SAPIENZA
UNIVERSITÀ DI ROMA

Sapienza University of Rome

Dipartimento di Informatica
PhD School in Computer Science

Thesis For The Degree Of Doctor Of Philosophy

Graphs in Action: From Real-World Systems to Explainable AI and Beyond

Thesis Advisor
Prof. Gabriele Tolomei

Candidate
Flavio Giorgi
16148998

Academic Year MMXXII - MMXXV (XXXVIII cycle)

Abstract

Graphs provide a unifying language to reason about structure, constraints, and flow in both engineered systems and machine learning pipelines. This thesis advances that perspective along three directions. First, it studies graph based optimization for real-world, networked settings where communication, computation, and sensing are interdependent. By coupling connectivity, routing, and data reduction within a single planning framework, the work develops scalable algorithms that improve task completion, latency, and resilience under realistic resource limits.

Second, it investigates explainability for graph learning through counterfactual reasoning. The thesis formulates a unified view in which changes to node features, edge attributes, and topology are optimized to produce compact, faithful, and plausible "what-if" explanations. It further explores how these counterfactuals can be rendered as clear, human readable narratives, evaluated with both quantitative criteria and user studies.

Third, it moves beyond the graph domain by introducing training time mechanisms that use counterfactual signals to shape decision boundaries. This regularization improves generalization while amortizing the cost of explanation, enabling rapid retrieval of informative counterfactuals at inference. Finally, the thesis proposes a lightweight pipeline to build counterfactual narratives in the domain of tabular data, this two-stage process balances quality and efficiency, making explanation generation practical for latency and energy constrained deployments.

Together, these contributions show that graphs provide a powerful tool to model complex, real-world systems under realistic resource and reliability constraints. Moreover, our work demonstrates that, despite the complexity of their structure, graphs and related learning models can be effectively explained. Finally, we show that explanation techniques used in graphs can be extended to regularize and explain other machine learning models.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Graphs offer a unifying abstraction for reasoning about structure, constraints, and flow, spanning the communication topologies of multi-robot swarms and the dependencies within deep models. This thesis leverages that shared language to develop two lines of work: graph-driven optimization for real-world, networked systems and explainable AI (XAI) for graph learning. The overarching objective is to develop methods that perform reliably under realistic constraints and to improve human-centered interpretability for graph learning models. Finally, beyond graph specific settings, we also advance the explainability landscape more broadly, improving the state of the art outside the graph domain.

Graphs in real-world systems

Small UAV swarms must sense, move, and communicate under tight energy, bandwidth, and latency budgets. Classic networking formulations typically decouple deployment, routing, and data adaptation, which can lead to brittle behavior when the value of the data depends on the accuracy of downstream inference. Two gaps motivate this chapter. First, application-aware offloading: compression reduces congestion but can harm task accuracy, while high-fidelity data can overload multi-hop links. Therefore, planning should couple connected coverage, routing, and per-task compression to maximize the number of correctly executed tasks under deadlines and link capacities. Second, periodic connectivity: when only short-range radios are available, continuous end-to-end links are unrealistic; swarms should periodically form connected offloading formations that are fast to build and energy-aware. The chapter argues for graph-theoretic formulations such as flows, Steiner-like backbones, Delaunay structures, and scalable heuristics that preserve structure while operating online.

Explainable AI for graphs

Graph Neural Networks (GNNs) are now state-of-the-art for node-, edge-, and graph-level prediction in domains such as molecules, finance, and knowledge graphs, yet their decisions remain opaque. Saliency-style explanations identify what mattered for a decision, but users and regulators increasingly ask for counterfactual "what-if" evidence: what minimal change would flip the prediction? This chapter is driven by three needs. First, unified counterfactuals: most methods perturb only

the structure (for example, edge removals), neglecting node and edge attributes. A unified search over features, edges, and edge attributes promises more faithful and sparser explanations. Second, usability: even high-quality counterfactuals are technical artifacts, so they should be translated into concise, faithful natural language narratives that non-experts can act upon.

Explainable AI beyond graphs

Beyond post-hoc explanations, there is a training-time opportunity: models with smoother decision boundaries tend to be more robust. This motivates counterfactual regularization, namely, incorporating counterfactual geometry into the learning objective to enlarge margins and improve generalization . A second motivation for this chapter can be found in the efficient human-centered narratives generation at scale: high-end language models can produce excellent narratives, but are computationally and energy-intensive. Therefore, the chapter explores knowledge distillation and multi-stage refinement pipelines that enable small language models to generate coherent, faithful narratives with a fraction of the resources.

1.2 Contribution

Graphs in real-world systems: Application-Aware planning and Stop & Route

We introduce A^2-UAV, an application-aware framework that jointly optimizes connected coverage and compression to maximize the number of correctly accomplished tasks at the edge. Before deployment, the A^2-TA module learns the compression–accuracy map $Q : S \times L \to \mathbb{R}^2$, which induces empirical curves used at planning time. This departs from networking-only formulations by binding bitrate choices to task accuracy and deadlines.

We formalize the problem as an MILP that couples connectivity, capacity, flow conservation, and accuracy constraints; while solvable, optimal solutions are too slow for online operations, motivating a graph-based polynomial heuristic. The greedy algorithm we propose builds a connected coverage by merging partial Triangular Steiner Trees and co-assigning compression levels to data, finds the path bottleneck, and picks the best feasible compression level; the routine has polynomial time complexity, and the coverage selection uses a cost function to trade average accuracy loss and UAV usage.

We further develop an online adaptive variant that, upon failures or topology changes, recomputes the solution and migrates the fleet by reducing it to a bottleneck bipartite min-matching, thereby minimizing the worst migration time from the old topology to the new one. This yields faster service restoration than Steiner-tree baselines in the same scenarios.

On Stop & Route for periodic connectivity, we supply both centralized and distributed strategies. The Greedy Assignment Algorithm (GAA) operates on a hexagonal rendezvous tiling, maximizing the minimum residual energy of the least-energetic UAV and achieves $O(m^5)$ time; the candidate frontier is $O(m)$ at each step. The Tree Contraction Algorithm (TCA) avoids discretization: it builds the Delaunay graph, extracts the EMST, then contracts paths toward the base station, providing an upper bound on traveled distance and $O(m \log m)$ complexity. For short-range-only operations, the distributed DGA utilizes a ρ-grid of rendezvous points and leader–follower rules, relying solely on local information and a loosely synchronized clock.

In simulations and a real testbed with four UAVs and a Jetson Nano edge server, A^2-UAV increases accomplished tasks by about 38% on average over networking-based baselines (up to about 400% in high-load regimes), and sustains performance under packet loss; Stop & Route achieves orders-of-magnitude lower formation-time than MILP, reduces building times versus AME, and increases sensing time; DGA outperforms CPVF/Floor-CPVF in formation time and sensing utilization.

Explainable AI for graphs: Counterfactuals and Narratives

We introduce UCExplainer, a unified counterfactual framework that balances feature, edge-attribute, and structural perturbations under scheduling policies. A key ingredient is edge-weight sparsification, which cuts runtime while retaining validity, fidelity, and sparsity; for example, on Citeseer, UCExplainer reduces generation time from approximately 45.6 s (CF-GNNExplainer) to approximately 7.7 s under comparable settings, and remains competitive against generative baselines that shift cost to training.

We also propose a pipeline that translates graph counterfactuals into natural language using open-source LLMs. Given counterfactuals obtained by perturbing the structure and the node features, we prompt Qwen2.5 models to produce faithful rationales and assess them using a suite of metrics. Larger LLMs consistently improve graph-understanding scores.

Explainable AI beyond graphs

We introduce a counterfactual regularizer that incorporates a counterfactual term into the training objective to improve generalization . Using a lightweight score-based generator with local linearization, CF-Reg matches or outperforms L1/L2, dropout, early stopping, and PGD on several tabular setups; at test time, approximate counterfactuals can be retrieved in $O(1)$ via nearest-neighbor lookup over precomputed training counterfactuals. This turns counterfactuals from a purely post hoc artifact into a training signal that widens decision margins in data-relevant directions.

In parallel, we develop a separate line of work on counterfactual narratives through a multi-stage Narrative Refinement pipeline. The pipeline first uses a Draft Narrative Generator (DNG) to produce multiple candidate narratives for each factual–counterfactual pair, then employs a distinct Refiner model to merge, reconcile, and improve these drafts into a single, coherent explanation. Both stages are fine-tuned via knowledge distillation from a large teacher to compact student models, enabling deployment on modest hardware. To assess correctness and utility, we introduce structured, automatically verifiable metrics alongside a human-centered Narrative Quality evaluation. Experiments on the Adult and Titanic datasets show that refinement consistently boosts factual correctness and readability. Distilled small models with refinement reach or surpass the teacher on several metrics while substantially reducing memory, latency, and energy consumption.

1.3 Background

1.3.1 Graph Theory

Graphs provide a common language to reason about structure, dynamics, and optimization on relational data. This section introduces the core operators and notions that we will be used later.

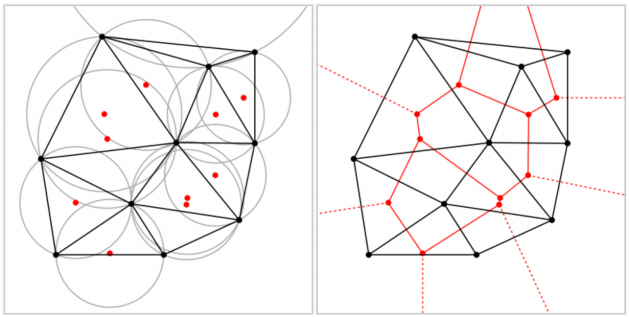


Figure 1.1: On the left, the Delaunay triangulation, on the right, the corresponding Voronoi diagram [144]

Graphs and their representation An undirected, weighted graph can be represented as $G = (V, E, w)$, where $V = \{1, \ldots, n\}$ is the node set, $E \subseteq \{\{i, j\} \mid i \neq j\}$ the edge set, and $w : E \to \mathbb{R}_{>0}$ an edge-weight function. In the directed case, $E \subseteq \{(i, j) \mid i \neq j\}$ and edges are ordered pairs. The structure is encoded by the adjacency matrix $A \in \mathbb{R}^{n \times n}$ with $A_{ij} = w(\{i, j\})$ (undirected) or $A_{ij} = w((i, j))$ (directed). The degree matrix is $D = \mathrm{diag}(d_1, \ldots, d_n)$ with $d_i = \sum_j A_{ij}$. For $S \subseteq V$, we write $\mathrm{vol}(S) = \sum_{i \in S} d_i$ and $\mathrm{cut}(S, \bar{S}) = \sum_{i \in S,\, j \in \bar{S}} A_{ij}$.

Paths and distances: local to global reachability. Shortest-path metrics turn movement and routing costs into algebra. Let $c_{ij} \geq 0$ be an edge cost (e.g., travel distance, latency, energy) and ρ_{ij} a link data rate. The weighted distance between u, v is

$$d(u, v) = \min_{p : u \rightsquigarrow v} \sum_{(i,j) \in p} c_{ij}, \qquad \text{and the bottleneck capacity is} \quad b(u, v) = \max_{p : u \rightsquigarrow v} \min_{(i,j) \in p} \rho_{ij}.$$

In our setting, distances support two roles: (i) Application-aware routing: in A2-UAV, routes must satisfy link capacities and deadlines; compression choices are scheduled along paths according to bottleneck bandwidths, coupling routing with task accuracy. (ii) Periodic connectivity: shortest paths guide how a disconnected swarm quickly forms a connected offloading topology.

Steiner trees: sparse backbones under terminal constraints. Given a set of terminals $T \subseteq V$, a Steiner tree seeks a minimum cost connected subgraph that spans T and may add auxiliary nodes (Steiner vertices). In Euclidean or graph settings, the problem is NP-hard [101], which motivates the use of approximation and heuristic methods for large instances. Classical 2-approximation schemes grow a minimum spanning tree over T in a metric closure and then map it back to the original graph [117]. More refined algorithms exploit full components and local improvements to achieve ratios below 1.6 in graphs [165]. In our applications, Steiner-like constructions provide parsimonious communication backbones: by concentrating capacity on a small number of relay paths, they reduce deployment and routing cost while maintaining terminal connectivity.

Delaunay graphs: geometric connectivity with good triangulations. For planar point sets $P \subset \mathbb{R}^2$, the Delaunay triangulation $\mathrm{DT}(P)$ is the straight-line planar graph whose triangles have empty circumcircles; it is the geometric dual of the Voronoi diagram and maximizes the minimum angle among all triangulations, avoiding skinny triangles [51, 54]. The triangulation has at most $3|P| - 6$ edges and average degree below 6, making it a sparse yet well-connected geometric

skeleton. Efficient algorithms compute $\mathrm{DT}(P)$ in $O(|P| \log |P|)$ time (e.g., divide-and-conquer or Fortune's sweep) [63]. Importantly, the Euclidean minimum spanning tree is a subgraph of the Delaunay triangulation, so $\mathrm{DT}(P)$ serves as a natural candidate superset from which to extract low-cost backbones for routing and coverage [51]. In our setting, Delaunay graphs provide planar, locality-preserving neighborhoods that stabilize mesh connectivity and facilitate shortest-path and flow computations on spatial deployments.

1.3.2 Graph Neural Networks

Graph Neural Networks (GNNs) learn representations that respect graph structure via message passing [72, 87]. Let $G = (V, E)$ with node features $X \in \mathbb{R}^{n \times d}$. At layer l, the embedding $h_i^{(l)}$ of node i aggregates information from its neighbors $\mathcal{N}(i)$:

$$h_i^{(l+1)} = \phi^{(l)}\Big(h_i^{(l)}, \square_{j \in \mathcal{N}(i)} \psi^{(l)}\big(h_i^{(l)}, h_j^{(l)}, e_{ij}\big)\Big), \tag{1.1}$$

where ψ and ϕ are neural functions and $\square$ is a permutation-invariant operator (sum/mean/max) [72]. This generic template admits concrete instantiations that trade off simplicity, adaptivity, and expressivity.

GCN (Graph Convolutional Network). The spectral model of [114] uses the propagation

$$H^{(l+1)} = \sigma\Big(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2} H^{(l)} W^{(l)}\Big), \qquad \hat{A} = A + I, \tag{1.2}$$

with $H^{(0)} = X$, learnable weights $W^{(l)}$, and nonlinearity σ. Here $\hat{D}$ is the degree matrix of $\hat{A}$, i.e., $\hat{D}_{ii} = \sum_j \hat{A}_{ij}$. The symmetric normalization $\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}$ (often denoted $\tilde{A}$) is the "renormalization trick": adding self-loops ensures each node injects its own features at every layer, while the normalization controls the scale of activations and improves numerical stability. Writing $\tilde{A} = \hat{D}^{-1/2}\hat{A}\ \hat{D}^{-1/2}$, a single layer is

$$H^{(l+1)} = \sigma\big(\tilde{A} H^{(l)} W^{(l)}\big),$$

i.e., neighborhood averaging by $\tilde{A}$ followed by a linear transform $W^{(l)}$ and a pointwise nonlinearity. After L layers the receptive field is L hops and the model implements a low-pass (smoothing) operator on the graph, which is effective under homophily.

A standard two-layer GCN for semi-supervised node classification reads

$$Z = \mathrm{softmax}\big(\tilde{A}\, \sigma(\tilde{A} X W^{(0)})\, W^{(1)}\big),$$

with cross-entropy loss on labeled nodes and L_2 regularization on $W^{(l)}$. With sparse $\tilde{A}$, the cost of the propagation $\tilde{A}H^{(l)}$ is $O(|E|\, d_l)$, so each layer scales linearly in the number of edges and the feature width d_l.

1.3.3 Explainable Artificial Intelligence

Explainable AI (XAI) aims to make the behavior of complex models understandable while balancing three desiderata: faithfulness to the model, sparsity of the explanation, and plausibility in the appli-

cation domain [57, 82]. Methods are commonly organized along two axes: intrinsically interpretable vs. post hoc, and local (explain one prediction) vs. global (characterize model behavior).

Regulatory context. In high-stakes domains (health, finance, public services), regulations emphasize transparency, documentation, and risk management. GDPR provisions require meaningful information about automated decisions and their logic where personal data are involved, while the EU AI Act introduces risk-based obligations (technical documentation, user transparency, post-market monitoring) for high-risk systems [1, 2]. These motivate graph-specific XAI pipelines that output verifiable artifacts: (i) a compact explanatory subgraph or small, feasible counterfactual edit set; (ii) metadata describing constraints and costs; and (iii) when needed, a user-facing textual rationale grounded in the actual structural/feature changes.

XAI for Graphs. In the GNNs context, explanations seek to identify the substructures and features that drive a decision at the node-, edge-, or graph-level. A dominant approach involves learning masks over graph components. For example, GNNExplainer learns a continuous mask M over edges and input features by maximizing a proxy ELBO that trades fidelity for sparsity, thereby recovering a compact explanatory subgraph and feature subset [218]. To amortize the cost of this per-instance optimization, PGExplainer utilizes a parametric network to directly predict edge importance from intermediate embeddings, thereby vastly reducing computation [131].

