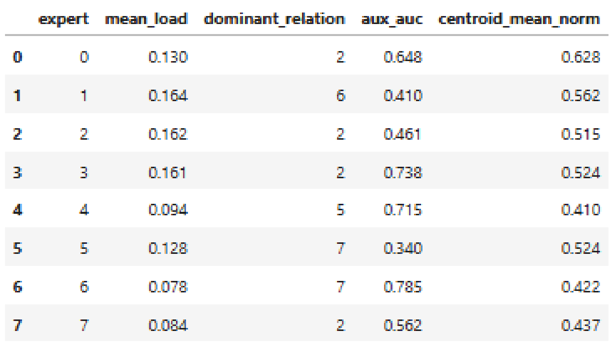


This heatmap visualizes the cosine similarity between the learned *expert centroids*. Diagonal entries are 1.0 (self-similarity), while off-diagonal values measure how distinct expert behaviours are from one another. Near-zero or negative similarities indicate well-differentiated experts, whereas high similarity would suggest redundancy.

Figure 3.13: Layer–2 diagnostic evaluations: (left) distribution of routing load across experts; (right) expert–centroid similarity structure.

Together, these two diagnostics provide a clear picture of how the Sparse Causal-MoE behaves internally. The routing-load analysis reveals that the model does not distribute traffic evenly: a few experts receive most of the tokens, suggesting specialization but also mild imbalance. The centroid similarity map shows that most experts have low mutual similarity, indicating that the MoE is successfully learning diverse functions rather than collapsing into redundant behaviour. Overall, the model demonstrates *specialized but not perfectly balanced* expert usage, which is reasonable for a causal–semantic architecture.





	expert	mean_load	dominant_relation	aux_auc	centroid_mean_norm
0	0	0.130	2	0.648	0.628
1	1	0.164	6	0.410	0.562
2	2	0.162	2	0.461	0.515
3	3	0.161	2	0.738	0.524
4	4	0.094	5	0.715	0.410
5	5	0.128	7	0.340	0.524
6	6	0.078	7	0.785	0.422
7	7	0.084	2	0.562	0.437

PCA of expert centroids: Each point represents the centroid of one expert in the MoE layer after projecting into 2D using PCA. The colour indicates the expert identity, while the label shows its *dominant routed relation*. Closer points imply experts learning similar behaviours, whereas well-separated points indicate specialization.

Expert statistics summary: This table reports, for each expert, its mean routing load, the predominant relation it handles, the auxiliary-head AUC (how useful its representation is), and the norm of its centroid vector. These values together highlight which experts are frequently used, which relations they specialise in, and how strong their internal representations are.

Figure 3.14: Layer–2 expert behaviour diagnostics: (left) specialization and similarity structure among experts; (right) quantitative statistics capturing usage, dominant tasks, and auxiliary signal quality.

The PCA plot shows that some experts form distinct clusters, meaning they have learned specialised roles tied to specific relations, while others sit closer together, suggesting overlap or shared behaviour. The expert summary table reinforces this: experts with higher mean load and higher auxiliary AUC appear more important for the model's reasoning, while low-load experts may be underused or responsible for niche edge-cases. Overall, the MoE layer displays a healthy mixture of *specialization* and *redundancy*, which is desirable in sparse-expert architectures.

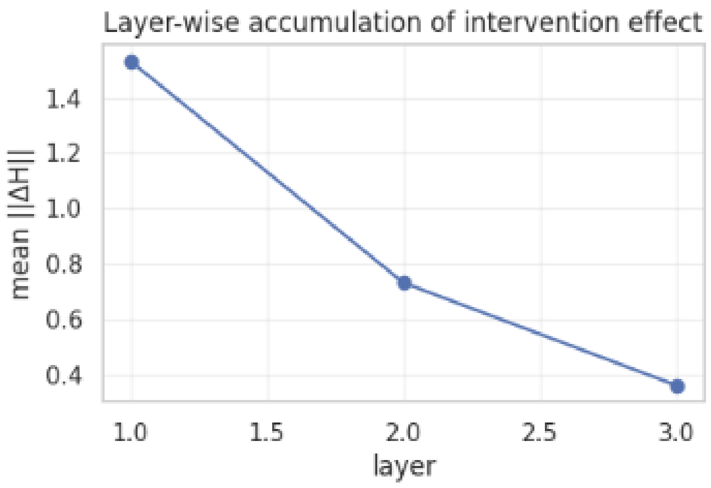


Figure 3.15: **Layer-wise accumulation of intervention effect.** This plot shows how the magnitude of change in node representations ($\|\Delta H\|$) propagates across the stacked Tokenphormer layers. A larger value indicates that the layer is more sensitive to upstream interventions (e.g., perturbing inputs or modifying gating). The decreasing trend from Layer 1 to Layer 3 suggests that deeper layers gradually stabilise the representation and dampen intervention effects.

Layer 1 reacts strongly to interventions because it is closest to raw local context (paths, neighbors). Layers 2 and 3 progressively smooth and consolidate information, reducing the impact of perturbations. This behaviour indicates that the deeper parts of the Causal-MoE Tokenphormer act as stabilising refinement layers rather than amplifying changes, which is desirable for reliable hierarchical causal reasoning.

3.4.2.4 The Amortized Proposer Network (Q_ϕ):

This module is a separate, lightweight decoder network, parameterized by ϕ, that serves a critical computational role: **amortized inference**.

1. Objective and Rationale:

The primary EBM (Energy-Based Model) E_ψ (detailed in Section 4.0) defines a complex energy landscape over the space of all possible graphs. Finding the optimal, low-energy graph A^* by minimizing $E_\psi(A, H)$ using MCMC (Langevin Dynamics) from a random starting point is intractably slow.

The Proposer Network Q_ϕ solves this "cold start" problem. It is trained to be a "fast guesser." In a single, non-iterative forward pass, it takes the final, causally-aware node embeddings $H^{(L)}$ from the encoder and directly proposes a high-quality, sparse "candidate" graph $A^{(0)}$. This $A^{(0)}$ serves as the "warm start" for the subsequent (and now much shorter) MCMC refinement (Section 5.0), drastically reducing the cost of inference.

This proposer is trained not just to find a "vaguely low-energy" region, but to *explicitly imitate* the final, refined output A^* of the EBM, thus learning to approximate the posterior mode in a single step.

2. Architectural Methodology (Polysemy-Aware & Parameter-Efficient):

The Q_ϕ network is a sophisticated decoder that takes the final encoder output $H^{(L)}$ and generates the set of low-rank factors $\{U_r^{(0)}, V_r^{(0)}\}_{r=1}^R$, which in turn define the candidate adjacency tensor $A^{(0)}$. Its design incorporates relation-aware facet pooling and hypernetwork-based parameter sharing.

- **Input:** The final, causally-aware, multi-faceted node embeddings $H^{(L)} \in \mathbb{R}^{N \times M \times d_{\text{out}}}$.
- **Output:** The complete set of initial low-rank factors $\{U_r^{(0)}, V_r^{(0)}\}_{r=1}^R$.

The generation process involves three steps:

Step A: Relation-Aware Facet Pooling (Polysemy-Aware)

To avoid the information bottleneck of simple mean-pooling (which would destroy the polysemous information in $H^{(L)}$), we compute a *relation-specific* summary vector $\bar{h}_i^{(L,r)}$ for each node i and relation r.