Other methods employ different paradigms. SubgraphX formulates explanation as a discrete subgraph search, estimating contributions via Shapley values calculated using Monte Carlo Tree Search (MCTS) to improve faithfulness under game-theoretic semantics [220]. In contrast, GraphLIME offers a local, model-agnostic approach by fitting sparse linear surrogates based on neighborhood features [99].

In parallel, traditional gradient and perturbation-based attribution methods (e.g., Saliency, Integrated Gradients, Occlusion) have been adapted for use on graphs. These are widely used as baselines or complementary signals, especially when mask learning proves unstable or when strict budgets on the number of revealed nodes/edges are imposed.

Counterfactual explanations for graphs. Counterfactuals ask for the smallest feasible change that flips the model outcome, providing actionable "what-if" evidence and a notion of recourse. Given x and classifier f, one seeks a nearby x' such that $f(x') \neq f(x)$ (or $f(x') = y_t$) while meeting realism constraints:

$$x' = \arg\min_{\tilde{x}} \; d(x, \tilde{x}) + \lambda \, \ell\big(f(\tilde{x}), y_t\big) \quad \text{s.t.} \quad \tilde{x} \in \mathcal{C}, \tag{1.3}$$

where $\mathcal{C}$ encodes feasibility and domain validity [110, 196]. On graphs, edits may affect structure (add/remove edges), node features, and edge attributes. Minimum-edit search is generally NP-hard (via reductions from Set Cover/Steiner-type problems), so practical solvers use continuous relaxations (e.g., Gumbel–softmax, straight-through estimators), stochastic search (MCMC, evolutionary strategies), or greedy heuristics with explicit sparsity regularization [127]. Constraints typically include degree/size budgets, connectivity preservation of critical components, type constraints (categorical one-hot, bounded continuous), and domain rules that prevent implausible edits. For recourse, heterogeneous cost models weight edits and promote diversity so users receive multiple distinct options.

Evaluation metrics. In order to properly evaluate the explanations key criteria include: faithfulness (how well the explanation accounts for model behavior; often probed with deletion/insertion curves or fidelity under masking), sparsity (size of the explanatory set of nodes/edges/features), stability (sensitivity to noise, seeds, or retraining), plausibility (agreement with domain constraints/prior knowledge, e.g., chemical valency), and consistency (similar inputs $\rightarrow$ similar explanations). In the counterfactual case, instead, we use metrics such as validity (flip occurs), proximity (magnitude/number of edits), sparsity, and diversity; these are often complemented by feasibility scores induced by $\mathcal{C}$ [142]. When explanations are rendered in natural language, semantic faithfulness and human-judged usefulness are additional lenses, but the underlying graph edits remain the source of truth.

1.3.4 Large Language Models

Large Language Models (LLMs) are autoregressive neural networks trained for next-token prediction over massive corpora. The dominant architecture is the Transformer (Figure 1.2) [190]. Let $X \in \mathbb{R}^{T\times d}$ be token embeddings. With learned projections W_Q, W_K, W_V, multi-head self-attention computes

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V, \tag{1.4}$$

$$\mathrm{Attn}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V, \tag{1.5}$$

$$\mathrm{MHAtt}(X) = \mathrm{Concat}(\mathrm{Attn}_1, \ldots, \mathrm{Attn}_H)\, W_O, \tag{1.6}$$

followed by residual connections and position-wise feed-forward blocks ($\mathrm{FFN}(x) = W_2\,\sigma(W_1 x)$). Training maximizes the causal log-likelihood

$$\max_\theta \sum_{t=1}^{T} \log p_\theta(x_t \mid x_{<t}). \tag{1.7}$$

Instruction tuning and RLHF further adapt LLMs to interactive use and human preferences [153].

LLMs and Graphs. In the context of this thesis, Large Language Models (LLMs) are utilized to describe graph structures, summarize subgraph level rationales, and verbalize counterfactual edits into natural language narratives [24]. This capability is operationalized in hybrid pipelines (e.g., GNN $\rightarrow$ Explainer $\rightarrow$ LLM), where the LLM serves a dual purpose. First, it translates structured rationales, such as explanatory subgraphs or counterfactual edit sets, into human-readable text. Second, this pipeline enables automatic checks for the coherence, and correctness of the generated explanations.

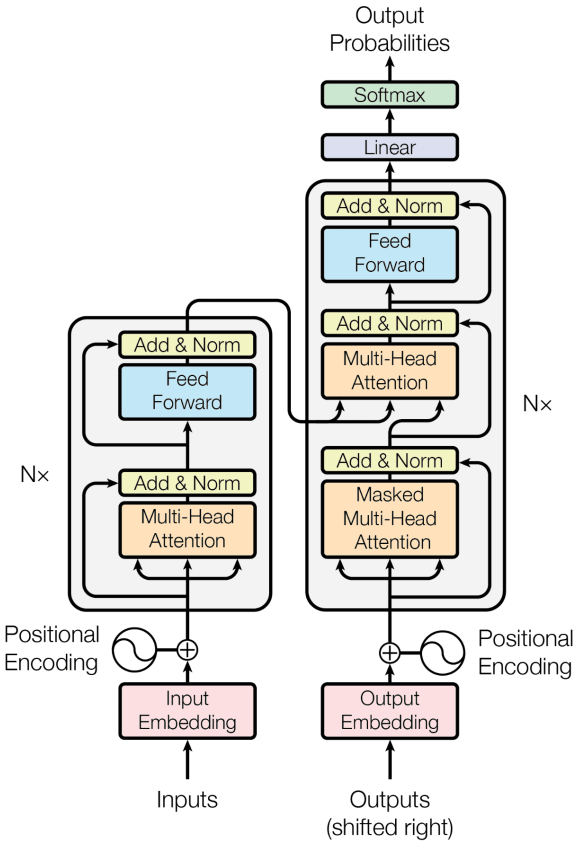


Figure 1.2: Transformer architecture from "Attention Is All You Need" [190].

Chapter 2

Graphs in real-world systems

2.1 Introduction

In UAV networks, graphs are powerful tools to reason about communication and computation. In our setting, nodes represent UAVs, targets, and edge servers; edges encode wireless reachability, routing paths, or semantic dependencies between data and models. This chapter shows how graphs enable principled formulations and scalable heuristics for two real-world scenarios: (i) semantic UAV communications, where the value of sensed data depends on deep-learning accuracy under compression; and (ii) periodic communication, where UAV swarms must intermittently build connected formations to offload data under energy and range constraints. Throughout, all optimization and heuristic solutions are graph-based: we cast deployment, routing, and data adaptation as problems on communication and geometric graphs, and we solve them using graph-theoretic operators, flows, and sparsifying backbones.

From semantics to connectivity-aware optimization. Semantic communication couples the networking load induced by offloading with the downstream model accuracy: higher compression alleviates congestion but may degrade inference. We formalize this trade-off by attaching a compression–accuracy profile learned offline to each target and embedding it into a communication graph, where capacities approximate link data rates. The resulting A^2-TPP pipeline exploits graph flows to enforce connectivity and deadlines, while selecting UAV positions and per-target compression levels. The MILP captures these couplings explicitly; its NP-hardness follows by reduction from Steiner-type problems, reflecting the inherent difficulty of constructing connected offloading structures. This motivates graph-driven heuristics that remain faithful to the structure of the problem: a greedy scheme grows Triangular Steiner Trees to connect the edge with selected targets while budgeting congestion; the compression assignment is computed along paths via bottleneck bandwidths. As shown later, this application-aware, graph-based design substantially increases the fraction of correctly executed tasks in simulation and on a physical testbed.

From intermittent links to connected offloading formations. When only short-range radios are available, global connectivity cannot be guaranteed at all times. We therefore orchestrate periodic stop-and-route phases that transiently create a connected offloading formation. Here again, the graph view is central. In the centralized case, the GAA algorithm maps UAVs to a sparse set

of rendezvous points, selecting assignments that maximize the minimum residual energy, effectively solving a matching over a discretized geometric graph. The TCA algorithm replaces discretization with geometry: starting from the Delaunay graph of current positions, it computes an Euclidean MST and contracts its edges until hop-by-hop communication is feasible, yielding near-linear time and an explicit bound on traveled distances. In the distributed case, the DGA algorithm achieves the same goal with only local neighborhood information by guiding UAVs along a rendezvous lattice and leader–follower rules that propagate connectivity toward the base station.

Why graphs help in practice. Across both scenarios, graphs are more than a modeling convenience. They provide (i) feasible constraints: connectivity via flow conservation and hop-limited paths; (ii) useful priors: Delaunay triangulations and Steiner backbones act as sparse, well-conditioned supports for routing and deployment; and (iii) effective heuristics: greedy expansions on trees, bandwidth-aware path updates, and distributed moves on rendezvous graphs preserve structure while scaling to large swarms.

This chapter draws on and consolidates material from [15, 16, 47, 48].

2.2 Graphs in Semantic UAV communications

Unmanned Aerial Vehicles (UAVs), or drones, have obtained significant attention thanks to their potential use in post-disaster scenarios, where human intervention is difficult or inefficient due to the vastness and/or harshness of the area. The key advantage of UAVs is the combined presence of advanced sensor equipment, wireless multi-hop networking and mobility in the same device, thus enabling critical applications such as automatic target (e.g., object, person) detection and tracking.

To perform their functions, modern UAVs necessarily depend on the execution of computation-heavy deep learning (DL) tasks to analyze in real time the images of the target area. These tasks usually rely on very deep neural networks (DNNs) such as ResNet [89] and DenseNet [98], which are computationally prohibitive for UAVs [138]. To extend UAVs battery lifetime and keep task execution time within acceptable levels, offloading the stream of tasks to neighboring edge servers (*e.g.*, the depot) is a feasible option [20, 39, 95, 167, 170, 209]. Unfortunately, *UAVs networks typically experience limited bandwidth and frequent packet loss* [46]. Prior work on UAV edge task offloading — discussed in details in Section 2.2.1 — assumes a single-hop communication between the UAVs and the edge [216][28], or focuses only on networking aspects on a multi-hop communication [38]. All fall short in considering the specific task requirements, which ultimately limits the number of correctly executed tasks. Existing work also rarely uses a testbed to measure performance experimentally.

Figure 2.1: Examples of application-aware compression. A and C (B and D) represent the images before (after) processing.

In stark contrast, we propose `A`^{2}`-UAV`, an application-aware (A^2) optimization framework which jointly optimizes UAV network deployment and task accuracy. Our key intuition is that a different image compression will result in a different accuracy for the DNN model. Specifically, compressed images have low impact on the network throughput but decrease the DNN accuracy. Conversely, uncompressed images are likely to be classified more correctly but cause higher network overhead. To better highlight this intuition, Figure 2.1 shows an example of an original and compressed ($l = 20$) images of *Wildlife* and *Tools* from the ImageNet dataset [55]. Figure 2.1.B shows how an hatchet at compression level $l = 5$ is less recognizable with respect to its original version (Figure 2.1.A), due to the low contrast with the background and the small object dimension. On the other hand, Figure 2.1.D shows how a jaguar at level $l = 20$ is easily distinguishable, thanks to its fur pattern and the higher contrast with the background. We consider lossy compression algorithms for images (e.g., JPEG algorithm [25]), as lossless algorithms do not usually meet the requirements of real-time applications.

Trading-off network load and latency with task accuracy while aiming at maximizing the target

coverage is a daunting challenge. A^2-UAV considers all these aspects and balances between images compression and network coverage according to current network conditions and application requirements. In Section 2.2.2 we show that different applications have starkly different compression-accuracy relationships. Specifically, A^2-UAV maximizes the number of accomplished tasks at the edge, producing a connected coverage formation for the UAVs of the squad and a compression level assignment for the UAVs that satisfy the application requirements. The connected coverage formation is designed to jointly maximize the number of covered targets, minimize network delay due to inefficient routes or channel contention, and minimize task misclassification due to high input compression. We show through simulation and real-world experiments with a testbed that Greedy-Application-Aware Task Planning Problem (A^2-TPP) attains on the average 38% more accomplished tasks than the state-of-the-art networking-based approach, with a sharp increase (400% more accomplished tasks) when the number of tasks to offload drastically increases.

2.2.1 Related Work

Only very recently has the literature considered task offloading in the context of edge-assisted DL-based applications [28, 46]. Chuprov et al. [46] show how the performance of the end-line ML systems is affected by the quality of data and network. They consider packet loss and limited bandwidth in a image classification task, and they recommend to stop the system when packet loss reaches 2-5%. In Section 2.2.4 we show that our system enables the classification task even with 15% of packet loss. Chen et al. [38] consider a hierarchical offloading of computation tasks. Conversely from us, they focus on the communication and routing of tasks toward a more computational powerful device, without focusing on the specific task requirements. Yang et al. [216] propose a hierarchical DL task execution framework, in which only a few lower layers of a Convolutional Neural Network (CNN) are on the UAVs, while the edge server contains the higher layers of the model, which need more resources. However, a single-hop high-performance 4G network is considered, while we focus on the more challenging scenario of multi-hop connectivity toward the edge. Recently, Callegaro *et al.* [28] proposed SeReMAS, a framework where the application-, network- and telemetry-based features are used to select and assign UAVs tasks to the most reliable edge servers. However, a single-hop system is considered, and data compression is not explored.

As one of the output of A^2-UAV is a connected coverage formation, we mention also some prior art on UAVs deployment optimization [18, 23, 38, 112, 160, 198], which however does not consider task offloading. Natalizio et al. [147] have considered the problem of minimizing networking resources while maximizing the user experience (i.e., perceived quality) when filming sport events. Moreover, [20, 170, 181] optimize network deployment under continuous or periodic connectivity constraints, but they do not consider critical indicators such as task accuracy with delay constraints, which are critical to the UAVs mission. Recently, Nguyen proposed a Steiner-Tree-Based Algorithm (STBA) for target coverage and network connectivity [149], where Fermat points and the node-weighted Steiner tree algorithm are used to find a tree such that most of the targets are covered, and the UAVs are minimized. In Section 2.2.4, we consider variants of [149] as performance benchmarks for A^2-UAV.

2.2.2 The A²-UAV Framework

In this section, we give an overview of A²-UAV (Section 2.2.2) and describe the two key components of A²-UAV: A²-TA (Section 2.2.2) and A²-TPP (Section 2.2.2).

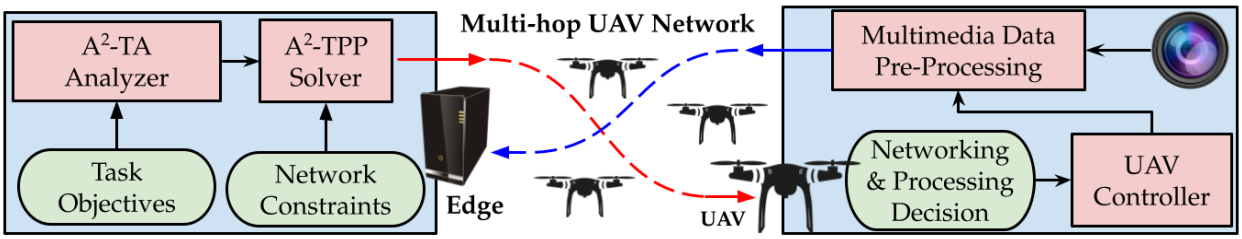


Figure 2.2: High-level overview of A²-UAV.

Figure 2.2 shows a high-level overview of A²-UAV. The Application-Aware Task Analyzer (A²-TA) at the *edge server* learns the relationship between the image compression and the accuracy on a set of classes of interest, and passes its output to the Application-Aware Task Planning Problem (A²-TPP) solver, which jointly optimizes UAVs positions, routing policy, and data compression strategy to maximize the number of correctly executed tasks per unit of time. The optimal network deployment and image compression levels are sent to the *UAVs network*, which moves to the targets, monitors them and streams tasks to the edge through the multi-hop connection.

System Model and Assumptions

We assume one or more UAVs are deployed over an Area of Interest (AoI), which contains several *targets*, e.g., the location of a vehicle, person, or any entity of interest. Each UAVs is equipped at least with (i) multimedia sensors (e.g., camera and microphone); (ii) a single radio for communication; and (iii) a computational unit. The edge server is equipped with low-latency hardware for DL computation. We do not rely on any communication infrastructures (e.g., 5G) and assume edge offloading is realized through multi-hop communication. A UAV monitors a target by sampling data through its sensors and generates a *task* to be executed at the edge. A task could be "car, bicycle, or bus detection on a video camera frame every 10 frames". The task is then sent to the edge server through a multi-hop connection, where a state-of-the-art DL model is run to perform the task. We assume each task has mission-driven constraints in terms of (i) minimum classification accuracy given a specified DL model; (ii) maximum latency, defined as the time between the task generation and its successful execution. Thus, a task is successfully executed if (i) promptly offloaded to the edge; and (ii) correctly analyzed by the model within a deadline.