- We define a **learnable** query vector $q_r \in \mathbb{R}^{d_q}$ for each relation r.
- For each node i, we compute an attention distribution $\alpha_i^{(r)}$ over its M facets, based on the relation's query. This allows the model to learn, for example, that the 'Causes' relation should attend to the "finance" facet of "Bank," while the 'LocatedIn' relation should attend to the "river" facet.

  $$\alpha_{i,k}^{(r)} = \text{softmax}_k \left((q_r^\top W_Q)(W_K h_{i,k}^{(L)})^T / \sqrt{d_k} \right)$$

  (where $W_Q, W_K \in \mathbb{R}^{d_q \times d_k}$ are **learnable** projection matrices).
- The final relation-aware node embedding $\bar{h}_i^{(L,r)}$ is the weighted sum of its facets, specific to this relation:

  $$\bar{h}_i^{(L,r)} = \sum_{k=1}^{M} \alpha_{i,k}^{(r)} h_{i,k}^{(L)} \quad \in \mathbb{R}^{d_{\text{out}}}$$

Step B: Hypernetwork Factor Generation (Parameter-Efficient)

Instead of learning $2 \times R$ large, independent MLPs (one for each U_r and V_r), we use a single **shared base network** $g(\cdot)$ and condition it using small, **learnable** relation embeddings. This improves parameter efficiency and generalization, as related relations (e.g., 'Causes' and 'Inhibits') can share statistical strength.

- We define $2 \times R$ small, learnable embedding vectors: $e_r^{(U)}, e_r^{(V)} \in \mathbb{R}^{d_e}$ for each relation r.
- The i-th row of the $U_r^{(0)}$ and $V_r^{(0)}$ tables is generated by feeding the relation-aware embedding $\bar{h}_i^{(L,r)}$ and the corresponding relation embedding e_r into the shared network $g(\cdot)$:

$$U_r^{(0)}[i,:] = g\left(\bar{h}_i^{(L,r)};\, e_r^{(U)}\right) \quad \in \mathbb{R}^{d_{\text{rank}}}$$

$$V_r^{(0)}[i,:] = g\left(\bar{h}_i^{(L,r)};\, e_r^{(V)}\right) \quad \in \mathbb{R}^{d_{\text{rank}}}$$

- The conditioning function $g(x;e)$ is itself a **learnable** network, implemented, for example, using a **FiLM (Feature-wise Linear Modulation)** layer:

$$g(x;e) = W_2\,\sigma\big((W_1x + b_1) \odot \gamma(e) + \beta(e)\big) + b_2$$

  where $\gamma(e)$ and $\beta(e)$ are small MLPs that generate scaling and shifting parameters from the relation embedding e. The parameters of g, γ, β, and the embeddings e_r are all part of ϕ.

Step C: Final Candidate Graph Construction
We explicitly define the candidate adjacency matrix $A_r^{(0)}$ by reconstructing it from the generated low-rank factors.

$$A_r^{(0)}(i,j) = \sigma\left(U_r^{(0)}[i,:]^\top V_r^{(0)}[j,:] + b_r\right)$$

where b_r is a learnable, per-relation bias scalar. This $A_r^{(0)}(i,j)$ value is interpreted as the proposed probability for the edge $(i \to j, r)$. The final candidate graph $G_{\text{candidate}} = (H^{(L)}, A^{(0)})$ is then passed to the EBM.

3. Training Mechanism (How Q_ϕ Learns):
The Proposer Network Q_ϕ (all parameters ϕ from Steps A and B) is trained jointly with the EBM E_ψ. To ensure stable training, we use a composite loss function that combines "teacher-forcing" (learning from the refined graph A^*) with energy minimization and sparsity regularization.

The total loss for the proposer, $L_{\text{proposer}}^{\text{total}}$, is a weighted sum of three components:

$$L_{\text{proposer}}^{\text{total}}(\phi) = \alpha_1 L_{\text{energy}}(\phi) + \alpha_2 L_{\text{distill}}(\phi) + \alpha_3 L_{\text{sparse}}(\phi)$$

- **1. Energy Loss with EMA Critic (L_{energy}):**
  This term trains Q_ϕ to find "vaguely low-energy" regions. To prevent Q_ϕ from "chasing a moving target" (as E_ψ is also training), we train Q_ϕ against a *smoother, slower-moving EMA (Exponential Moving Average) copy* of the EBM, $\tilde{E}_\psi$.

  Let $\tilde{\psi}$ be the EMA parameters of the EBM: $\tilde{\psi} \leftarrow \tau\tilde{\psi} + (1-\tau)\psi$. The loss is the expected energy of the proposed graph, as scored by this **stable critic** $\tilde{E}_\psi$:

$$L_{\text{energy}}(\phi) = \mathbb{E}_{H^{(L)} \sim \text{Data}}\left[\tilde{E}_\psi(H^{(L)}, Q_\phi(H^{(L)}))\right]$$

  The gradient from this loss, $\nabla_\phi \tilde{E}_\psi$, updates Q_ϕ to propose graphs that the stable critic "likes" (assigns low energy to).

- **2. Distillation Loss (Teacher-Forcing, L_{distill}):**
  This is the most important "how it learns" term. It forces Q_ϕ to *explicitly imitate* the final, refined graph A^* that is produced by the full MCMC/Langevin refinement (detailed in Section 5.2). This teaches Q_ϕ to approximate the posterior mode.

Let $A^* = \text{MCMCRefine}(A^{(0)})$. We use a 'stopgrad' operation on A^* so that we are only training Q_ϕ to *match* A^*, not training A^* to match Q_ϕ. The loss is an L1 divergence:

$$L_{\text{distill}}(\phi) = \mathbb{E}\left[\left\|A^{(0)} - \text{stopgrad}(A^*)\right\|_1\right]$$

The gradient from this loss, $\nabla_\phi \|A^{(0)} - A^*\|_1$, directly teaches the Q_ϕ parameters (g, γ, β, e_r, etc.) to produce factors that will reconstruct to a graph $A^{(0)}$ that is as close as possible to the "correct" refined answer A^*.

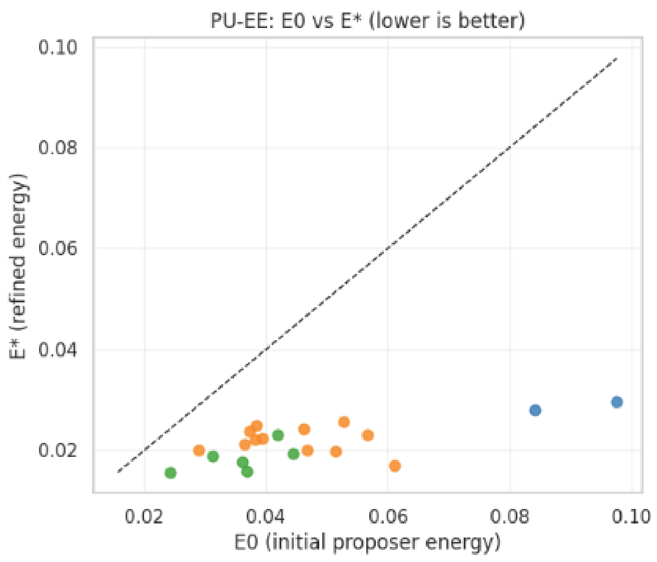
- **3. Explicit Sparsity Loss (L_{sparse}):**
  This term "offloads" the sparsity burden from the EBM to the proposer. We directly penalize the proposer for outputting a dense graph.

  The loss is an L1 penalty on the *output probabilities* of the proposer:

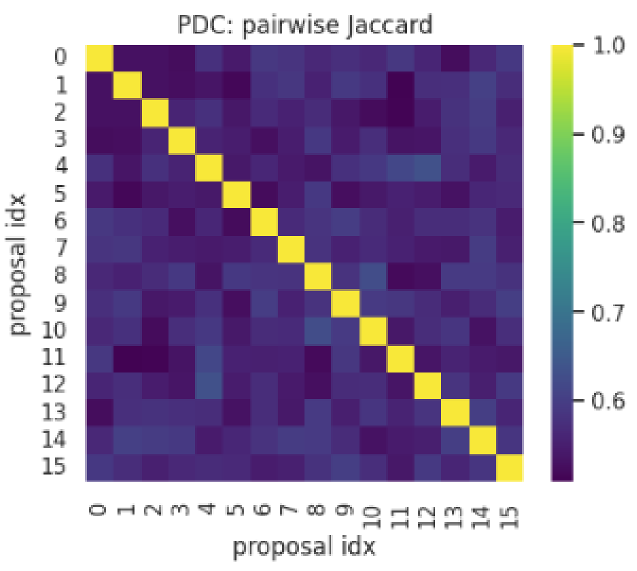
$$L_{\text{sparse}}(\phi) = \beta \cdot \sum_{r,i,j} \left|A_r^{(0)}(i,j)\right|$$

  This encourages Q_ϕ to propose an $A^{(0)}$ that is *already sparse*, so the MCMC refinement (Section 5.0) can start from a much more realistic and high-quality "warm start."

By minimizing this composite $L_{\text{proposer}}^{\text{total}}$ loss, the Proposer Network Q_ϕ learns to generate a high-quality, sparse, and low-energy graph in a single forward pass, making the entire architecture computationally tractable.



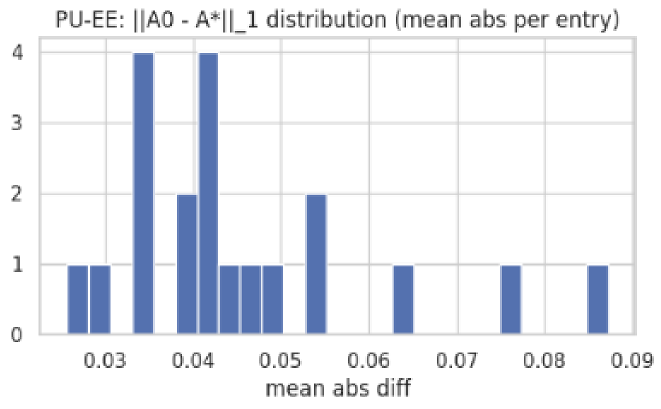
PU–EE: This scatter plot compares the *initial proposer energy* E_0 with the *refined energy* E obtained after MCMC refinement. Points below the diagonal line indicate that refinement improves proposal quality (lower energy), meaning the sampler found a better adjacency structure than what the proposer initially predicted.



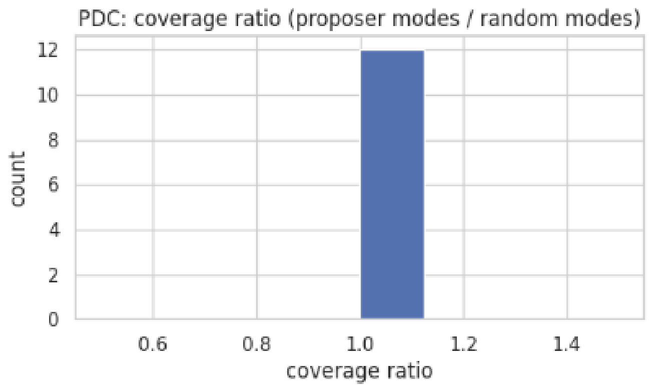
PDC: Pairwise Jaccard similarity between a batch of proposals generated by Q_ϕ. Lower off-diagonal similarities indicate that the proposer is generating *diverse* candidate graphs. Strong diagonal values are expected, while varied off-diagonal values suggest healthy diversity.

Figure 3.16: Diagnostics for the amortized proposer Q_{phi}: (left) energy improvement after refinement; (right) diversity of sampled adjacency proposals.

The PU–EE scatter shows that most refined samples have lower energy than the initial ones, implying that the proposer generates reasonable structures but refinement still finds better solutions. The Jaccard heatmap indicates that proposals are not collapsing to a single mode; off-diagonal similarity is moderate to low. Together, these metrics suggest that Q_ϕ is *both* (1) producing meaningful initial guesses and (2) maintaining diversity, which is essential for effective amortized inference in the CausGT-HS pipeline.



PU–EE absolute difference distribution. This histogram shows the mean absolute ℓ_1 difference between the proposer's initial adjacency prediction (A_0) and the refined sample $(A^\star)$ produced by Langevin MCMC. Smaller values mean the proposer already produces samples close to the energy-refined optimum, while larger values indicate corrections were necessary.



PDC coverage ratio. This plot compares the number of distinct structural "modes" discovered by the proposer against those obtained from random sampling. A coverage ratio near 1.0 means the proposer explores at least as many modes as random sampling, while ratios > 1 indicate superior structural coverage.

Figure 3.17: Additional diagnostics for the amortized proposer: (left) how far the initial proposal A_0 is from the refined $A^\star$; (right) how well the proposer explores diverse structural modes compared to a random baseline.

The PU–EE error histogram shows that most proposer outputs require only small corrections from the energy model, suggesting that the amortized proposer learns a good approximation of the refined distribution. The PDC coverage ratio close to 1.0 indicates that the proposer maintains diverse exploration instead of collapsing to a narrow subset of relation patterns. Combined, these results imply that the proposer is both *accurate* (small refinement gap) and *diverse* (good mode coverage), which are crucial for stable joint EBM training.

3.4.2.5 Self-Supervised Training: The Causal Energy Curriculum

This section details the core innovation of the CausGT-HS-EBM. The entire system is trained as an Energy-Based Model (EBM). The EBM is not a single component, but rather the overarching "training philosophy" or "judge" for the entire system.
The EBM's goal is to learn a "global energy function" $E(G)$, parameterized by ψ, which assigns a single, scalar "cost" or "plausibility" score to any proposed graph G. A low-energy graph is considered "good" and plausible (e.g., a ball at the bottom of a valley), while a high-energy graph is "bad" and implausible (e.g., a ball on a hilltop).
The training objective for all components (the Encoder f_θ, the Proposer Q_ϕ, and the Energy Function E_ψ) is to work together to find a graph G that minimizes this total energy.

1. **The Global Energy Function $E_\psi(G)$**
   This section details the core of the `CausGT-HS-EBM`. The entire system is judged by a "global energy function," parameterized by ψ, which assigns a single, scalar "cost" or "plausibility" score to any proposed graph G. A low-energy graph is considered plausible, while a high-energy graph is implausible. The model's objective is to find a graph G that minimizes this total energy.

1.1 The Hierarchical Energy Function (E_{total})

Our architecture is hierarchical, so our energy function is also hierarchical. The total energy E_{total} is a composite sum of the coarse-grained energy, the sum of all fine-grained energies, and a consistency term to couple them.

$$E_{\text{total}}(G) = \lambda_{\text{coarse}} \cdot E_{\psi,\text{coarse}}(G_{\text{coarse}}) + \lambda_{\text{fine}} \cdot \sum_{k=1}^{K} E_{\psi,k}(G_k) + \lambda_{\text{cons}} \cdot E_{\text{consistency}}(G)$$

where λ are hyperparameter weights that balance the contribution of each component.