Overview of A²-UAV

The ultimate goal of A²-UAV is to maximize the number of correctly executed tasks. To approach this challenging issue, and conversely from existing work, *A²-UAV takes into account how the task success is affected by the image compression.* To this end, A²-UAV jointly optimizes the deployment of UAVs and the task offloading to maximize the number of executed tasks. Each UAV is made up of two key modules. First, the *UAV Controller* implements networking and data processing decisions (next position, targets to cover, sampling process, and offloading routes) received from

Notation	Description
$\mathcal{U}$	A set of available UAVs of the fleet
$\mathcal{T}$	A set of targets to cover
σ	The edge server
r_{com}	Drone's communication radius
r_{sens}	Drone's sensing radius
l_u	Distance traveled by a drone
δ_u	Drone's overall energy consumption
$\hat{e}^s_{ij}$	Amount of data transmitted through the link between UAV i and UAV j
$\hat{e}^a_{ij}$	Expected task accuracy at the edge, for each task
$\rho_{i,j}$	Estimated channel data rate
$\hat{p}_u$	Position vector assigned by the solver to drone
β_u	Energy spent for each distance unit traveled at constant speed
α_u	Energy spent in a steady position
$\hat{\omega}^u_{ij}$	Drone u monitors the target i with a compression j or not
ϕ_u	UAV initial energy
$Q(s,l)$	A tuple with expected accuracy and data size of the frame of the application scenario s, with compression level l
Ψ	The final connected coverage formation returned by the greedy algorithm
τ_{best}	Best coverage found during an iteration
τ_{par}	Partial coverage to enhance or join with Ψ

Table 2.1: Table of Symbols.

the A^2-TPP. Second, the *Multimedia Data Pre-Processing* module samples data and creates tasks by pre-processing collected multimedia data according to the A^2-TPP solution.

Application-Aware Task Analyzer (A^2-TA)

The A^2-TA module determines the relationship between different image compression levels and task accuracy so that UAVs can reduce the amount of transmitted data and avoid network congestion, while satisfying application requirements.

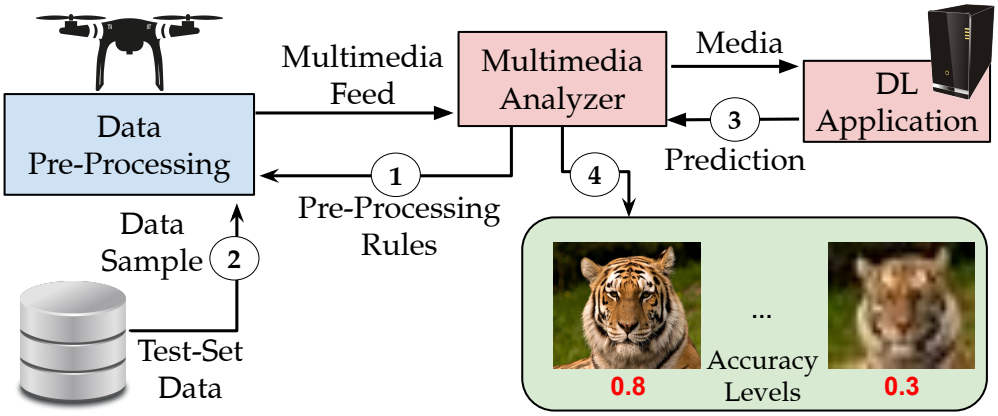


Figure 2.3: Main Operations of the A^2-TA module.

Figure 2.3 shows the workflow of the A^2-TA, which is executed before the network deployment, by using the datasets of the specific DL application. The A^2-TA iterates over the sampled data,

changing compression according to the UAV sensors capabilities (**step 1** and **step 2**). Each revised data sample is fed to the DL application, which outputs a prediction (**step 3**). Finally, predictions are compared with the ground truth, to infer and store the model performance according such data compression (**step 4**).

More formally, the function: $Q : \mathcal{S} \times \mathcal{L} \rightarrow \mathbb{R}^2$ maps an application scenario in the set $\mathcal{S}$ and a compression level in the discrete set $\mathcal{L} = \{1, .., 100\}$ a tuple in $\mathbb{R}^2$ representing the average *accuracy* and *data size*. The function is determined by iterating over the application scenarios, possible compression levels and relative dataset entries. Each sampled image is compressed according the compression level l by the JPEG compression algorithm [25], and it is fed into the DL model to determine accuracy and data size.

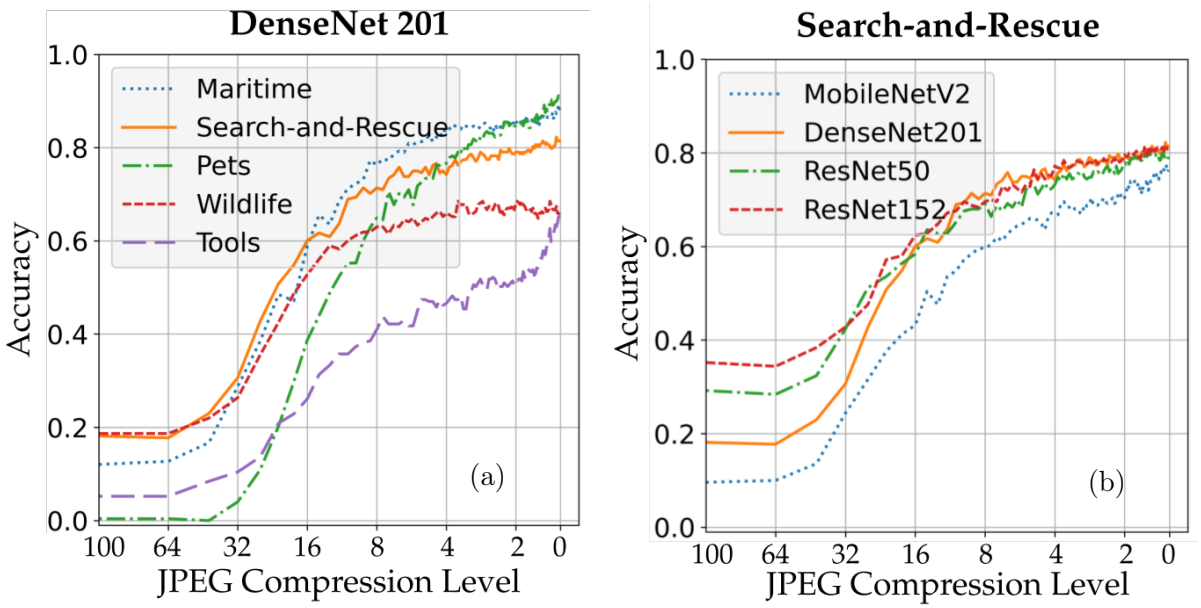


Figure 2.4: Accuracy as a function of JPEG compression level.

To give an example, we consider 5 scenarios: **Maritime** (Fireboat, Wreck, Lifeboat, Ocean liner, Speedboat), **Search-and-Rescue (SaR)** (Fire truck, Ambulance, Police van, German shepherd, Pickup truck) **Wildlife** (Kit fox, Polecat, Red wolf, Zebra, Jaguar), **Tools** (Screwdriver, Power drill, Hatchet, Hammer, Chainsaw), **Pets** (Golden retriever, Pomeranian, Guinea pig, Persian cat, Hamster). Figure 2.4.a shows the accuracy for the different scenarios as a function of the compression, while Figure 2.4.b shows the accuracy for the same scenario when different DL models are used. The figures highlight the need of the $\texttt{A}^2\texttt{-TA}$. For example, images of *Tools* have low accuracy, constraining the compression at level $l = 8$ to achieve at least 40% of accuracy, while *Wildlife* can achieve the same accuracy with higher compression $l = 25$.

$\texttt{A}^2\texttt{-TPP}$ MILP Formulation

We formalize $\texttt{A}^2\texttt{-TPP}$ as a Mixed Integer Linear Programming (MILP). Hereafter we denote the edge server as σ, the set of targets to monitor as $\mathcal{T}$, the set of available UAVs of the fleet as $\mathcal{U}$. The $\texttt{A}^2\texttt{-TPP}$ solver outputs: (i) a connected coverage formation of UAVs, and (ii) the compression level each drone must adopt to capture images when inspecting a target.

Definition 2.2.1. *A subset of UAVs $F \subseteq \mathcal{U}$ is deployed according to a **connected coverage formation**, when some of the UAVs in F are employed to inspect a subset of targets $M \subseteq \mathcal{T}$, while being connected to the base station σ, either directly or through a multi-hop sequence of the other UAVs in F.*

Definition 2.2.2. *A **task** for a drone $u \in F$ covering a target $t \in M$, consists in delivering an image captured from the drone's on-board cameras to the base station σ. It is said to be an **accomplished task** if two conditions hold: (1) when received at σ, the time since the task was created is not superior to a threshold Δ; (2) when the captured image reaches σ, the DL model outputs a correct prediction for it.*

We define the UAV sensing range and communication range as r_{sens} and r_{com}, respectively.

We denote with p the position vector of the entities involved, and to (p_u^x, p_u^y) for the x and y coordinates respectively. In particular p_u is the position of UAV u, $\forall u \in \mathcal{U}$ at the beginning of the mission; p_i to the position of the target i, $\forall i \in \mathcal{T}$; p_σ the position of the edge server. We denote with $\hat{p}_u$ the position vector assigned by the solver to drone u, $\forall u \in \mathcal{U}$. We define the distance traveled for each drone as $l_u = |p_u - \hat{p}_u|$, $\forall u \in \mathcal{U}$.

To estimate energy consumption, we define β_u as the energy spent for each distance unit traveled at constant speed, and α_u as the energy spent in a steady position, for a given unit of time. The values of β_u and α_u are estimated through on-field experiments or from technical specifications. The overall energy consumption for UAV u is defined as $\delta_u = l_u \cdot \beta_u + \alpha_u \cdot \Lambda$, $\forall u \in \mathcal{U}$, where Λ is an upper-bound of the time required, once reached the targets, to monitor the targets and complete the mission. We constraint the reachable points according to the UAVs initial energy ϕ_u, as follows:

$$\delta_u + |p_\sigma - \hat{p}_u| \cdot \beta_u \leq \phi_u, \ \forall u \in \mathcal{U} \tag{2.1}$$

This constraint defines the positions that are reachable as they let the UAVs with enough energy to come back to the edge-server for recharging operations. We use the binary variables $\hat{\omega}_{ij}^u \in \{0, 1\}$, which define if the drone u monitors the target i with compression level j or not. A target i is monitored if an only if the UAV u is close enough to the target position p_i. Formally we want to constraint $\hat{\omega}_{i,j}^u = 1 \iff |\hat{p}_u - p_i| \leq r_{\text{sens}}$ which becomes $\forall u \in \mathcal{U}$, $\forall i \in \mathcal{T}$, $\forall j \in \mathcal{L}$: :

$$\begin{aligned} r_{\text{sens}} &\geq |\hat{p}_u - p_i| - M_{\text{cost}} \cdot (1 - \hat{\omega}_{i,j}^u) \\ r_{\text{sens}} &\leq |\hat{p}_u - p_i| + M_{\text{cost}} \cdot \hat{\omega}_{i,j}^u \end{aligned} \tag{2.2}$$

where M_{cost} is a big constant number. Next, we enforce that a target is covered by at most one UAV and that each UAV can cover only one target:

$$\sum_{u \in \mathcal{U}} \sum_{j \in \mathcal{L}} \hat{\omega}_{ij}^u \leq 1, \forall i \in \mathcal{T}, \ \sum_{i \in \mathcal{T}} \sum_{j \in \mathcal{L}} \hat{\omega}_{ij}^u \leq 1, \forall u \in \mathcal{U} \tag{2.3}$$

We introduce an extended node set $\mathcal{V} = \mathcal{U} \cup \{\sigma\}$ and we consider all the possibles communication paths among nodes, i.e., all the edges (i, j) $\forall i, j \in \mathcal{V}$. A binary variable $\hat{\gamma}_{i,j} \in \{0, 1\}$, indicates if the nodes i and j are too distant to communicate. The relation $|\hat{p}_i - \hat{p}_j| \geq r_{\text{com}} \Rightarrow \hat{\gamma}_{ij} = 0$ is enforced as follows:

$$|\hat{p}_i - \hat{p}_j| \leq r_{\text{com}} + M_{cost} \cdot (1 - \hat{\gamma}_{ij}), \ \forall i, j \in \mathcal{V} \tag{2.4}$$

We define the data frame offloading as a network flow formulation. We introduce a set of variables $\hat{e}^s_{ij}$ defining the amount of data transmitted through the link between UAV i and UAV j. We define $\hat{e}^a_{ij}$ to account for the expected task accuracy at the edge, for each task. We impose that the edge does not generate any outgoing flow, for both data and accuracy flows:

$$\sum_{j\in\mathcal{V}} \hat{e}^s_{\sigma j} \leq 0, \quad \sum_{j\in\mathcal{V}} \hat{e}^a_{\sigma j} \leq 0 \tag{2.5}$$

We allow a flow only for between neighboring nodes:

$$\hat{e}^s_{ij} + \hat{e}^a_{ij} \leq \hat{\gamma}_{ij} \cdot M, \ \forall i,j \in \mathcal{U} \tag{2.6}$$

The maximum bandwidth allowed between two UAVs is constrained to respect the estimated channel data rate $\rho_{i,j}$:

$$\sum_{j\in\mathcal{V}} \hat{e}^s_{ij} \leq \rho_{i,j}, \ \forall i \in \mathcal{U} \tag{2.7}$$

We specify that a UAV can transmit only towards another UAV, resulting into a tree rooted at the edge:

$$\sum_{j\in\mathcal{V}} \hat{\gamma}_{ij} \leq 1, \ \forall i \in \mathcal{U} \tag{2.8}$$

We impose flow conservation as follows:

$$\sum_{k\in\mathcal{V}} \hat{e}^s_{uk} - \sum_{k\in\mathcal{V}} \hat{e}^s_{ku} = \sum_{i\in\mathcal{T}} \sum_{j\in\mathcal{L}} b_{i,j} \cdot \hat{\omega}^u_{ij}, \ \forall u \in \mathcal{U} \tag{2.9}$$

which imposes that, for each outgoing edge from u, the flow is increased by expected data size of the target covered by the UAV u . We also impose that the edge receives all the data produced by the covered targets:

$$\sum_{k\in\mathcal{V}} \hat{e}^s_{k\sigma} = \sum_{i\in\mathcal{T}} \sum_{j\in\mathcal{L}} b_{i,j} \cdot \hat{\omega}^k_{ij} \tag{2.10}$$

To conclude, we constraint the accuracy of the targets at the edge-server, as follows:

$$\sum_{k\in\mathcal{V}} \hat{e}^a_{uk} - \sum_{k\in\mathcal{V}} \hat{e}^a_{ku} = \sum_{t^i_j\in\mathcal{T}'} a_{i,j} \cdot \hat{\omega}^u_{ij}, \ \forall u \in \mathcal{U} \tag{2.11}$$

Objective Function: maximize covered targets, DL tasks accuracy, and energy spent by the UAVs:

$$\max \quad \alpha \cdot \sum_{j\in\mathcal{V}} \hat{e}^a_{\sigma j} + \beta \cdot \sum_{i\in\mathcal{T}, j\in\mathcal{L}, u\in\mathcal{U}} \hat{\omega}^u_{ij} - \eta \cdot \sum_{u\in\mathcal{U}} l_u \tag{2.12}$$

The term α prioritizes the maximization of the accuracy, while β weights the importance of covering the targets and η minimized the distance traveled by the UAVs.

Theorem 2.2.1. *The $\mathtt{A}^2$-TPP problem is NP-Hard.*

Proof. We show that $\mathtt{A}^2$-TPP generalizes the *Steiner tree problem with minimum number of Steiner points and bounded edge-length* STPMSPBEL, a known NP-hard problem [122]. Given a set P of n terminal points in a 2-dimensional plane, a positive constant R, and a non-negative integer B,

STPMSPBEL asks whether it exists a tree spanning a set of points $P \subseteq Q$ s.t. each edge has a length less than R and the number of Steiner points (i.e., $Q \setminus P$) is less than or equal to B. Any instance of STPMSPBEL can be reduced to an instance of our problem in polynomial time. The set of points P represents our target set $\mathcal{T} \cup \{\sigma\}$, and B defines the number of available UAVs, with communication range equal to R. We consider UAVs with unlimited batteries and one compression level (i.e, $|\mathcal{L}| = 1$). This problem instance finds a solution that maximizes the number of connected targets with the edge server, moving the minimum number of UAVs. If such a solution exists, and covers all the points in P, then it also exists a tree spanning a set of points $P \subseteq Q$, where each edge has length less than R and the number of Steiner points is less then or equal to B. The complexity of the above reduction is polynomial, thus we derive that $\texttt{A}^2\texttt{-TPP}$ problem is at least as hard as the STPMSPBEL problem [122]. □

2.2.3 A Graph Based Polynomial Time Heuristic for $\texttt{A}^2\texttt{-TPP}$

We propose a greedy heuristic to solve $\texttt{A}^2\texttt{-TPP}$ in polynomial time. We first introduce the algorithm, and then prove its polynomial time complexity.