The Coarse-Fine Consistency Energy ($E_{\text{consistency}}$)

This is a critical energy term that couples the hierarchical levels, forcing the "highway map" (A_{coarse}) and the "local roads" (A) to be mutually consistent.

(a) **Differentiable Aggregation:** We first compute an "implicit" coarse graph, $\hat{A}_{\text{coarse}}$, by differentiably aggregating the fine-grained graph A using the soft assignment matrix C (from Section 3.2.1).

$$\hat{A}_{\text{coarse},r} = C^T A_r C$$

(b) **Consistency Loss:** $E_{\text{consistency}}$ is the L2 (Frobenius) norm between the *learned* coarse graph A_{coarse} (from $E_{\psi,\text{coarse}}$) and this *aggregated* fine-grained graph $\hat{A}_{\text{coarse}}$.

$$E_{\text{consistency}} = \sum_{r \in R} \left\| A_{\text{coarse},r} - \hat{A}_{\text{coarse},r} \right\|_F^2$$

This term penalizes any divergence between the "big picture" map and the sum of its "local parts."

1.2. The Coarse-Grained Energy Function ($E_{\psi,\text{coarse}}$)

This is the "Chief Planner" EBM, parameterized by ψ_{coarse}. It is a separate, dense model that scores the small $K \times K$ "highway map" A_{coarse}.

$$E_{\psi,\text{coarse}}(G_{\text{coarse}}) = E_{\text{global}}(A_{\text{coarse}}) + E_{\text{prior,coarse}}(A_{\text{coarse}}, \mathcal{P}_{\text{coarse}})$$

- **$E_{\text{global}}(A_{\text{coarse}})$: (The "Global Coherence" Term)**
  This term enforces the fundamental "rules" of a valid causal graph.
  - **Acyclicity Constraint:** A differentiable penalty using the **NOTEARS** trace-exponential constraint. For each causal relation slice A_r of A_{coarse}:
  $$h(A_r) = \text{tr}(e^{A_r \odot A_r}) - K$$
  This function $h(A_r) = 0$ iff A_r is a DAG.
  - **Sparsity Constraint:** A standard L1 penalty to encourage sparsity:
  $$L_1(A_r) = \|A_r\|_1 = \sum_{i,j} |A_r(i,j)|$$

  The final term is: $E_{\text{global}} = \sum_{r \in \text{CausalRels}} (\lambda_{\text{dag}} \cdot h(A_r) + \lambda_{\text{sparse}} \cdot L_1(A_r))$.
- **$E_{\text{prior,coarse}}$: (The "Highway Rulebook")**
  This term scores the consistency of A_{coarse} against the aggregated, community-level "super-rules" $\mathcal{P}_{\text{coarse}}$.

1.3. The Fine-Grained Energy Function ($E_{\psi,k}$)

This is the main "Local Engineer" EBM. It is instantiated K times in parallel (once for each community k). It scores the detailed, node-to-node graph G_k within that community.

$$E_{\psi,k}(G_k) = E_{\text{correlational}}(G_k) + E_{\text{causal}}(G_k)$$

1.3.1. $E_{\text{correlational}}$ (Correlational/Textual Energy)

- **Goal:** To score consistency with observational, superficial evidence (the text).
- **Methodology:** This is the E_{local} (Masked Language/Link Prediction) task. The EBM parameters ψ_k include decoder heads that compute these probabilities:

$$E_{\text{correlational}} = \lambda_{\text{mlm}} \cdot \left(-\sum_i \log P(t_{\text{masked}} | Seq_{i,k}; \psi_k) \right) + \lambda_{\text{mlp}} \cdot \left(-\sum_{i,j} \log P(A_{ij} | H_i, H_j; \psi_k) \right)$$

  This term ensures the model's representations are grounded in the text.

1.3.2. E_{causal} (Causal Structure Energy)

- **Goal:** To score the graph's consistency with the deep, invariant causal structure.
- **Methodology:** This is the sum of our sophisticated Causal Curriculum and a local regularizer.

$$E_{\text{causal}} = E_{\text{prior}}(A_k, \mathcal{P}_{\text{rich},k}) + E_{\text{localstruct}}(A_k)$$

1.3.3. Detailed Breakdown of E_{prior} (The "Causal Rulebook")

This component is a collection of loss functions that judge the graph A against the "answer key," $\mathcal{P}_{\text{rich}}$.

Note on Adjacency Matrix A: The adjacency matrix $A(i,j,r)$ being scored here is the differentiable output of our low-rank factorization (Section 3.4):

$$A(i,j,r) = \sigma\left(U_r[i,:]^\top V_r[j,:] + b_r\right)$$

The energy "penalties" are applied to this output.

$$E_{\text{prior}} = \lambda_{\text{dir}} E_{\text{direct}} + \lambda_{\text{med}} E_{\text{mediation}} + \lambda_{\text{con}} E_{\text{confound}}$$

- **A. E_{direct} (Direct Link Distillation)**
  This is a heteroskedastic (uncertainty-weighted) L2 loss. For all '(i, j, 'DIRECT')' rules:

  $$E_{\text{direct}} = \sum_{(i,j)\in\mathcal{P}_{\text{direct}}} \underbrace{\frac{1}{\sigma_{ij}^2}}_{\text{Confidence Weight}} \cdot \left(\underbrace{A(i,j,r_c)}_{\text{Model's Score}} - \underbrace{\text{score}_{ij}}_{\text{Rulebook Score}} \right)^2$$

  This forces the model to match high-confidence priors (σ^2 is small, penalty is high) and allows it to deviate from low-confidence priors (σ^2 is large, penalty is low).

- **B. $E_{\text{mediation}}$ (Smooth Mediation Penalty)**
  We use a smooth, product-based (product t-norm) penalty. The "satisfaction" of a mediation rule ($i \to k \to j$ and $NOT\ i \to j$) is:

  $$\text{medsatisfaction}(i,j,k) = (1 - A(i,j,r_c)) \cdot A(i,k,r_c) \cdot A(k,j,r_c)$$

  The energy (penalty) is 1 minus this satisfaction, weighted by the rule's uncertainty σ_{ijk}^2:

  $$E_{\text{mediation}} = \sum_{(i,j,k)\in\mathcal{P}_{\text{med}}} \frac{1}{\sigma_{ijk}^2} (1 - \text{medsatisfaction}(i,j,k))$$

- **C. E_{confound} (Smooth Confounding Penalty)**
  Similarly, the "satisfaction" of a confounding rule ($k \to i$, $k \to j$, and $NOT\ i \to j$) is:

  $$\text{confsatisfaction}(i,j,k) = (1 - A(i,j,r_c)) \cdot A(k,i,r_c) \cdot A(k,j,r_c)$$

  The energy (penalty) is 1 minus this satisfaction, also weighted by uncertainty:

  $$E_{\text{confound}} = \sum_{(i,j,k)\in\mathcal{P}_{\text{conf}}} \frac{1}{\sigma_{ijk}^2} (1 - \text{confsatisfaction}(i,j,k))$$

1.3.4. Detailed Breakdown of $E_{\text{localstruct}}$ (Local Structural Regularizer)

This is a penalty term to "nudge" the fine-grained graphs toward a reasonable structure.

- **Goal:** To prevent the fine-grained graphs A_k from becoming overly dense or complex, without incurring the cost of a full $N \times N$ DAG constraint.

- **Method:** We add a **degree distribution penalty**. We encourage the out-degree of each node i in the causal graph to be close to some small, expected mean out-degree μ (e.g., $\mu = 2$). The out-degree for a node i (in community k) is the differentiable sum:

$$\text{deg}_k^+(i) = \sum_{j \in G_k} A_k(i, j, r_{\text{causes}})$$

The energy (penalty) is the squared deviation from the mean μ:

$$E_{\text{localstruct}} = \lambda_{\text{deg}} \cdot \sum_{i \in G_k} \left(\text{deg}_k^+(i) - \mu\right)^2$$

2. **The Total Loss Function: Contrastive Energy Minimization**
The entire architecture, comprising the encoder (f_θ), the proposer (Q_ϕ), and the energy function (E_ψ), is trained jointly (end-to-end) by minimizing a composite, multi-part loss function.

1. Automatic Loss Balancing (Uncertainty Weighting)

A key challenge in this architecture is the "hyperparameter hell" of manually tuning the relative weights (e.g., $\lambda_{\text{contrastive}}, \lambda_{\text{proposer}}, \ldots$) of the many disparate loss components. To solve this, we employ a **homoscedastic uncertainty-based weighting** mechanism.

- **Methodology:** We treat each of the N major loss components L_i as a task with its own **learnable** noise parameter σ_i. The model learns to down-weight "noisy" or high-variance tasks (by increasing their σ_i) and up-weight "confident" or stable tasks (by decreasing their σ_i).
- **Learnable Parameters:** The **learnable** parameters are the N noise scalars, $\{\sigma_i\}_{i \in \mathcal{L}}$. These are learned via standard backpropagation by minimizing L_{total}.
- **Total Loss (L_{total}):** The final training objective is to minimize this auto-balancing loss function, where $\mathcal{L}$ is the set of all our loss components:

$$L_{\text{total}} = \sum_{i \in \mathcal{L}} \left(\frac{1}{2\sigma_i^2} L_i + \log \sigma_i \right)$$

The set of losses $\mathcal{L}$ is defined as:

$$\mathcal{L} = \{L_{\text{contrastive}}, L_{\text{proposer}}^{\text{total}}, L_{\text{inv}}, L_{\text{aux}}\}$$

This formulation is more robust than a fixed weighted sum and replaces the manual λ hyperparameters for these top-level terms. We now detail each L_i component.

2. $L_{\text{contrastive}}$ (The EBM Teacher)

This is the primary training objective for the **Energy Function parameters** ψ. It is a contrastive divergence loss (InfoNCE) that teaches E_ψ to assign low energy to "correct" graphs and high energy to "incorrect" graphs.

- **What is Learnable:** The parameters ψ of the Energy Function E_ψ (Section 4.1).
- **How it Learns:** By minimizing this loss, E_ψ learns to "pull down" the energy of G_{pos} while "pushing up" the energy of all $G_{\text{neg},m}$.

- **Positive Sample ($G_{\textsf{pos}}$) (Improvement A):** To train E_ψ to find the "true" solution, we do not use the Proposer's raw guess $A^{(0)}$. Instead, the "positive" sample is the **final, refined graph** A^* (the output of the MCMC/Langevin refinement from Section 5.2). This forces E_ψ to learn an energy landscape where the minimum (the "valley") is centered on the final solution, not the initial guess.

$$G_{\textsf{pos}} = (H^{(L)}, A^*) \quad \text{where} \quad A^* = \text{stopgradient}(\text{MCMCRefine}(A^{(0)}))$$

- **Negative Samples ($G_{\textsf{neg},m}$) (Improvement D):** We use a **diversified curriculum** for negative sampling to prevent the EBM from only learning about the local neighborhood around A^*.
  - **Stage 1 (Early Epochs):** We use "easy" negatives, such as uniformly random graphs, $A^{(0)}$ from other minibatch items, or perturbations of $A^{(0)}$. This helps E_ψ learn a robust, coarse energy landscape.
  - **Stage 2 (Later Epochs):** We switch to "hard" negatives, generated via **short-run Stochastic Gradient Langevin Dynamics (SGLD) ascent**. We start from A^* and run a few steps t of noisy gradient *ascent* (note the + sign) to find plausible-but-wrong graphs in the low-energy valley.

$$A_{t+1} = \text{Proj}_{[0,1]}\big(A_t + \eta \nabla_A E_\psi(A_t, H) + \sqrt{2\eta}\xi_t\big)$$

- **Loss Function (InfoNCE):** The contrastive loss is the negative log-likelihood of the positive sample over all samples, with a temperature τ:

$$L_{\textsf{contrastive}} = -\log \frac{e^{-E_\psi(G_{\textsf{pos}})/\tau}}{e^{-E_\psi(G_{\textsf{pos}})/\tau} + \sum_{m=1}^{M} e^{-E_\psi(G_{\textsf{neg},m})/\tau}}$$

3. $L^{\textsf{total}}_{\textsf{proposer}}$ (The Amortization Teacher)

This is the composite loss function that trains the **Proposer Network parameters** ϕ. It is composed of three terms designed to make Q_ϕ a stable and accurate "fast guesser."

$$L^{\textsf{total}}_{\textsf{proposer}}(\phi) = \alpha_1 L_{\textsf{energy}} + \alpha_2 L_{\textsf{distill}} + \alpha_3 L_{\textsf{sparse}}$$

(Note: these α_i weights *can* be manually tuned or, more robustly, be absorbed into the main auto-balancing $L_{\textsf{total}}$ by treating $L_{\textsf{energy}}, L_{\textsf{distill}}$, and $L_{\textsf{sparse}}$ as their own tasks L_i).

- **1. $L_{\textsf{energy}}$ (Energy Minimization with EMA Critic):** This term trains Q_ϕ to find "vaguely low-energy" regions. To prevent Q_ϕ from "chasing a moving target" (as E_ψ is also training), it is trained against a **stable, slow-moving EMA (Exponential Moving Average) copy** of the EBM, $\tilde{E}_\psi$.

$$L_{\textsf{energy}}(\phi) = \mathbb{E}_{H^{(L)}}\left[\tilde{E}_\psi(H^{(L)}, Q_\phi(H^{(L)}))\right]$$

  where the critic's parameters $\tilde{\psi}$ are updated via: $\tilde{\psi} \leftarrow \tau_{\textsf{ema}}\tilde{\psi} + (1 - \tau_{\textsf{ema}})\psi$.

- **2. $L_{\textsf{distill}}$ (Teacher-Forced Amortization):** This is the "teacher-forcing" term (Improvement A). It explicitly trains Q_ϕ to imitate the final, refined MCMC output, A^*. The gradient ∇_ϕ teaches Q_ϕ to produce an $A^{(0)}$ that is as close as possible to the "correct answer" A^*.

$$L_{\textsf{distill}}(\phi) = \mathbb{E}\left[\left\|A^{(0)} - \text{stopgradient}(A^*)\right\|_1\right]$$

- **3. L_{sparse} (Explicit Sparsity):** This offloads the sparsity burden to the cheap proposer, penalizing it for outputting a dense graph.

$$L_{\text{sparse}}(\phi) = \beta \cdot \sum_{r,i,j} \left| A_r^{(0)}(i,j) \right|$$

4. L_{inv} (The Invariance Robustness Teacher)

This loss trains the model to be **causally invariant**. It is the primary "teacher" for the **Causal-MoE Stream** parameters (part of θ) and the E_ψ parameters. We use a strengthened, composite invariance loss.