Algorithm Overview

GREEDY-$\texttt{A}^2\texttt{-TPP}$ outputs a *connected coverage formation* – also referred to as *coverage* for brevity – for the UAVs, and a *compression level assignment* for each covered target. Both coverage and compression need to meet the criteria expressed in Equation 2.12, that is, optimizing the *number of accomplished tasks*. Our approach is to maximize the number of inspected targets, producing a coverage of minimum congestion, and minimizing task misclassification due to low frame resolution.

Specifically, a coverage $\tau = (V, E, W)$ is a Triangular Steiner Tree [172] in which the set of nodes V represents the positions UAVs must reach to cover the target nodes in M, while staying connected with the base station σ in a multi-hop manner. The set E represents the link between UAVs, thus the routes data streams must follow through the network. The function W maps each $e \in E$ to a weight that represents link's bandwidth. In our implementation we estimate this value empirically. It is assumed that at the base station, communication happens through dedicated transceivers and does not require actual coverage with a drone. Thus, at any time it holds $|V| \leq |\mathcal{U}| + 1$.

GREEDY-$\texttt{A}^2\texttt{-TPP}$

Algorithm 1 returns a coverage formation Ψ, merging partial coverage formations τ_{par} generated to cover targets using the minimum deployment cost at each iteration. In the initialization phase, we let: $\widehat{\mathcal{T}}$ be the set of covered targets, initially containing only the base station; Ψ be the coverage archived so far; τ_{par} be the partial coverage iteratively grown that is added to Ψ when it cannot be further expanded; c_{par} be the cost of the partial coverage generated so far (**line 1**). The while loop iterates until either all the targets are covered $\mathcal{T} - \widehat{\mathcal{T}} \neq \emptyset$ or the number of UAVs used does not exceed the fleet size (**line 2**).

The variables $t_{\text{best}}, \tau_{\text{best}}, c_{\text{best}}$ contain respectively the best target found at each iteration, the coverage including that target and its cost. A for loop over the uncovered targets $\mathcal{T} - \widehat{\mathcal{T}}$ allows to find the best target to add, building new temporary coverage formations τ_{temp} using the targets already covered by τ_{par} (namely the set $\mathcal{C}(\tau_{\text{par}})$) and adding to them the candidate target t. Then

Algorithm 1 GREEDY A^2-TPP

Input: $\mathcal{U}$: set of UAVs, $\mathcal{T}$: set of targets
Output: Ψ a connected coverage formation

$\widehat{\mathcal{T}}, \Psi, \tau_{\text{par}}, c_{\text{par}} \leftarrow \{\sigma\}, \{\sigma\}, \{\sigma\}, 0$
while $\mathcal{T} - \widehat{\mathcal{T}} \neq \emptyset$ **or** $|V_\Psi \cup V_{\text{par}}| < |\mathcal{U}|$ **do**
  $t_{\text{best}}, \tau_{\text{best}}, c_{\text{best}} \leftarrow \emptyset, \emptyset, \infty$
  for $t \in \mathcal{T} - \widehat{\mathcal{T}}$ **do**
    $\tau_{\text{temp}} \leftarrow \text{TST}(\{t\} \cup \mathcal{C}(\tau_{\text{par}}), r_{\text{com}})$
    $c_{\text{temp}} \leftarrow \text{COST}_\alpha(\tau_{\text{temp}}, \tau_{\text{par}}, \Psi, \text{COMPRESSION}(\tau_{\text{temp}}))$
    if $c_{\text{temp}} < c_{\text{best}}$ **and** $|V_{\text{temp}}| - 1 \leq |\mathcal{U}| - |V_\Psi \cup V_{\text{par}}|$ **then**
      $t_{\text{best}}, \tau_{\text{best}}, c_{\text{best}} \leftarrow t, \tau_{\text{temp}}, c_{\text{temp}}$
  if $t_{\text{best}} = \emptyset$ **then**
    $\Psi \leftarrow \Psi \cup \tau_{\text{par}}$
    break
  $\tau_{\text{los}} \leftarrow \text{TST}(\{\sigma, t_{\text{best}}\}, r_{\text{com}})$
  $c_{\text{los}} \leftarrow \text{COST}_\alpha(\tau_{\text{los}}, \tau_{\text{par}}, \Psi, \text{COMPRESSION}(\tau_{\text{los}}))$
  if $c_{\text{los}} < c_{\text{best}} - c_{\text{par}}$ **then**
    $\Psi \leftarrow \Psi \cup \tau_{\text{par}}$
    $\tau_{\text{par}}, c_{\text{par}} \leftarrow \tau_{\text{los}}, c_{\text{los}}$
  else
    $\tau_{\text{par}}, c_{\text{par}} \leftarrow \tau_{\text{best}}, c_{\text{best}}$
  $\widehat{\mathcal{T}} \leftarrow \widehat{\mathcal{T}} \cup \{t_{\text{best}}\}$
$R, \cdot \leftarrow \text{COMPRESSION}(\Psi)$
return Ψ, R

we evaluate the cost of τ_{temp} (**lines 3-6**). This cost combines the number of drones needed for the coverage, and the loss in accuracy due to the channel contention. We will talk in more detail about how this cost is computed when describing Algorithm 2. Then we check if: (i) τ_{temp} has a lower cost than τ_{best} (ii) and if τ_{temp} can be covered with the remaining UAVs.

If both checks go through, then t becomes the best candidate t_{best} and the associated candidate coverage τ_{temp} with its cost c_{temp} are stored into τ_{best} and c_{best} respectively (**lines 7-8**). In case no target was set as a best candidate, (i.e., $t_{\text{best}} = \emptyset$) the while loop breaks. This happens only when the second condition at line 7 is not met for any target, that is no coverage formations can stick to the remaining fleet size constraint.

Then, the cost paid to cover only t_{best} that is $c_{\text{best}} - c_{\text{par}}$ is compared to the cost c_{los} of a new line-of-sight (*los*) branch τ_{los} grown using only t_{best} as target. If τ_{los} costs less than the partial grown tree so far τ_{par}, then τ_{par} is merged with the final tree Ψ. Then τ_{los} becomes the new partial connected coverage to grow. Otherwise growing τ_{par} is still convenient, so τ_{best} becomes the new partial deployment including the new target t_{best} and τ_{los} is discarded (**lines 12-19**). When the algorithm terminates (**line 20**) the final coverage Ψ along with all the compression levels assigned to each target are returned.

Assignment of Compression Levels

Algorithm 2 determines the compression levels for each UAV in F inspecting targets in M. The rationale is to increase the compression of data flowing from a target, based on the bandwidth

Algorithm 2 COMPRESSION

Input: a coverage formation τ_i

Output: R vector with compression levels for all targets in τ_i, L vector with loss in accuracy due to all targets in τ_i

sort t by $Q(\mathcal{S}(t), *).b \; \forall t \in \mathcal{C}(\tau_i)$ in ascending order

$\widehat{\mathcal{C}}, L, R \leftarrow \mathcal{C}(\tau_i), \langle\rangle, \langle\rangle$

for $t \in \widehat{\mathcal{C}}$ **do**

$P \leftarrow$ SHORTEST-PATH(τ_i, σ, t)

$B \leftarrow$ BOTTLENECK(τ_i, P)

$R(t) \leftarrow \arg\max_{l \in \mathcal{L}} Q(\mathcal{S}(t), l).b \; \leq \min\{B;\; Q(\mathcal{S}(t), *).b\}$

$L(t) \leftarrow Q(\mathcal{S}(t), *).a - Q(\mathcal{S}(t), R[t]).a$

$W_i(e) \leftarrow W_i(e) - Q(\mathcal{S}(t), R(t)).b \quad \forall e \in P$

$\widehat{\mathcal{C}} \leftarrow \widehat{\mathcal{C}} - \{t\}$

return R, L

assigned to it, leaving more bandwidth to targets having more to send. The algorithm iterates over the targets t covered by the candidate input tree, sorted in ascending order based $Q(\mathcal{S}(t), *).b$, that is the load produced by the target t according to the task analyzer `A`^{2}`-TA`, belonging to the application scenario $\mathcal{S}(t)$ and at the minimum compression level (denoted by $*$) (**lines 1-3**). At each iteration a bottleneck bandwidth B for the target is computed. This quantity is the bottleneck capacity on the path from the source of flow t, to the destination σ. This value is influenced by the number of targets t shares this path with. The bandwidth allocation function can be thought as slight modification of the Depth First Search (DFS) (**line 5**). To derive the maximum quantity of load that can be transferred from the target t per unit time, we vary the compression level while remaining subject to the flow constraint (**line 6**). We store in the vector R the compression level for each target. We store the loss in accuracy for t subject to compression level $R(t)$, comparing the accuracy due to the best quality $Q((\mathcal{S}(t), *).a$ (**line 7**). The weights of the tree are updated considering the used bandwidth (**line 8**). Both the compression levels and the loss for each target are returned.

Cost of a Coverage

The cost of a connected coverage formation is parameterized by α. This exogenous parameter weights the importance given to the the accuracy of the tasks. Notice that the importance given to task accuracy opposes to the minimization of the number of UAVs employed. Therefore the cost is a linear combination of the average loss in accuracy, and the percentage of used UAVs to cover the new target in τ_i which was not present in the previous formation τ_{i-1}, the cost is computed as COST$_\alpha$:

$$\alpha \cdot \frac{\sum_{t \in \mathcal{C}(\tau_i)} L(t)}{|\mathcal{C}(\tau_i)|} + (1 - \alpha) \cdot \frac{|V_i - V_{i-1}| - 1}{|\mathcal{U}| - |V_\Psi \cup V_{i-1}|} \tag{2.13}$$

GREEDY-`A`^{2}`-TPP` Example Execution

Figure 2.5 shows an example of execution of GREEDY-`A`^{2}`-TPP`. The gray triangle is the edge server σ. The black dots and red squares represent the target and relay positions, respectively. Figure 2.5-a shows three temporary coverage τ_{temp}, each covering a different target. The cost of each of the

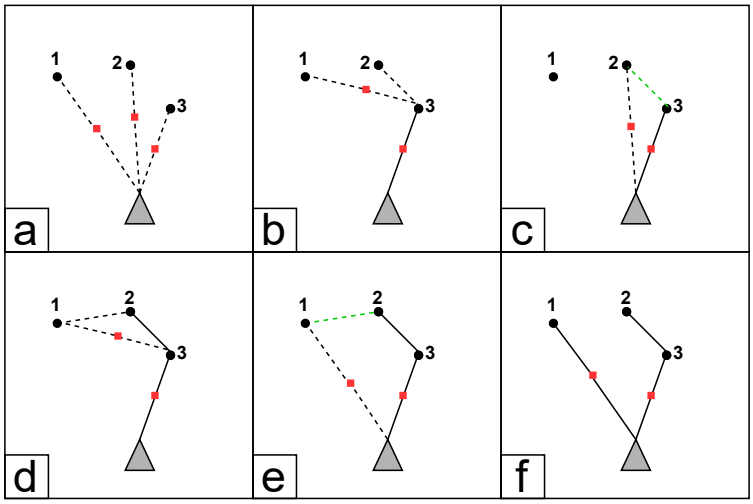


Figure 2.5: GREEDY-A^2-TPP algorithm example

coverage is compared (algo. 1, **line 7**). Say $\tau_{\text{temp}}\langle\sigma, t_3\rangle$ is the cheapest coverage among them, that is the tree covering σ and t_3. At the subsequent iteration shown in Figure 2.5-b, two Triangular Steiner Trees covering σ, t_3 and a new target among the remaining uncovered ones in $\mathcal{T} - \widehat{\mathcal{T}}$ (i.e., t_2 and t_1) are proposed. Say the tree $\tau_{\text{temp}}\langle\sigma, t_3, t_2\rangle$ is the cheapest coverage among them. In Figure 2.5-c the cost of $\tau_{\text{temp}}\langle\sigma, t_3, t_2\rangle$ is compared with a line of sight coverage $\tau_{\text{los}}\langle\sigma, t_2\rangle$. The cheapest coverage among the two becomes the one to grow from the subsequent iterations (algo. 1, **line 14**). Say the cheapest coverage among them is $\tau_{\text{temp}}\langle\sigma, t_3, t_2\rangle$. In Figure 2.5-d we see two grown versions of the tree covering t_1, whereas in Figure 2.5-e we see a line of sight coverage of t_1. Say that comparing the cost of $\tau_{\text{temp}}\langle\sigma, t_3, t_2, t_1\rangle$, and $\tau_{\text{los}}\langle\sigma, t_1\rangle$, the cheapest is the line-of-sight version. The tree $\tau_{\text{los}}\langle\sigma, t_1\rangle$ becomes the new tree to grow from the subsequent iterations. $\tau_{\text{temp}}\langle\sigma, t_3, t_2\rangle$ is archived in Ψ. There are no more targets to cover. $\tau_{\text{los}}\langle\sigma, t_1\rangle$ is archived in Ψ the algorithm stops returning Ψ.

Properties of GREEDY-A^2-TPP

Lemma 2.2.2. *Computing the compression level assignment has polynomial time complexity of* $O(|\mathcal{U}|^2)$.

Proof sketch. To measure the cost of a coverage tree τ_i, the set of targets in the tree $\mathcal{C}(\tau_i)$ is sorted by their expected transmission load in ascending order. Sorting requires $O(|\mathcal{T}| \log |\mathcal{T}|)$ time complexity. The for loop iterates over the targets in τ_i first computing the bottleneck bandwidth for t, having approximately the cost of a Depth First Search and Shortest Path, that is $O(\log |V_i|)$ for the tree. Iterating over the compression levels to find the highest resolution to fit the bandwidth has constant complexity $|\mathcal{L}|$ i.e., the cardinality of the discrete set of possible compression levels. Iterating over the edges of the path P to update the residual bandwidth has cost $O(\log |V_i|)$. Other assignments have evident constant complexity. By noticing that $|V_i| = O(|\mathcal{U}|)$ the overall time complexity of computing the cost of a coverage tree is $O(|\mathcal{T}| \log |\mathcal{U}|)$. The complexity further simplifies by considering $|\mathcal{T}| = O(|\mathcal{U}|)$, thus resulting in $O(|\mathcal{U}|^2)$. □

Theorem 2.2.3 (Time Complexity of GREEDY-A^2-TPP)**.** *GREEDY-A^2-TPP with input $\mathcal{T}$ targets sets has polynomial time complexity of* $O(|\mathcal{U}|^6)$.

Proof sketch. The while loop is executed, in the worst case, until all the targets in $\mathcal{T}$ are included

in the final solution Ψ. Within the while loop, a for loop iterates over the set of uncovered targets. For each of them a Triangular Steiner Tree t_{temp} is computed, and the time complexity is bounded by $O(|\mathcal{T}|^4)$ [172]. The cost of each tree is computed with complexity $O(|\mathcal{U}|^2)$ as shown in Lemma 2.2.2. Once the best candidate target to cover has been chosen, the Triangular Steiner Tree t_{los} of the shortest path path towards the target, and the relative cost c_{los} are computed. The time complexity to find a stripe can be considered constant in time. The overall time complexity of GREEDY-A^2-TPP is thus given by: $O(|\mathcal{T}|(|\mathcal{T}|(|\mathcal{T}|^4 + |\mathcal{U}|^2) + |\mathcal{U}|^2)) = O(|\mathcal{U}|^6)$. □

2.2.4 Performance Evaluation

We extensively evaluate A^2-UAV through simulation (Sect. 2.2.4) as well as real-world experiments (Sect. 2.2.4).

Evaluation Setup

Application. We consider a monitoring application where UAVs need to perform image classification or object detection tasks on *target* locations by sampling images at given frame rate (e.g., 24 frames per second (fps)). We adopt (i) *ResNet-50*, a CNN with 50 layers [89]; (ii) *ResNet-152*, an extended version with 152 layers [89]; (iii) *DenseNet* [98], which consists of a Dense Convolutional Network (i.e., each layer is connected to all the other layers in a feed-forward fashion); (iv) *MobileNet-V2* [168], a new neural architecture for mobile devices; (v) YoloV4, the state-of-the-art model for object detection. All the models were trained on the ImageNet database [55].

Scenarios. To emulate common scenarios for UAVs, we use the five scenarios described in Section 2.2.2, i.e., *Maritime, Search-and-Rescue, Wildlife, Tools, Pets*. We also design an *Urban* reconnaissance scenario including various objects, such as *wreck, fireboat, ambulance, police van, revolver, crate, packet, backpack, mountain bike, motor scooter*. To ensure repeatability of our experiments, we let the UAVs sample images from a labeled subset of ImageNet. Where not otherwise stated, each target location generates 500 tasks (images) uniformly sampled among these classes.

Metrics. We measure the *Percentage of Accomplished Tasks*, defined as the ratio between the number of successfully completed tasks (according to Definition 2.2.2) and the number of generated tasks. The accomplishment of a task is influenced by its de line Δ. In order to study the performance of A^2-TPP at varying application scenarios, we let Δ vary: low values represent delay critical applications (e.g., intrusion detection), whereas high values, delay tolerant ones (e.g., agriculture).

We also measure *Computational Time*, that is the time required by the algorithms to output a connected coverage formation and compression levels for the targets.

Comparison. We evaluate A^2-UAV through real-field experiments and simulation, considering both the optimal solution OPT-A^2-TPP and the greedy algorithm GREEDY-A^2-TPP, against STBA [149]. STBA is a state-of-the-art networking-based approach that is the closest to our work. STBA covers a set of targets while providing network connectivity to the edge server. To find a connected tree, STBA uses a node-weighted Steiner tree algorithm, which computes a set of Fermat points to place relays, and then computes a tree among the targets and the edge-server, minimizing the needed UAVs. To allow for a fair comparison, we enhance STBA with data compression in three variants: 1) H-STBA, which does not compress data, but uses the **H**ighest available quality for collected data ($l = 1$); 2) M-STBA which uses the **M**edium compression ($l = 50$); and 3) L-STBA

which uses an extreme compression ($l = 100$) resulting in the **L**owest data quality.

Simulation Results

We used the NS-3 network simulator [150], setting most of parameters in line with the devices used in our real-field experiments (e.g., WiFi interface 802.11n at 2.4 GHz), and testbed measured values (UAVs transmission range is $16m$, sensing radius $1m$, and maximum speed $5m/s$)[1]. The simulated area is a square of $500 \times 500m$, with an edge-server positioned in the center of the bottom border. The number of targets varies from 4 to 50, and the number of UAVs from 4 to 50.

Multiple Scenarios

Figures 2.6 illustrates the efficacy of our solution for different scenarios, reporting also the theoretical upper bound (blue dotted line). In the most challenging scenario, i.e., *Tools*, with a strict task deadline ($\Delta = 0.1$sec) and DenseNet-201 DL model, **OPT-A^2-TPP completes** 41% **of tasks, with an improvement of** 52% **respect the best STBA variant**, i.e., H-STBA, which completes less than 27% of tasks. The theoretical upper-bound for DenseNet-201 in the same scenario is 60%, meaning that under ideal network conditions of zero latency and no compression, the DL model would correctly classify only 60% of the tasks (Figure 2.4.a show the complexity of predicting tools images, even for un-compressed images).

In the case of *Pets and Maritime*, OPT-A^2-TPP reaches the highest percentage of accomplished tasks — 65% and 70% respectively — where the upper-bounds are 90% and 88%. **The improvement with respect to the best STBA variant, i.e., M-STBA, is** 55% **and** 50%. *Pets* require a compression level lower than $l = 50$ (see Figure 2.4.a) to achieve satisfactory performance, forcing both OPT-A^2-TPP and GREEDY-A^2-TPP to select a medium compression level, more similarly to M-STBA.

In *Wildlife* and *Search-and-Rescue* (SaR), the gap between both the A^2-TPP versions and STBA variants increases significantly. OPT-A^2-TPP and GREEDY-A^2-TPP complete respectively 60%, 63% and 49% and 54% of tasks, against 39% and 37% of M-STBA. The motivation behind this sharp improvement is the use of the A^2-TA, which understands that even high compressed images can achieve satisfactory performance. Therefore, both our solutions can achieve high accuracy with low network usage, executing the tasks within their deadline $\Delta = 0.1$ seconds. GREEDY-A^2-TPP completes 20% and 31% more tasks than M-STBA.

Urban Scenario

Figure 2.7 shows the performance in the *Urban* scenario as a function of task deadline $\Delta \in \{0.06, 0.07, 0.08, 0.09, 0.1\}$, when DenseNet-201 is employed.

OPT-A^2-TPP accomplishes tasks up to 72% in the case of $\Delta = 0.1$sec, while the best variant M-STBA achieves only 48% of tasks at the same Δ. **OPT-A^2-TPP accomplishes** 58% **more tasks than M-STBA with the tightest deadline**, as it adapts the compression of images to meet the latency constraint. The plot also confirms the performance of OPT-A^2-TPP that outperforms the network-based approaches (i.e., M-STBA) up to $45 - 50\%$. **GREEDY-A^2-TPP follows the OPT-A^2-TPP trend always remaining widely above the performance of STBA solutions.** Figure

[1]The code is available at https://github.com/flaat/AA-UAV

2.8 shows the percentage of accomplished tasks as function of the number of UAVs, with 6 targets randomly distributed in the area. We employ DenseNet-201, which achieves a maximum accuracy of 80%, and set $\Delta = 0.1$sec. Both OPT-A^2-TPP and GREEDY-A^2-TPP outperform the STBA variants, as they cover all the targets with only 8 UAVs. Conversely, the STBA variants require at least 10 UAVs to cover all the targets, and achieve lower performance. OPT-A^2-TPP covers $15-20\%$ more targets than STBA algorithms, in all the scenarios, completing 69% of tasks (using 10 UAVs), while the best variant M-STBA accomplishes only 42% of tasks with the same number of UAVs. We can notice how GREEDY-A^2-TPP performs better as quickly as number of UAVs grow reaching similar performance of OPT-A^2-TPP.

Robustness to Channel Errors

In Figure 2.9 we plot the percentage of accomplished tasks by varying the probability of channel error $\psi \in \{0, 0.05, 0.1, 0.15\}$, in a setting with 20 UAVs and 4 targets. Both OPT-A^2-TPP and GREEDY-A^2-TPP are the most robust algorithms, increasing their improvement with respect to STBA variants. **OPT-A^2-TPP completes up to 170% more tasks than the other approaches**. On the other hand, M-STBA and H-STBA experience severe delays and drastic performance reduction due to frequent TCP re-transmissions, which introduces additional data in the network, further overloading communication links.

Scalability

Figure 2.10 investigates the percentage of accomplished tasks in a scenario with 50 UAVs, varying the number of targets from 10 to 50. We do not include the OPT-A^2-TPP when the targets are more than 20, due to prohibitive computational time. This result underlies the huge benefit introduced by the polynomial time solution GREEDY-A^2-TPP, which scales gracefully when the problem instance grows in complexity. The figure shows that GREEDY-A^2-TPP has near optimal performance with 10 targets, accomplishing 63% of the tasks, while L-STBA accomplishes only 38% of them. All the algorithms have a slightly decreasing trend as the number of targets increases, as the UAVs have to offload more tasks with possible network congestion and missed deadlines. The STBA variants quickly drop their performance due to congestion and long delays, while GREEDY-A^2-TPP is able to keep satisfactory performance around 50%, trading off compression and accuracy to cover all targets and offload their data. With 50 targets GREEDY-A^2-TPP accomplished 5 times the tasks of the best STBA variant. Finally, in Figure 2.11 we investigate the computational time. We restrict the time to a maximum of 5 hours (18000 seconds), and we consider no solutions after that time. We consider a general STBA instance without compression levels, as they do not affect the execution time. While OPT-A^2-TPP has very huge computational times even with 10 targets, **GREEDY-A^2-TPP is 15x faster than the STBA solutions**.

Experimental Testbed Results

We now evaluate the performance through our experimental testbed. **The testbed is composed of 4 UAVs and an edge server with dedicated GPU**. We emulate missions with up to 4 targets. We run 10 experiments for each scenario and we average the results. Each UAV includes a DJI Mavic Air 2 drone, mounting a Raspberry PI 4 model B and a power bank, as shown in Figure

2.12. The on-board Raspberry PI, powered by the power bank is used to generate and pre-process tasks, and to offload them to the edge according to the optimization plan. For repeatability and to emulate different scenarios, we sample images from the ImageNet dataset [166].

The edge server is a Jetson Nano board, used to run the DL models and execute tasks. It mounts a Raspberry PI for computation and communication. TCP links are established for reliable connectivity. Considering the limited capabilities of the edge server, we execute only ResNet-50 and MobileNet-V2 on the Jetson Nano, which have approximately 0.03 seconds of inference time [151]. For DensNet-201, ResNet-152, and YoloV4, we used a laptop with an NVIDIA RTX-2060 Graphics Processing Unit (GPU). In the experiments, we consider up to 4 targets placed at around 15 meters from the edge. Table 2.2 reports the experimental settings in the *Urban* scenario. The first set of experiments evaluates the impact of increasing the number of targets (from 1 to 4), with MobileNet-V2. Figure 2.13 shows the percentage of accomplished tasks at the edge-server with a task deadline of $\Delta = 0.4$sec. The plot shows that the OPT-A^2-TPP finds the best trade-off between accuracy and data compression. It completes more than 67% of the tasks, independently of the number of targets. This is close to the maximum performance achievable with the DL model (i.e., 78%), represented by the blue horizontal line. **GREEDY-A^2-TPP instead reaches up to 57% accomplished task, with a 20% average improvement over the best STBA version** (M-STBA). Conversely, the best STBA variant (i.e., M-STBA) does not complete more than 46% of tasks, independently of the number of targets. In particular, with 2 targets, all STBA variants perform very poorly, completing less than 30% of tasks. The superiority of A^2-UAV in both the approaches (Opt and Greedy) is confirmed by results on the percentage of accomplished tasks by varying the deadline $\Delta \in [0.1, 0.5]$ (see Figure 2.14). Opt-A^2-TPP reaches an improvement over the percentage of executed tasks with respect to M-STBA up to 76% when $\Delta = 0.3$sec. The GREEDY-A^2-TPP approach instead improves M-STBA results (when Δ=0.3sec) around 50% upholding our intuition. We investigated the performance of GREEDY-A^2-TPP and OPT-A^2-TPP also when other DL models are applied. Table 2.3 summarizes the results in the case of $\Delta = 0.3$sec and 4 targets, for ResNet50, MobileNet-V2 (executed on the Jetson Nano) and ResNet152, DenseNet201 and YoloV4 (executed on a laptop with a dedicated GPU). The results show that both our solutions outperforms all STBA variants independently of the applied model. In particular, with DenseNet201 OPT-A^2-TPP has the best performance.

DL Model	OPT-A^2-TPP	GREEDY-A^2-TPP	L-STBA	M-STBA	H-STBA
ResNet50	**66.64**	62.88	25.42	36.14	21.86
ResNet152	**67.89**	65.27	29.93	37.96	24.70
DenseNet201	**70.17**	68.33	32.92	41.87	26.99
MobileNet-v2	**69.29**	57.36	18.37	38.94	21.91
YoloV4	**59.32**	51.3	15.22	31.52	17.18

Table 2.3: Percentage of Completed Tasks, $\Delta = 0.3$sec

2.2.5 A further improvements to make the network resilient: Adaptive A^2-TPP

In this section we present *adaptive* A^2-TPP, a seamless integration to A^2-TPP to ensure network's resilience even in the face of unpredictable events. This version of the protocol prioritizes adaptability

Algorithm 3 ADAPTIVE $\mathtt{A}^2$-TPP

Input: $\mathcal{U}_t$: sets with the available UAVs at time t; Ψ_{t-1}, Ψ_t the connected coverage formation at time $t-1$ and t resp.; $\mathcal{T}_t$: set of targets at time t.
Output: Ψ a new connected coverage formation

$\Psi_t, R_t \leftarrow$ GREEDY $\mathtt{A}^2$-TPP$(\mathcal{U}_t, \mathcal{T}_t)$
$\mathcal{M} \leftarrow$ MINMATCHING(Ψ_t, Ψ_{t-1})

return $\Psi_t, R_t, \mathcal{M}$

to accommodate changing mission conditions, including variations in UAVs and targets availability.

Specifically it is designed to handle UAVs failures and the addition of new UAVs to the fleet. Given the inherent susceptibility of UAV networks to node failures, caused by factors such as limited battery or unpredictable adverse environmental conditions, the protocol is designed to swiftly identify affected nodes and restore the network to its original operational integrity.

Moreover, the protocol takes into account the dynamic nature of targets, which may be added, removed, or change positions during a mission. The *adaptive* $\mathtt{A}^2$-TPP ensures that the network remains optimized and ready for evolving missions, guaranteeing resilience in the face of unforeseen circumstances.

Algorithm Overview

Adaptive $\mathtt{A}^2$-TPP is shown in Algorithm 3 and is triggered at time t when one of the following five events occurs: (i) *UAV failure*, a UAV of the network fails; (ii) *UAV provisioning* a new UAV is added to the fleet (iii) *target removal* a target has no more the need to be monitored; (iv) *target add* a new target was added; (v) *target move*, a target changed its position. We denote by e the reference to the occurred event.

Let $\mathcal{U}_t$ be the sets with the available UAVs at time t. Let Ψ_{t-1}, Ψ_t be the connected coverage formations at time $t-1$ and t respectively, and finally $\mathcal{T}_t$ the sets of targets at time t. Upon trigger event e at time t. The algorithm computes a new connected coverage formation and compression level assignment with the UAVs and targets available at that time (**line 1**). Then the algorithm determines which UAVs, formerly covering Ψ_{t-1}, have to move towards the new formation Ψ_t *minimizing the worst migration time*. Such cost represents the time for the last UAV to connect to the new formation. We implement such migration using a procedure called *MIN-MATCHING*. In particular, we reduce our problem to what in the literature is known as the Bottleneck Matching Problem (BMP) for bipartite graphs [**?**].

We recall that BMP takes a bipartite graph $G = (V_0, V_1, E)$, and a non-negative edge cost function $c(e) > 0$, $\forall e \in E$, and finds a perfect matching $M \in M_p$ where M_p is the set of subsets of E of cardinality $|V_0| = |V_1|$, of minimal cost, i.e., $\min_{M \in M_p} \max_{e \in M} c(e)$. In our MIN-MATCHING reduction, nodes in V_0 and V_1 are the nodes of the connected formations Ψ_{t-1} and Ψ_t respectively, representing the position of UAVs in the formation. In both the sets, the nodes representing the position of spare UAVs (i.e., UAVs that are not used for granting connectivity) are modeled as duplicated nodes of the base station. The bipartite graph is complete and the cost function of the edges $c((u, w)) > 0$, $\forall\, e : (u, w) \in E$ represents the time to fly from position u to w.

This reduction allows to minimize the effort needed to migrate from one connected formation to another. The output of the matching algorithm is a map $\mathcal{M}$ from every UAV in the formation Ψ_{t-1} to the new Ψ_t with the least effort in terms of travel time for the fleet (**line 2**). The UAVs on Ψ_{t-1} efficiently migrate towards Ψ_t according to the map $\mathcal{M}$ and compression level R_t.

Adaptive A^2-TPP

To test Adaptive A^2-TPP we chose to use the same scenario shown in Figure **??** described in the previous section. To evaluate the impact of a *UAV failure* event, we measure the impact over the *tasks executed per second*. The failure event affects one of the four UAVs, thus, the entire network must reorganize in order to communicate again with the base station. For a fair comparison, we applied Algorithm 3 also to STBA, in order to compare the results even though it does not encompass a recovery mechanism.

In Figure 2.15 shows the variation of executed tasks over time for all the considered algorithms. When a *UAV failure* event happens (vertical red bar in the figure), communication is dropped and so tasks executions. As the figure shows, our algorithm enables faster recovery when compared to the other approaches. In particular adaptive A^2-TPP rebuilds the configuration 5 seconds earlier with respect to the second best L-STBA. The average time needed to compute the new topology for our solution is 0.00224 ± 0.00043 seconds and is 0.00653 ± 0.00012 seconds for all STBA variants. We recall that a reduced computation time enables swift service restoration upon heterogeneous events during mission.

2.2.6 Conclusions

In this paper we proposed A^2-UAV, a novel *application-aware* optimization framework for reliable and effective DL task offloading in multi-hop UAVs networks.

For the first time, we considered the *accuracy* and *delay* requirements of the specific DNN tasks, to jointly optimize task assignment and offloading. Through extensive simulation, we demonstrated that A^2-UAV is able to deal with different network conditions, maximizing the application performance at the edge. A^2-UAV outperforms existing approaches, getting and average improvement w.r.t. the state-of-the-art algorithm of 38%. We finally validated our solution through real-field experiments, considering four DJI Mavic Air 2 UAVs and a Jetson Nano board as edge server. We share datasets and code with the research community to allow reproducibility. Our framework also incorporates a re-optimization scheme to handle any node failures or changes in the target points. Through extensive simulation and real testbed experiments, we demonstrated that A^2-UAV is able to deal with different network conditions and node failures, maximizing the application performance at the edge. A^2-UAV outperforms existing approaches, getting and average improvement w.r.t. the state-of-the-art algorithm of 38%. We share datasets and code with the research community to allow reproducibility.