- **Process (IRM):** We generate two environments, $\mathcal{E}_1$ (original text) and $\mathcal{E}_2$ (LLM-augmented text). We run the full model to get two graphs (G_1, G_2) and, crucially, two sets of causal-stream node embeddings (Z_1, Z_2) from the output of the Causal-MoE stream (Section 3.3.2.B).
- **Composite Invariance Loss:**

$$L_{\text{inv}} = \gamma_1 L_{\text{inv,energy}} + \gamma_2 L_{\text{inv,repr}}$$

  - **1. $L_{\text{inv,energy}}$ (Energy-Level Invariance):** This is the weak, scalar-level loss. It forces the *final energy score* to be the same for both environments, forcing E_ψ to learn an invariant landscape.

$$L_{\text{inv,energy}} = \text{variance}\left(E_\psi(G_1), E_\psi(G_2)\right)$$

  - **2. $L_{\text{inv,repr}}$ (Representation-Level Invariance):** This is a much stronger, denser loss. It forces the *causal-stream embeddings themselves* to be invariant. We penalize the L2 distance between the embeddings of the same node i under the two different environments.

$$L_{\text{inv,repr}} = \mathbb{E}_i \left[\|Z_{1,i} - Z_{2,i}\|_2^2 \right]$$

    This provides a much denser gradient signal to the Causal-MoE stream, forcing it to learn a "canonical" representation of causal facts, regardless of superficial text.

5. L_{aux} (Decoupled Auxiliary Regularizers)

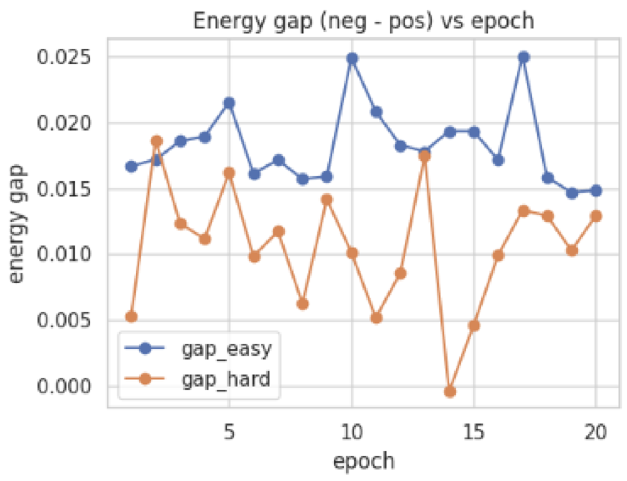
This is a collection of smaller losses that keep the model's components stable and specialized. We decouple these losses to prevent them from interfering with the main energy-based learning.

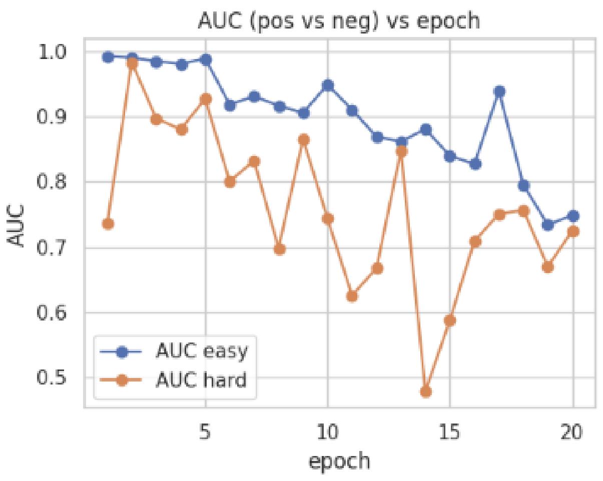
$$L_{\text{aux}} = L_{\text{moespec}} + L_{\text{loadbalance}} + L_{\text{facetsparsity}}$$

- **L_{moespec} (MoE Specialization):** This is the sum of the auxiliary losses from the expert-specialization heads $(L_{\text{sem}}, L_{\text{struct}}, L_{\text{int}})$ from Section 3.3.2.B.3.

  Decoupling: To prevent these "side-tasks" from "polluting" the main encoder's representations, we (a) apply these loss heads **only to the final 1-2 layers** of the encoder, and (b) may use **stop-gradients** so the gradients from L_{moespec} *only* update the expert-specific parameters (e.g., W_Q^e, W_K^e) and not the shared encoder trunk.
- **$L_{\text{loadbalance}}$ (MoE Routing):** This is the standard MoE loss that penalizes the router for not distributing work evenly, ensuring all experts are utilized.

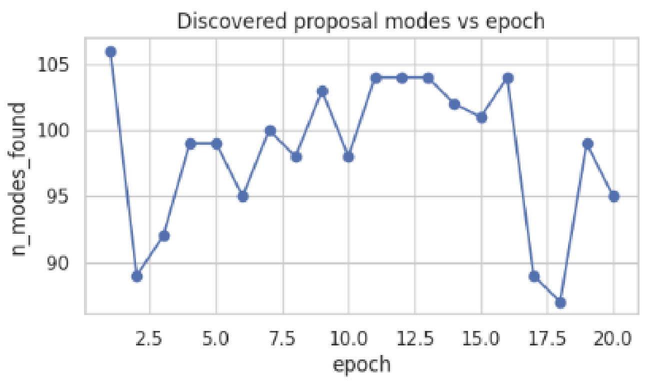
- $L_{\textbf{coltfacetsparsity}}$ **(Facet Gating):** This is the L1 penalty on the facet *gating parameters* g_i. The weight for this loss, $\lambda_{\text{facet}}(t)$, is **annealed** (starts at 0 and grows over time). This allows the model to discover its facets first before being forced to make them sparse.

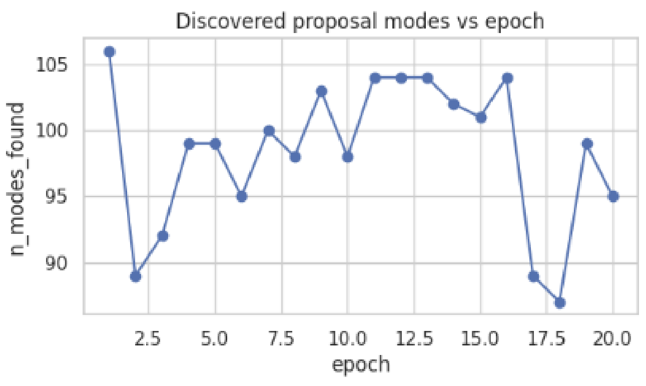




Energy Gap Across Epochs: The gap between negative–sample energies and positive (refined) energies. Larger gaps indicate better separation by the energy model. **AUC Across Epochs:** ROC–AUC for distinguishing positive proposals from negative ones. Higher values imply more reliable ranking of causal proposals by the EBM.

Figure 3.18: Self-supervised Causal Energy Curriculum diagnostics. (Left) evolution of energy gaps for easy and hard negatives; (Right) AUC curves tracking how consistently the EBM separates high-quality proposals from noise.