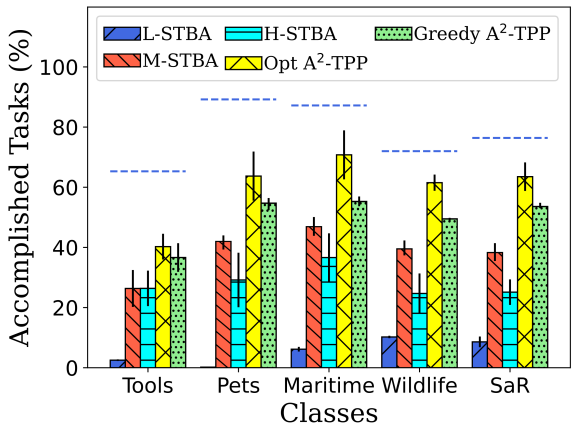


Figure 2.6: Accomplished tasks (%), $\Delta = 0.1$sec

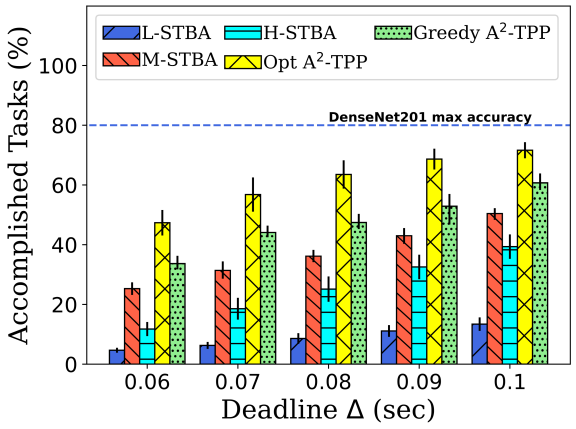


Figure 2.7: Accomplished tasks (%) at increasing of Δ

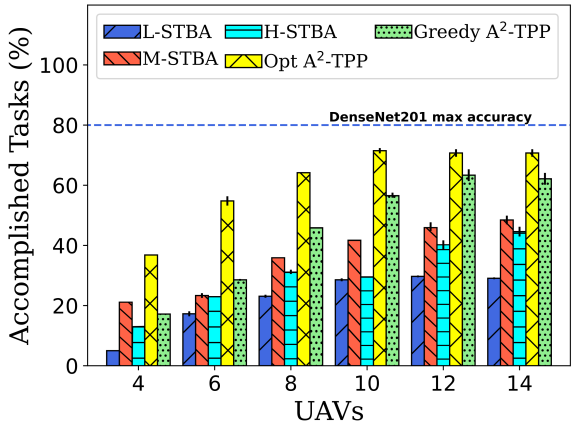


Figure 2.8: Accomplished tasks (%) with 6 Targets, $\Delta = 0.1$sec

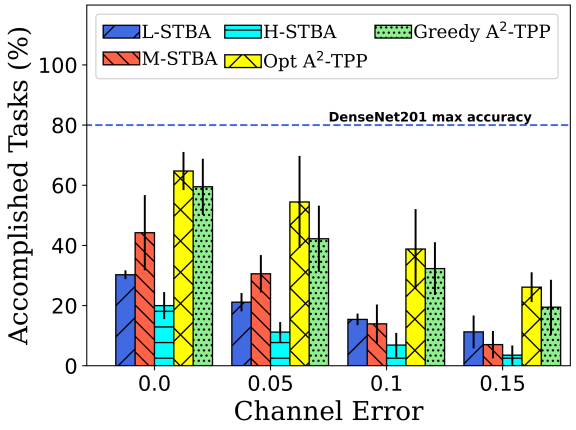


Figure 2.9: Accomplished tasks (%), 4 targets, $\Delta = 0.1$sec

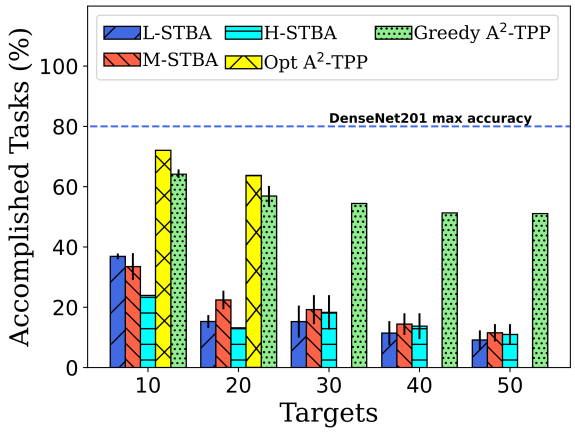


Figure 2.10: Accomplished tasks (%), increasing targets

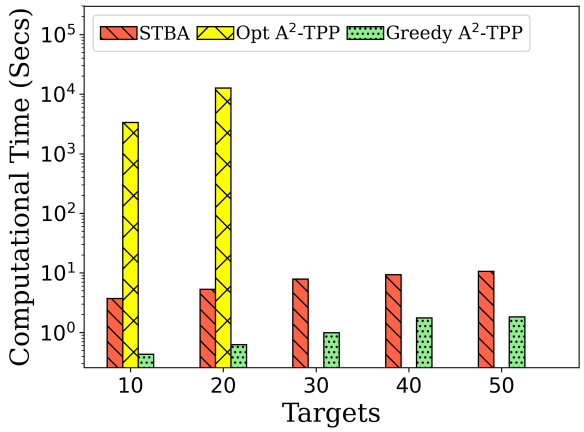


Figure 2.11: Computational time (sec)

Field	*****
Time	9:00-18:30 a.m.
Temperature	+4 - 15°C
Wind Speed	0.0 to 4.3 m/s
Field Size	65x35(meters)
Nr. Of UAVs	4
Nr. Of Targets	Variable (from 1 to 4)
Humidity	60% - 77%
AMSL	2 meters

Table 2.2: Experimental Setting

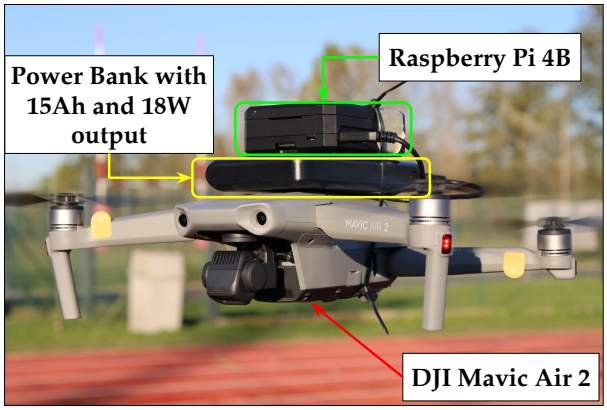


Figure 2.12: UAV implementation.

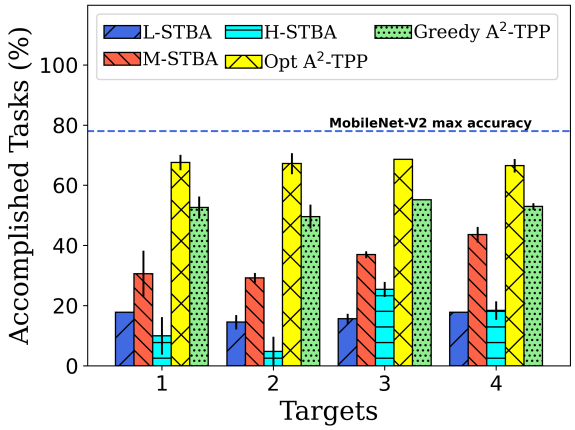


Figure 2.13: Accomplished Tasks (%) at increasing targets, MobileNet-V2, $\Delta = 0.4$sec

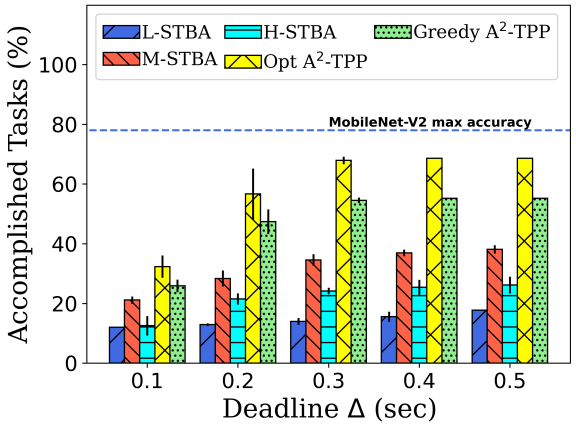


Figure 2.14: Accomplished Tasks (%) at increasing of Δ, using 3 targets and MobileNet-V2

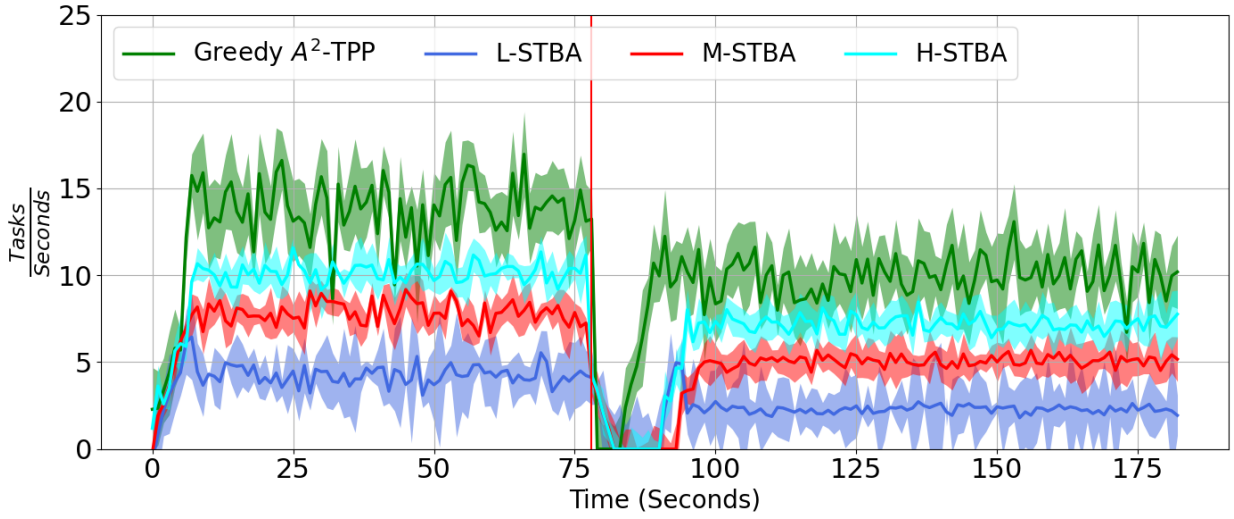


Figure 2.15: Task execution ratio during mission. The vertical red line represents the a *UAV failure* event.

2.3 Graphs in periodic communication

In the previous section, we addressed the problem of semantic communication to enhance task execution at the edge server in UAV networks using graph-based approaches. In this section, we utilize graph heuristics to enhance periodic communication in dynamic UAV networks.

The past few years have witnessed an unprecedented proliferation of Unmanned Aerial Vehicles (UAVs) in people's daily lives. Thanks to their increasing capabilities and moderate cost, the use of UAVs has expanded to countless application scenarios, from last-mile delivery to precision agriculture, border patrolling, and many others [5, 19, 120].

The use of small multi-rotor UAVs in swarm formations has garnered particular attention due to their ease of assembly and quick deployability. They are the technology of reference in monitoring critical environments where the existing communication infrastructure is damaged or knocked off. Due to their power limitations, however, small UAVs cannot rely on high data-rate long-range communications (e.g., 5G) to offload the collected data.

In such scenarios, UAVs are typically required to navigate to the base station every time they need to offload monitoring data. As a consequence, they suffer from rapid battery depletion, which limits the network's monitoring capabilities and shortens its lifespan.

One way to address these limitations is to allow the UAVs of the swarm to exploit multi-hop communication paths to deliver data to the base station. However, when deployed over large areas, the monitoring swarm may experience temporary connectivity losses. The necessity of establishing multi-hop paths for data delivery highlights the need to address the connectivity problem and design efficient routing algorithms to deliver data packets through the swarm network. To jointly tackle both the connectivity and routing issues, we propose `Stop & Route`, an innovative algorithm that periodically creates connected formations of the UAV network. Once a connected formation (STOP action) is attained, the UAVs can deliver their monitored data to the base station through an efficiently selected routing path (ROUTE action).

`Stop & Route` enables the monitoring swarm to move quickly as needed, forming a connected network topology to deliver data. Thanks to its movement efficiency, it enables the network to spare communication time in favor of more device availability for monitoring and sensing tasks.

The objective of `Stop & Route` is to build the most efficient formations starting from any monitoring deployment, so as to maximize the minimum residual energy of the UAVs of the swarm, and thereby maximizing the network availability for the mission tasks and its lifetime.

Throughout the paper, we refer to the formation obtained during a stop phase as a *connected offloading formation*. `Stop & Route` guides the UAVs to build connected offloading formations in two modalities. The first allows for centralized coordination of the swarm. It is based on the assumption that each UAV is equipped with a long-range transmitter whose data rate, though very limited, is sufficient to establish a control channel with the base station. Through the control channel, the UAVs send and receive the necessary messages to periodically construct the desired connected formations in a centralized manner. In centralized scenarios, the UAVs form a connected formation based on the reception of a trigger message from the base station, hereafter referred to as the *offloading trigger*.

The second modality, instead, considers a distributed execution of the algorithm activities. It relaxes the assumptions on communication range and considers UAVs that are only equipped with

short-range communication devices, i.e., the swarm works without any control channel. In this scenario, the network devices only rely on a local view of the achieved deployment while moving to build a connected formation. To coordinate their movements and schedule the algorithm phases, the devices adopt a loosely synchronized clock.

The `Stop & Route` algorithm provides an iterative execution of the following activities. The first is the mission activity: the UAVs execute their mission tasks, moving along the area to gather sensing data related to targets of interest.

Examples of sensing applications include monitoring of large infrastructures such as roads, railways, and bridges to gather data about the condition of these structures. Or post-disaster: like earthquakes, hurricanes, or flooding, where they can be used to monitor the damage, and help with post-disaster recovery.

Upon reception of the data offloading trigger (centralized execution) or periodically (distributed execution), the UAVs execute a second activity, i.e., they build a connected offloading formation to establish communication paths with the base station. Once connected, the UAVs deliver the monitored data and resume their monitoring tasks.

The repeated execution of the two activities continues until the end of the network lifetime, that is, when the battery of the last available drone runs out of power and it must be recharged at the recharging station.

2.3.1 Related Work

The problem of exploring an area of interest with sensor networks gained a lot of attention in the literature. To minimize the accumulated age of information during the monitoring activity, the problem is augmented with communication requirements. We focus on exploration pursued with multi-robots, in particular, UAVs in communication-restricted contexts. Most works in the literature focus on terrestrial robots, and they cannot be directly applied to UAV squads. The latter have unconstrained mobility, different energetic constraints, and requirements. For example, the amount of energy spent by UAVs in a stationary hovering position is not negligible, unlike that of terrestrial robots. In this work, we focus on event-based, periodic connectivity, which requires UAVs to offload their discoveries to the base station upon periodic offloading triggers. To the best of our knowledge, none of the works in the state-of-the-art addressed the same problem for UAV networks.

In [7], a taxonomy of the *communication requirements* in multi-robot exploration systems is proposed. Some exploration missions require *continuous connectivity*, that is, the squad operates information gathering while remaining in partial or global communication, typically including the base station. An example of this kind is that of real-time streaming of monitoring video for human inspection at the base station. [20, 59, 123, 135, 162, 169, 170, 179]. Other works require *event-based connectivity* in which UAVs regain partial or global connection with the squad, periodically or triggered by specific events [11, 14, 50, 78, 88, 94, 96]. Some approaches exploit temporary pairwise connectivity, intermittently present among devices [14, 96].

Dutta et al. [59] consider the problem of information collected from a polygonal environment using a multi-robot system, subject to continuous connectivity constraints. Banfi et al. [13] design exploration strategies that enable robots to coordinate with teammates to form a network, satisfying recurrent connectivity constraints, which require data to be shared with the base station when new observations are made at assigned locations. Banfi et al. [12] provide a solution by modeling the

signal's distribution with a Gaussian Process, exploiting online different sensing strategies to coordinate and guide robots during data acquisition. Reich et al. [162] explore distributed mechanisms for maintaining the physical layer connectivity of a mobile wireless network while still permitting significant area coverage. Bartolini et al. [17] consider critical scenarios with a squad of UAVs requiring to autonomously inspect an area of interest under uncertainty of time and location of target events ensuring maximum coverage of event monitoring with minimum average inspection delay. Banfi et al [11] study effective multi-robot exploration strategies under recurrent connectivity by considering a centralized and asynchronous planning framework. They provide both the problem formalization of selecting the optimal set of locations robots should reach, the exact formulation to solve it and an approximation algorithm to obtain efficient solutions with a bounded loss of optimality. Tan et al. [179] present two heuristics based on virtual forces, with the goal to maximize sensing coverage and also guarantee continuous connectivity at the cost of a small moving distance. The heuristics do not need any knowledge of the field layout. The latter two works are used as a comparison in the experimental analysis.

2.3.2 System Model and Assumptions

We consider the scenario in which a swarm of UAVs denoted with the set $\mathcal{U} = \{u_1, \ldots, u_m\}$, is cooperatively pursuing information-gathering over an area of interest. We denote with $d_{\max}$ the longest distance between two points within the area. The control base station, denoted by σ, is placed within the area for squad coordination and acts like a sink for the information gathered from the squad. Every UAV is equipped with a GPS module and sensors to bypass obstacles in the area with ease. We denote with $p(u) \in \mathbb{R}^2$ the position of UAV u in space, analogously $p(\sigma)$ for the base station. We further denote with $b_u(t)$ the available energy of UAV u at time t, and with β_0, β_1 the energy consumption factor per unit of time spent on hovering by the UAVs and per unit of traveled distance, respectively. We denote by $h(u)$ the time spent hovering by a UAV u.

As we consider harsh environments, we cannot assume the presence of any reliable long-range communication infrastructure, such as 5G. Therefore, to send sensed data to the base station, the UAVs use short-range radio transceivers (e.g., WiFi) over multi-hop routes. This communication channel, called ***primary channel***, allows communication only between neighbor nodes, ensuring a high data rate and low energy consumption. A disk model is adopted, we denote with r^{com} the transmission range of UAVs.

In some cases, we also consider the availability of a low data-rate long-range communication technology, such as LoRa, that allows the transmission of control packets to the entire network. We call ***control channel*** this long-range communication technology.

Both the UAVs and the base station use this channel to transmit simple commands and coordinate with each other. To guarantee periodic information offloading at the base station, the UAVs *stop* their monitoring activity when triggered by the base station through the control channel (*offloading trigger*) and gather towards it to *route* sensing data, creating a connected offloading formation.

Definition 2.3.1 (UAVs Communication)**.** *A pair of UAVs u_i, u_j is said to be in communication* ***com****$(p(u_i), p(u_j))$ if and only if,*

1. $\|p(u_i) - p(u_j)\| \leq r^{\text{com}}$ *or*
2. $\exists\, u_k$, ***com****$(p(u_i), p(u_k))$ and* ***com****$(p(u_k), p(u_j))$*

Definition 2.3.2 (Connected Offloading Formation)**.** *A connected offloading formation $f : \mathbb{R}^2 \to \mathbb{R}^2$ is a function mapping the coordinates of every UAV $u \in D \subseteq \mathcal{U}$ to new coordinates $f(p(u))$ such that* ***com****$(f(p(u)), p(\sigma))$.*

Definition 2.3.3 (Optimal Connected Offloading Formation)**.** *An optimal connected offloading formation $f^* : \mathbb{R}^2 \to \mathbb{R}^2$ upon an offloading trigger at time t, is a coordinates map such that,*

$$\max \min_{u \in \mathcal{U}} \; b_u(t) - \beta_0 h(u) - \beta_1 \|p(u) - f^*(p(u))\| \tag{2.14}$$

it maximizes the minimum residual energy for the UAVs.

Notice that the UAVs in the swarm reach their target position $f(p(u))$ at different times, depending on their starting position $p(u)$ and traveling speed. Thus, the hovering time spent by all the UAVs waiting for the last UAV to join the formation has an impact on the solution and will be taken into account in the optimal formulation presented in the next section.

We study two different ways to build connected offloading formations. First, we consider a ***centralized*** approach in which the UAVs and the base station are equipped with both the primary and the control channels. The base station and the UAVs use the control channel to build the offloading formation: the base station broadcasts the stop command, the UAVs reply by sending their positions, and the base station in turn sends the connected offloading formation.

We then consider a ***distributed*** approach in which only the primary channel is available. In this case, the UAVs have only a partial view of the network (i.e., the neighborhood), and use their local information to derive a connected offloading formation in a distributed way.

Figure **??**.a, shows a squad of 5 UAVs performing a monitoring mission over an area of interest. They are all connected with the base station through the control channel. Figure **??**.b shows neighbor UAVs communicating over the primary channel. Figure **??**.c shows a possible offloading formation, in which each UAV has an offloading route to the base station. Data offloading takes place over the primary channel.

2.3.3 Optimal Problem Formulation

To obtain optimal connected offloading formations we propose a solution based on linear programming. We denote by $\hat{p}(u)$ the decision variable representing the target position of UAV u to generate a connected offloading formation. We first constraint the new position of the UAV to stay within the area of interest,

$$\hat{p}(u) \text{ in AOI } \forall u \in \mathcal{U} \tag{2.15}$$

We constrain the points that are reachable with the available energy as:

$$\|p(u) - \hat{p}(u)\| \cdot \beta_1 \leq b_u(t), \;\; \forall u \in \mathcal{U}. \tag{2.16}$$

Then, we add a constraint to impose the network connectivity of the new formation, to allow communication among all the UAVs and the base station. Let $\mathcal{U}^+ = \mathcal{U} \cup \{\sigma\}$. To constrain the connectivity between the UAVs and the base station, we consider that each UAV generates a unit of flow and that the base station wants to receive m units. Let $\hat{s}_{ij}$ be the decision variable representing the information flow from UAV u_i to u_j. We constrain the base station to receive m units of flow as follows:

$$\sum_{i \in \mathcal{U}^+} \hat{s}_{i\sigma} = m \tag{2.17}$$

and to produce 0 units of flow as follows:

$$\sum_{i \in \mathcal{U}^+} \hat{s}_{\sigma i} \leq 0 \tag{2.18}$$

Then, we impose that UAVs generate a flow only towards their neighbors. We add an auxiliary binary variable γ_{ij} s.t. $\gamma_{ij} = 1 \iff \|\hat{p}(u_i) - \hat{p}(u_j)\| \leq r^{\text{com}}$, defined as follows,

$$\begin{aligned} \|\hat{p}(u_i) - \hat{p}(u_j)\| - M \cdot (1 - \gamma_{ij}) &\leq r^{\text{com}} \\ \|\hat{p}(u_i) - \hat{p}(u_j)\| + M \cdot \gamma_{ij} &\geq r^{\text{com}} \\ \gamma_{ij} &\in \{0, 1\} \end{aligned} \tag{2.19}$$

for each i, j in $\mathcal{U}^+$ when M is an upper bound variable.
Then, we impose that the flow is given by:

$$\sum_{j \in \mathcal{U}^+} \hat{s}_{ij} \leq \hat{\gamma}_{ij} \cdot m, \ \forall i \in \mathcal{U} \tag{2.20}$$

an edge can have a flow only if the path length between i and j is less than the communication range. Finally, to enforce connectivity, we impose flow conservation through the following constraint:

$$\sum_{j \in \mathcal{U}^+} \hat{s}_{ij} = \sum_{j \in \mathcal{U}^+} \hat{s}_{ji}, \ \forall i \in \mathcal{U} \tag{2.21}$$

which imposes that, for each out-edge from i the flow is increased by the UAV, only if i and j otherwise. These three constraints enable the formation's connectivity.

Let w be the variable representing the maximum travel time to reach the target point, defined as $w \geq \|p(u) - \hat{p}(u)\|/v \ \forall u \in \mathcal{U}$ with v the uniform speed of the UAVs. Thus the hovering time for a UAV is defined as

$$h(u) = w - \|p(u) - \hat{p}(u)\|/v \tag{2.22}$$

We finally constraint ω to be least residual energy as:

$$\omega \leq \ b_u(t) - \beta_0 h(u) - \beta_1 \|p(u) - \hat{p}(u)\|, \ \ \forall u \in \mathcal{U} \tag{2.23}$$

The objective function is as follows:

$$\max \omega \tag{2.24}$$

We approximate the Euclidean distance, which is not linear to compute, using the approximation

function proposed by [213].

2.3.4 Centralized Heuristics

The MILP formulation presented in the previous section is solvable in polynomial time. However, in many scenarios, the computational time for an optimal connected offloading formation is unacceptable. Figure 2.17 shows an improvement in computational time that goes from 3 to 4 orders of magnitude for the heuristics w.r.t. the MILP solution. This problem is exacerbated by the fact that during a mission the UAVs are called to offload data several times, thus requiring the base station to calculate a new optimal formation each time. After sending their positions to the base station the UAVs have to wait (in hovering) the time needed by the base station to calculate the optimal formation online. This amount of time not only has a catastrophic impact on the delay to build the offloading formation, but also requires substantial energy consumption due to UAVs hovering. To face this problem, we propose efficient heuristics and evaluate their computational complexity.

Greedy Assignment Algorithm

We first propose the Greedy Assignment Algorithm (GAA). Algorithm 4 illustrates GAA formally. It considers the set of UAVs, $D(t) = \{u \in \mathcal{U} \mid b_u(t) > 0\}$ having a sufficient residual battery to go back to the base station for recharging. UAVs that deplete their battery, go back to the base station and are no longer available for the monitoring mission. We consider a regular hexagonal tiling of the area of interest and refer to the centers of the hexagons as rendezvous points in $\mathcal{R} \subset \mathbb{R}^2$. This set is such that taken two rendezvous points belonging to any two adjacent tiles $r_1, r_2 \in \mathcal{R}$, $\boldsymbol{com}(r_1, r_2)$. The main idea of GAA is to iteratively select for each UAV a rendezvous point that allows connection with the base station and has the least impact on the battery. First, we define $\Theta(D(t), R)$ as the residual energy of the least energetic UAV of those in $D(t)$ available at time t, after reaching the rendezvous point in R, maximizing the residual energy, formally:

$$\Theta(D(t), R) = \min_{u \in D} \max_{r \in R} b_u(t) - \|p(u) - p(r)\| \cdot \beta_1 \tag{2.25}$$

Let $X \subseteq \mathcal{R}$ be a set of rendezvous points. We denote by $\Gamma(X) \subseteq \mathcal{R}$ the set of rendezvous points adjacent to those in the set X. We define $F(X) \subseteq \mathcal{R}$ the frontier of X, to the set of rendezvous points lying on the perimeter of the convex hull induced by the rendezvous points in X. GAA iteratively builds the connected offloading formation f, by mapping at each while loop iteration a UAV u^* to a target rendezvous point r^*, chosen in the neighborhood of connected rendezvous points.

Proposition 2.3.1. *The set of candidate rendezvous points at each iteration is upper bounded by m.*

Proof. The number of rendezvous points in R is upper bounded by the cardinality of the union of the adjacent hexagonal tiles of the UAVs positions and of the base station, that is $\mathcal{O}(6(m+1)) = \mathcal{O}(m)$.

Theorem 2.3.1. *GAA computational complexity is $\mathcal{O}(m^5)$.*

Proof. Referring to Algorithm 4, the while loop iterates until all the UAVs are not mapped to a target rendezvous point. A map happens once at each iteration, the while loop lasts for $\mathcal{O}(m)$

Algorithm 4 Greedy Assignment Algorithm (GAA)

Input: A set of UAVs $D(t) \subseteq \mathcal{U}$ under offloading trigger at time t
Output: An Connected Offloading Formation f

$f(p(u)) \leftarrow \emptyset \; \forall u \in D(t)$
$R \leftarrow F(\Gamma(\{\sigma\}))$
while $\exists u \in D(t)$ *s.t.* $f(p(u)) = \emptyset$ **do**
  $(u^*, r^*) \leftarrow \arg \max_{r \in R,\, u \in D(t)} \Theta(D(t) - \{u\}, F(R \cup \Gamma(\{r\})))$
  $R \leftarrow R \cup F(\Gamma(r^*))$
  $D(t) \leftarrow D(t)/\{u^*\}$
  $f(p(u^*)) \leftarrow r^*$
return f

iterations. The Θ function is called for every candidate rendezvous point among the $\mathcal{O}(m)$ in the growing frontier R (see Proposition 2.3.1), and for every UAV among the $\mathcal{O}(m)$. This function has $\mathcal{O}(m^2)$ computational complexity. It follows that the overall computational complexity of GAA is polynomial in the number of UAVs and is $\mathcal{O}(m^5)$. □

Tree Contraction Algorithm

To lift the assumption of map discretization and improve time complexity, we propose the Tree Contraction Algorithm (TCA). Algorithm 5 illustrates TCA formally. It consists in contracting the edges of a tree, by moving the UAVs along the most convenient route toward the base station and allowing communication through the primary channel with the whole squad. The algorithm computes the Delaunay graph $G = (P, E, W)$, derived from the points $p(u)$ $\forall u \in D(t) \cup \{\sigma\}$, with $W : (n_1, n_2) \in E \rightarrow \|p(n_1) - p(n_2)\|$. Then the algorithm derives from G the Euclidean Minimum Spanning Tree (EMST) $T = (P, E')$ which inherits from the Delaunay graph the property of geometric k-spanners. A geometric k-spanner is a graph in which all the path weights are upper bounded by k times the spatial distance between the path endpoints. As Theorem 2.3.2 illustrates, this property is useful to have an upper bound on the distance traveled by the UAVs [136]. It follows an edges contraction phase, where the edges of T are made shorter to match the connection requirements through the primary channel. The shrinking process starts from the root of the tree T (representing the base station) to the leaves and proceeds iterating over the nodes of the tree sorted in a Breath First Search (BFS) fashion. For every UAV in $r \in P$, representing UAVs a new target position is computed. The new position is relative to the parent node in the tree parent$(r) \in P$. In particular the angle α between r and parent(r) is used to determine the new position the UAV should reach to gain connectivity, i.e., the coordinates of the parent plus the dimension of the minimum communication range between r and its parent. Finally, $\gamma \in (0, 1]$ can be used to uniformly reduce communication range to guarantee a much stabler communication.

Theorem 2.3.2. *The distance traveled by a UAV $u \in D(t)$ to connect to the base station σ according to a connected offloading formation produced by TCA is upper bounded by* $1.998 \cdot \|p(u) - p(\sigma)\| - \gamma r^{\text{com}} \cdot \rho(u, \sigma, T)$.

Proof. The EMST T computed by TCA guarantees that the path linking each node with the base station is the shortest. The EMST derived from the Delaunay graph is a geometric spanner. In such graphs the maximum distance between two nodes n_0, n_1 is no greater than $1.998 \cdot \|n_0 - n_1\|$ [212]. This property is maintained in the EMST. In the TCA contraction phase of the EMST, any path from u to σ having $\rho(u, \sigma, T)$ number of hops, is contracted in one of length $\gamma r^{\text{com}} \cdot \rho(u, \sigma, T)$

Algorithm 5 Tree Contraction Algorithm (TCA)

Input: A set of UAVs $D(t) \subseteq \mathcal{U}$ under offloading trigger at time t
Output: An Connected Offloading Formation f

$P \leftarrow \{p(u)\ \forall u \in D(t) \cup \{\sigma\}\}$
$G(P, E, W) \leftarrow \texttt{DELAUNAY}(P)$
$T(P, E') \leftarrow \texttt{KRUSKAL}(G)$
$P \leftarrow \texttt{BFS}(T)$
$f(p(u)) \leftarrow p(u)\ \forall u \in D(t)$

for $r \in P$ **do**
  if $r \neq \sigma$ **then**
    if $\|p(r) - p(\text{parent}(r))\| > \gamma r^{\text{com}}$ **then**
      $\alpha \leftarrow \arctan\left(\frac{\Delta_Y(r,\text{parent}(r))}{\Delta_X(r,\text{parent}(r))}\right)$
      $f(p(r)) \leftarrow p(\text{parent}(r)) + \gamma r^{\text{com}} \cdot \langle \cos(\alpha), \sin(\alpha) \rangle$
return f

units of distance. Since we place UAVs belonging to the same path exactly at distance γr^{com} the one from the other, that distance can be excluded from the overall count because it will not be traveled by any UAV. □

Theorem 2.3.3. *TCA computational complexity is* $\mathcal{O}(m \log m)$.

Proof. The Delaunay graph $G = (V, E, W)$ can be computed in $\mathcal{O}(m \log m)$. Kruskal's algorithm for the EMST runs in $\mathcal{O}(|E| \log |E|)$. Since the Delaunay graph is a planar graph, the upper bound on the number edges is $|E| \leq 3m - 6$, as given by Euler's formula. It follows that Kruskal's algorithm runs in $\mathcal{O}(m \log m)$. The BFS on the EMST runs in linear time $\mathcal{O}(m)$. The shortening procedure requires $\mathcal{O}(m)$ iterations. Hence the overall computational complexity is $\mathcal{O}(m \log m)$. □

2.3.5 Distributed Heuristics

We now consider a scenario in which the long-range control channel is not available and communication is possible only among neighboring UAVs by means of a short-range primary channel. Also, the base station can communicate only with neighboring UAVs. In this scenario, we propose two distributed algorithms that allow building a connected multi-hop offloading formation leveraging only partial information about the network.

Two types of control packets are exchanged to build a local network view. For neighborhood discovery, each UAV periodically sends an *hello packet*, containing, among other things, its identifier, time of creation, current position, and if it is connected with the base station or not. The base station instead notifies its presence by periodically sending heartbeat packets. When a UAV receives a heartbeat packet, it rebroadcasts it to all the neighbors to notify its connection with the base station. We assume that the UAVs are lazy-synchronized and at every Δ interval they stop their monitoring activity and gather towards the base station to route sensing data by creating a connected offloading formation.