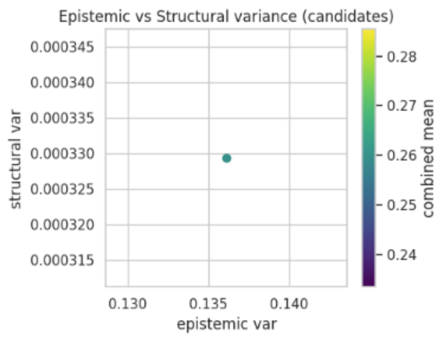


Discovered proposal modes vs epoch: This plot shows how many distinct proposal "modes" the amortized proposer discovers at each training step. Higher values indicate better exploration of the space of causal graphs, while drops suggest temporary collapse or unstable exploration.

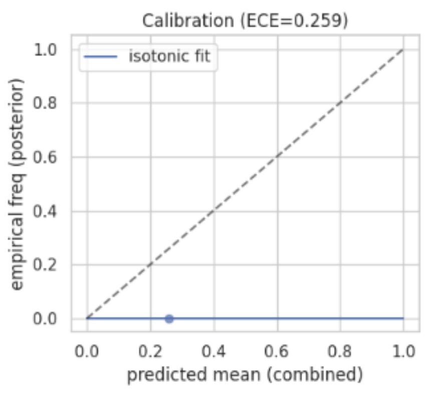
Mode energy distribution (epoch 20): Each box corresponds to a discovered proposal mode, showing the distribution of positive energies E_{pos} assigned by the hierarchical EBM. Lower energies correspond to more plausible causal structures, while higher energies indicate unlikely proposals.

Figure 3.19: Diagnostics for the amortized proposer under the causal energy curriculum: (left) evolution of discovered proposal modes; (right) energy distribution across modes at the final epoch.

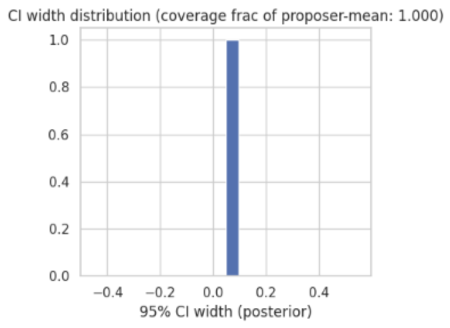
Together, these results show that the proposer initially explores widely but gradually stabilizes on a consistent set of modes. The final distribution reveals a clear separation between low-energy (valid) causal modes and high-energy (invalid) ones, indicating that the causal energy curriculum successfully guides the model toward more plausible causal graphs while still maintaining diversity.



(a) Epistemic vs Structural Variance.
Epistemic variance comes from MC-dropout ensembles; structural variance reflects posterior uncertainty from SGLD refinement. High-epistemic / low-structural cases indicate proposer uncertainty but confident posterior.



(b) Calibration Curve (ECE = 0.259).
Deviation from the diagonal shows moderate miscalibration; the model tends to underestimate uncertainty.



(c) 95% CI Width Distribution.
Extremely narrow CIs show a highly concentrated posterior. Coverage fraction 1.00 indicates strong consistency between proposer and posterior.

Figure 3.20: **Uncertainty diagnostics for final causal inference.** Side-by-side visualization of epistemic vs structural uncertainty, calibration behaviour, and posterior CI concentration.

3.4.2.6 Final Output & Inference (Reasoning with Uncertainty)

After training (Sections 4.0–4.2) is complete, the system consists of three specialized components:

1. **Encoder (f_θ):** An L-layer Tokenphormer that reads raw text and graph-derived inputs and produces final, causally-aware node embeddings $H^{(L)}$.
2. **Proposer (Q_ϕ):** A fast, amortized network that takes $H^{(L)}$ and outputs a high-quality candidate graph in the form of low-rank factors $U^{(0)}, V^{(0)}$.
3. **Energy Function (E_ψ):** A critic that has learned the "physics" of a good causal graph and assigns a plausibility score (energy) to any graph G.

Inference is *not* a single forward pass. It is a three-step probabilistic procedure that combines these components to produce a robust, uncertainty-aware answer, rather than a single point estimate.

Uncertainty Types. The pipeline models two distinct forms of uncertainty:

- **Epistemic Uncertainty:** Uncertainty in the model parameters (θ, ϕ), i.e., "How confident is the model in its own weights?" This is quantified using Monte Carlo (MC) Dropout.
- **Structural (Model) Uncertainty:** Uncertainty about the graph structure itself, i.e., "Is there only one good graph, or are there multiple low-energy graphs that are all plausible?" This is quantified via MCMC sampling from the energy landscape.

Step 1: Hierarchical MC Dropout (Fast Epistemic Uncertainty)

- **Goal:** Obtain an initial estimate of the graph $A^{(0)}$ and quantify epistemic uncertainty arising from both the Encoder f_θ and the Proposer Q_ϕ.
- **Methodology:** Use hierarchical full-pipeline MC Dropout, with dropout active in both f_θ and Q_ϕ.
- **Process:**

  1. **Encoder Ensemble.** Run the Encoder f_θ N_{enc} times with dropout enabled, producing an ensemble of node embedding tensors:
  $$\{H_1^{(L)}, H_2^{(L)}, \dots, H_{N_{\text{enc}}}^{(L)}\}.$$
  2. **Proposer Ensemble.** For each $H_s^{(L)}$, run the Proposer Q_ϕ N_{prop} times with dropout enabled, yielding:
  $$\{(U_n, V_n)\}_{n=1}^{N_{\text{mc}}}, \quad \text{where } N_{\text{mc}} = N_{\text{enc}} \cdot N_{\text{prop}}.$$

- **Output 1: Mean Factors (Warm Start).** The warm-start factors are the averages of the sampled factors:
$$U_{\text{mean}}^{(0)} = \frac{1}{N_{\text{mc}}} \sum_{n=1}^{N_{\text{mc}}} U_n, \quad V_{\text{mean}}^{(0)} = \frac{1}{N_{\text{mc}}} \sum_{n=1}^{N_{\text{mc}}} V_n.$$
These serve as the initialization for the MCMC sampling in Step 5.2.
- **Output 2: Per-Edge Epistemic Uncertainty.** Instead of computing variances for all N^2 edges, define a smaller *candidate set* $\mathcal{C}$ of edges of interest (for example, edges with mean probability $A_{\text{mean}}^{(0)}(i,j,r) > 0.1$, or edges incident to queried nodes). For each $(i,j,r) \in \mathcal{C}$:

  1. Compute the N_{mc} edge probabilities:
  $$A_n(i,j,r) = \sigma\big(U_n[i,:]^\top V_n[j,:] + b_r\big), \quad n = 1, \dots, N_{\text{mc}}.$$
  2. Compute the epistemic mean and variance:
  $$\mu_{ijr}^{\text{epistemic}} = \frac{1}{N_{\text{mc}}} \sum_{n=1}^{N_{\text{mc}}} A_n(i,j,r),$$
  $$\Sigma_{A^{(0)}}(i,j,r) = \operatorname{Var}\big(\{A_n(i,j,r)\}_{n=1}^{N_{\text{mc}}}\big).$$

  The variance $\Sigma_{A^{(0)}}(i,j,r)$ measures the parameter (epistemic) uncertainty for edge (i,j,r).

Step 2: Global MCMC Sampling (Structural Uncertainty)

- **Goal:** Capture structural uncertainty by sampling from the model's approximate posterior over graphs
$$P(G) \propto \exp\big(-E_\psi(G)\big).$$
This reveals whether multiple structurally distinct, low-energy graphs are plausible.
- **Algorithm:** Use tempered Stochastic Gradient Langevin Dynamics (SGLD) in the low-rank factor space (U_r, V_r).

- **Process:**

  1. **Diversified Initialization.** Start k parallel Markov chains. Initialize the factors for chain c as:

  $$U_0^{(c)} = U_{\text{mean}}^{(0)} + \epsilon_U^{(c)}, \quad V_0^{(c)} = V_{\text{mean}}^{(0)} + \epsilon_V^{(c)}, \quad c = 1, \dots, k,$$

  where $\epsilon_U^{(c)}, \epsilon_V^{(c)}$ are small Gaussian perturbations.

  2. **Persistent Chains.** Maintain these k chains across queries. Their states are updated incrementally, which reduces burn-in costs for subsequent inferences.

  3. **Tempered SGLD Updates.** For each step $t = 0, \dots, T-1$ and each chain c:

     – Reconstruct the current adjacency:

     $$A_t^{(c)} = \mathrm{Reconstruct}\big(U_t^{(c)}, V_t^{(c)}\big).$$

     – Compute the tempered gradients:

     $$\nabla_U E_\psi^{\text{temp}} = \frac{1}{T_t}\left(\frac{\partial E_\psi}{\partial A_t^{(c)}} \cdot \frac{\partial A_t^{(c)}}{\partial U_t^{(c)}}\right),$$

     $$\nabla_V E_\psi^{\text{temp}} = \frac{1}{T_t}\left(\frac{\partial E_\psi}{\partial A_t^{(c)}} \cdot \frac{\partial A_t^{(c)}}{\partial V_t^{(c)}}\right),$$

     where T_t is a temperature schedule (e.g., annealed from a high value $T_{\max}$ to 1).

     – Apply the SGLD updates:

     $$U_{t+1}^{(c)} = U_t^{(c)} - \eta_t \nabla_U E_\psi^{\text{temp}}\big(A_t^{(c)}, H^{(L)}\big) + \sqrt{2\eta_t}\, \xi_{t,U}^{(c)},$$

     $$V_{t+1}^{(c)} = V_t^{(c)} - \eta_t \nabla_V E_\psi^{\text{temp}}\big(A_t^{(c)}, H^{(L)}\big) + \sqrt{2\eta_t}\, \xi_{t,V}^{(c)},$$

     where η_t is a step size and $\xi_{t,U}^{(c)}, \xi_{t,V}^{(c)}$ are Gaussian noise terms.

- **Sampler Approximation.** The above procedure uses unadjusted Langevin dynamics (SGLD), which is a standard practical approximate sampler in energy-based models. It yields a high-quality approximation to samples from $P(G) \propto \exp(-E_\psi(G))$.

- **Output: Structural Uncertainty Ensemble.** After a sufficient number of steps (and ignoring initial transient steps), the final states of the k chains define an ensemble of graphs:

  $$G_{\text{ensemble}} = \{G_1, \dots, G_k\}, \quad G_c = \big(H^{(L)}, A_T^{(c)}\big).$$

  This ensemble represents the model's structural uncertainty. For any edge (i, j, r), its structural mean and variance are estimated by:

  $$\mu_{ijr}^{\text{struct}} = \frac{1}{k}\sum_{c=1}^{k} A_T^{(c)}(i,j,r), \qquad \Sigma_{ijr}^{\text{struct}} = \mathrm{Var}\big(\{A_T^{(c)}(i,j,r)\}_{c=1}^{k}\big).$$

Step 3: Efficient UQ-Aware Counterfactuals

- **Goal:** Answer interventional ("what-if") queries efficiently, with full uncertainty quantification.
- **Methodology:** Warm-start the persistent chains from the observational ensemble and adapt them under a clamped (intervened) representation.
- **Process:**

  1. **Intervention Definition.** For an intervention on node i, set its embedding to a clamped value H_i^{cf} (e.g., representing "knockout" or a specific intervention state). All other node embeddings remain as in $H^{(L)}$.
  2. **Warm Start.** Initialize the counterfactual chains from the final observational states:
  $$U_0^{(c,\text{cf})} = U_T^{(c)}, \quad V_0^{(c,\text{cf})} = V_T^{(c)}, \quad c = 1, \dots, k.$$
  3. **Adaptation.** Run tempered SGLD again for $T_{\text{adapt}} \ll T$ steps, now computing the energy E_ψ and its gradients with the clamped embedding H_i^{cf} in place of H_i. This allows the chains to move from observational low-energy modes to counterfactual low-energy modes with significantly reduced burn-in.

- **Output: Counterfactual Ensemble.** The resulting counterfactual ensemble
  $$G_{\text{ensemble}}^{\text{cf}} = \{G_1^{\text{cf}}, \dots, G_k^{\text{cf}}\}, \quad G_c^{\text{cf}} = \left(H^{\text{cf}}, A_{T_{\text{adapt}}}^{(c,\text{cf})}\right),$$
  provides a full posterior over graphs under the specified intervention. For any edge or causal query, both observational and counterfactual distributions can be compared.

Example Inference Walkthrough

Consider the query:

> "What is the strength of the causal link (Node $5 \to$ Node 20, `'Causes'`)? What happens to this link if we knock out Node 5?"

- **Step 1: Epistemic Uncertainty.**
  - Run f_θ $N_{\text{enc}} = 5$ times with dropout enabled to obtain $\{H_1^{(L)}, \dots, H_5^{(L)}\}$.
  - For each $H_s^{(L)}$, run Q_ϕ $N_{\text{prop}} = 10$ times, yielding $N_{\text{mc}} = 50$ factor samples $\{(U_n, V_n)\}$.
  - Compute warm-start factors $U_{\text{mean}}^{(0)}, V_{\text{mean}}^{(0)}$.
  - For the edge $(5, 20, \texttt{'Causes'})$, compute the epistemic mean and variance from $\{A_n(5, 20, \texttt{'Causes'})\}_{n=}^{50}$
- **Step 2: Structural Uncertainty.**
  - Initialize $k = 100$ chains at $U_{\text{mean}}^{(0)} + \epsilon_U^{(c)}$, $V_{\text{mean}}^{(0)} + \epsilon_V^{(c)}$.
  - Run tempered SGLD for T steps, annealing T_t to 1.

- Obtain $G_{\text{ensemble}} = \{G_1, \dots, G_{100}\}$ and estimate:

$$\mu^{\text{struct}}_{(5,20)} = \frac{1}{100} \sum_{c=1}^{100} A_T^{(c)}(5, 20, \text{`}\texttt{Causes}\text{'}),$$

  along with its variance and credible interval.

- **Step 3: Counterfactual Answer.**
  - Clamp $H_5 \to H_5^{\text{cf}}$ (e.g., a "knockout" embedding).
  - Warm-start the 100 chains from their T-step observational states.
  - Run SGLD for T_{adapt} steps with the clamped embedding to obtain $G^{\text{cf}}_{\text{ensemble}}$.
  - Compute the counterfactual mean and credible interval for $A(5, 20, \text{`}\texttt{Causes}\text{'})$ across $G^{\text{cf}}_{\text{ensemble}}$ and compare with the observational ensemble.

This procedure yields a deterministic inference protocol that produces both point estimates and principled uncertainty quantification for observational and counterfactual queries over the learned causal graph.