Distributed Gathering Algorithm

Our Distributed Gathering Algorithm (DGA) is based on predefined rendezvous points and *leader-follower* strategy. We denote with $\mathcal{R}$ the set of predefined rendezvous points where the UAV traffic

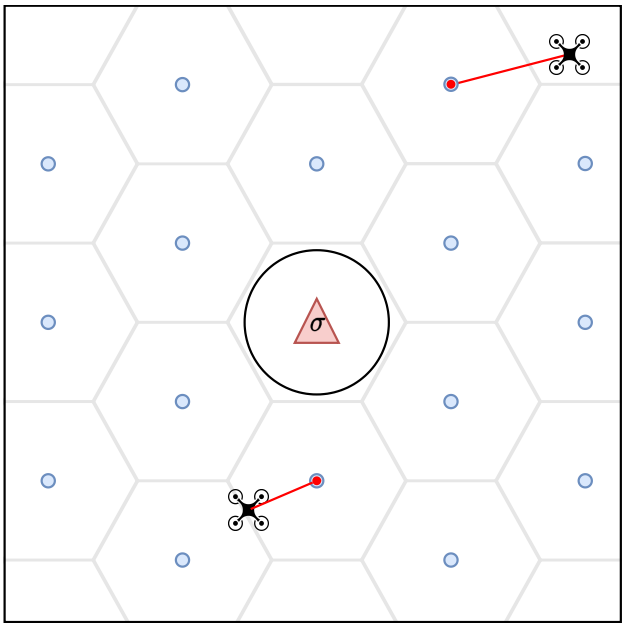


Figure 2.16: *ρ-grid* rendezvous placement

is conveyed to facilitate the creation of the connected formation. When a UAV u has to build a connected formation, it moves towards the rendezvous point that is closest to it and geographically closer to the base station.

When u has reached the rendezvous point, three cases may happen. First, u is a neighbor of the base station or of a connected UAV. In this case, u has reached its point for data offloading; it remains there and broadcasts any hearth-beat packet it receives to notify its connection with the base station. Second, u is a neighbor of a non-connected UAV. In this case, u becomes a follower of this leader neighbor, meaning that u will follow the leader neighbor in case this starts moving. Third, u is isolated. Then u moves towards the next rendezvous point, and checks again which of the three previous cases is verified. Algorithm 6 presents the DGA algorithm.

We consider a rendezvous placement called *ρ-grid* that is derived from a hexagonal tiling, in which all hexagons have uniform radius ρ (see Figure 2.16).

Algorithm 6 Distributed Gathering Algorithm (DGA)

Input: A UAV u running the algorithm
Output: Action for the UAV u

$\Gamma(u) \leftarrow \{u' \in \mathcal{U} \ : \ \mathbf{com}(u, u') \ \}$

if $|\Gamma(u)| > 0$ **then**
  if $\exists u' \in \Gamma(u) \mid \mathbf{com}(u', \sigma)$ **then**
    stop
  else
    leader $\leftarrow \arg\min_{u' \in \Gamma(u)} \|p(u') - p(\sigma)\|$
    follow leader
else
  if $\mathbf{com}(u, \sigma)$ **then**
    stop
  else
    $\Pi \leftarrow \{r \in \mathcal{R} \mid \|p(r) - p(\sigma)\| < \|p(u) - p(\sigma)\|$
    reach $\arg\min_{r \in \Pi} \|p(r) - p(u)\|$

2.3.6 Performance Evaluation

We now evaluate our proposed algorithms through simulations and compare them with state-of-the-art approaches. We use a custom-designed discrete-time simulator written in Python

for experimenting with routing protocols and path planning for UAV networks. We run our tests on an i9-10920X 12 core (24 thread) up to 4,60 GHz and 256 GB RAM.

Scenarios. For all tests, we consider an area of interest of 1500 m x 1500 m, with the UAVs moving according to a random way-point mobility model, at a constant speed of 8 m/s under constant energy consumption factors β_0, β_1. The base station is located in the middle of the area, and each simulation starts with UAVs placed in random positions. In our experiments we let the number of UAVs vary from 5 to 25. We consider two scenarios.

In the first scenario, we evaluate centralized heuristics. UAVs are equipped with both primary and control channels, having communication ranges 140 m and 6 km respectively. Connected offloading formations are computed according to the optimal formulation OPT (see Section 2.3.3), GAA and TCA (see Section 2.3.4) and the Asynchronous Multi-robot Exploration Algorithm [11]. The latter is a topology formation algorithm for terrestrial sensor networks. It exploits Steiner trees to link a set of candidate locations to the base station using robots as relays. The new positions are computed iteratively depending upon the number of nodes to connect. For short we refer to it as AME. For the optimal solution, we used the Gurobi solver [85]. In the second scenario, UAVs are equipped only with the primary channel.

Here we compare DGA (see Section 2.3.5) with Connectivity-Preserved Virtual Force (CPFV) and its floor-based scheme (FloorCPVF) [179]. The latter are two state-of-the-art algorithms for decentralized search for connected offloading formations. They consider mobile sensors moving to re-establish global connectivity with the base station with the goal of maximizing sensing range and residual energy, with total communication as a constraint. Disconnected mobile sensors move toward the base station according to a lazy movement strategy, which consists in getting closer to it and halting for a small amount of time recurrently with the hope of meeting other sensors; stopping when it finds a connected sensor. The virtual forces method is then applied to maximize area coverage. The FloorCPVF variant imposes fixed corridors for the sensors to follow, which are placed in such a way as to limit sensors' overlap. **Metrics.** We measure the performance of the algorithms in terms of:

Sensing Time (seconds) defined as the average time spent by UAVs in sensing during the whole mission. It includes the time to reach targets plus the time to inspect them.

Computational Time (seconds) describes the time needed to compute the connected topology, it is computed only for the centralized solutions.

Building Time (seconds) is the time required for the UAVs to build a connected offloading formation. In centralized approaches, it is measured as the amount of time passing between when all the UAVs received their offloading coordinates, and when they are all connected. In distributed approaches, the building time is measured as the time passing between when the UAVs stop their sensing activity and when they are all connected.

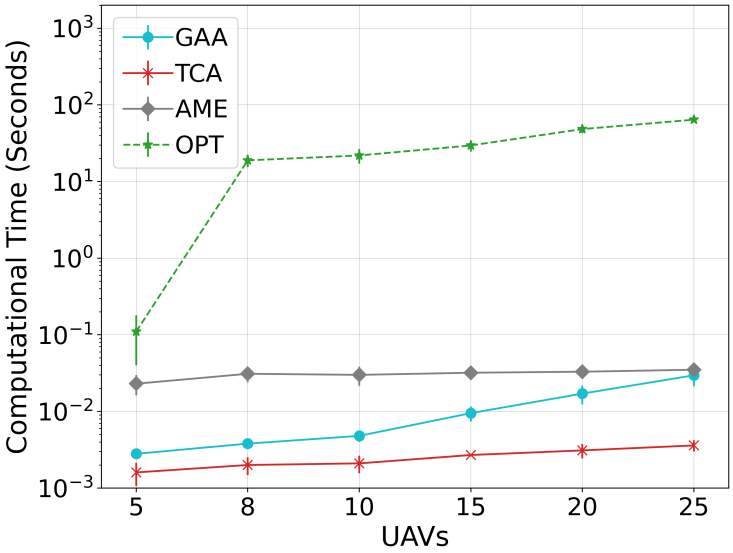


Figure 2.17: Computational Time (seconds) for centralized approaches

Results for Centralized Algorithms

We recall that in centralized approaches the base station has a global view of the network. When it triggers data offloading, it calculates the connected offloading formation and sends the coordinates to the UAVs, which in the meantime have to hover on their position. The time required to calculate the offloading formation is thus a critical aspect as it determines the waiting time for the UAVs. Once received the coordinates, they can move toward the indicated point. When data offloading is completed, each UAV can return to perform sensing.

Figure 2.17 shows the computational time required by centralized approaches to calculating the offloading formation. Results demonstrate that the OPT solution requires long computational time even for small squads of UAVs (from 18.6s in the case of 8 UAVs up to 64.2s for 25 UAVs on average), making its *online* applicability very limited. Our GAA and TCA algorithms instead present very low computational time. TCA for instance is able to calculate an offloading formation in 0.0036s in the case of 25 UAVs, with a 10-fold improvement with respect to the AME algorithm.

The significant computational time reduction does not cause any performance deterioration. Figure 2.20 shows that both GAA and TCA are faster than AME in building the offloading formation independently of the number of UAVs: with TCA the UAVs build a formation in only 85s in the case of 5 UAVs, while AME takes over than 120s for the same number of UAVs. The building time decreases significantly when the number of UAVs increases: TCA employs less than 50s when the squad is composed of 25 UAVs, decreasing the building time of 38% with respect to AME. The OPT is faster than any other solution but it pays in computational time as seen before (Fig. 2.17).

Being faster in offloading operations (with shorter computational and building time), both GAA and TCA can devote more time to sensing with respect to AME. Figure 2.21 shows that GAA and TCA dedicate 14% and 10% more time, respectively, to sensing than AME, while OPT's performance drops due to the long computational time.

Thus, our first set of results clearly shows the benefits of our TCA algorithm which is faster in calculating and building a connected formation and hence it can spend more time in performing sensing.

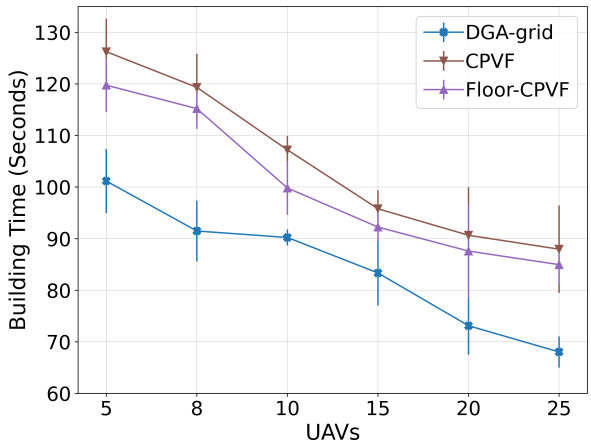


Figure 2.18: Topology building time (seconds) for distributed approaches

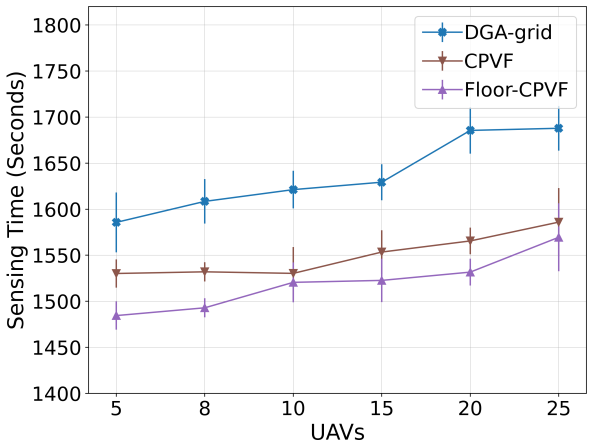


Figure 2.19: Sensing time (seconds) for distributed approaches

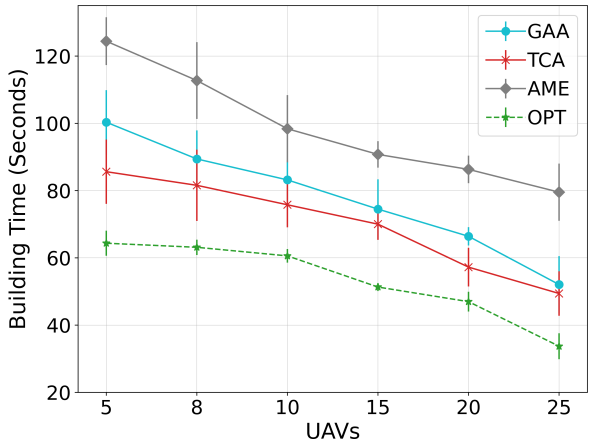


Figure 2.20: Topology building time (seconds) for centralized approaches

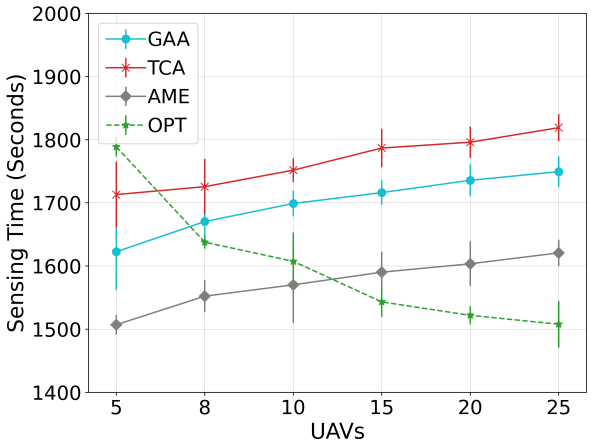


Figure 2.21: Sensing time (seconds) for centralized approaches

Results for Distributed Algorithms

We now evaluate our distributed DGA-grid algorithm and compare it with the two state-of-the-art solutions, CPVF and Floor-CPVF [179]. Figure 2.19 shows the formation building time by varying the number of UAVs. CPVF and Floor-CPVF take more than $120s$ to build a formation of 5 UAVs, while this time decreases to $85-90s$ when the number of UAVs increases to 25. DGA-grid is always faster than CPVF and Floor-CPVF: it builds a connected formation of 5 UAVs in $100s$ and this time decreases to $60s$ when there are 25 UAVs in the squad, achieving a maximum improvement of 25% in case of 8 UAVs.

The time gained while building the connected formation can be used to perform sensing. Figure 2.18 shows that DGA-grid performs a better utilization of mission time as it increases the sensing time of 7% with respect to CPVF.

2.3.7 Conclusions

We presented `Stop & Route`, a novel approach to handling a squad of UAVs under recurrent connectivity constraints. We assume two types of network UAV equipment: long-range and short-range communication hardware. In the first scenario, UAVs are equipped with both hardware compo-

nents. In this case, we propose two different centralized approaches, GAA and TCA, which resort to a central unit to compute the connected offloading formation based on the most recent UAV positions. The other scenario, instead, involves only short-range hardware components. In this scenario, we provide a decentralized solution, called DGA, to build the connected offloading formation. Results show that our proposed algorithms, GAA and TCA, outperform the state-of-the-art centralized algorithm (AME), reaching up to 29% and a 33% improvement in building time. We also demonstrate that both our algorithms achieve an improvement in sensing time over the state-of-the-art of approximately 14% for TCA and 10% for GAA, reducing computational time by up to one order of magnitude. Regarding decentralized approaches, we tested the DGA-grid rendezvous placement. Results show that DGA-grid improves the performance of the state-of-the-art by about 15% in building time and 7% in sensing time.

Chapter 3

Explainable AI for graphs

3.1 Introduction

Graphs are also a natural way to model many real-world systems, from financial transactions and fraud rings to molecular interactions, social networks, recommender systems, up to industrial knowledge graphs. In these settings, Graph Neural Networks (GNNs) have become state-of-the-art for node and graph-level prediction and link discovery. Yet their growing impact also raises a central requirement: explainability. End users, developers, and regulators need to understand why a model produces a particular decision, especially in high-stakes domains governed by regulations (for example, GDPR and the EU AI Act) that emphasize clear, actionable explanations.

This chapter addresses explainability for GNNs from two complementary angles: (i) rigorous, optimization-driven methods to generate faithful counterfactuals on graphs; and (ii) human-centered techniques to communicate those counterfactuals in plain language. The first part introduces UCExplainer, a unified framework that searches for counterfactual explanations by jointly perturbing graph structure (edges), node features, and edge attributes. We formalize the resulting Targeted Counterfactual Explanation Problem for GNNs and show it is NP-hard. To make it tractable in practice, we propose an objective that balances validity (does the prediction flip?), fidelity (is the change faithful to the original sample?), and sparsity (are modifications minimal), together with scheduling policies that adapt these trade-offs over training. A key practical contribution is an efficient edge sparsification strategy based on edge-weight vectors rather than dense adjacency masks, enabling scalability without sacrificing quality. Extensive experiments across datasets and tasks show consistent gains on standard metrics and competitive runtimes against strong baselines.

The second part, Graphs Explainability and Large Language Models, tackles usability: even high-quality counterfactuals remain technical artifacts (masks, discrete features, structural edits) that are difficult to read and understand. We therefore leverage Large Language Models to translate graph counterfactuals into coherent natural-language explanations that remain grounded in the actual edits. We introduce evaluation metrics that directly test an LLM's graph understanding and complement them with a human study. Results indicate that feature-oriented counterfactuals are particularly amenable to clear, actionable narratives for non-experts. This chapter draws on and consolidates material from [73, 75]

3.2 Explaining Graph Neural Network via Counterfactual Samples

Recent breakthroughs in deep learning have propelled significant advancements in artificial intelligence (AI) systems across a wide array of scientific and non-scientific fields. From engineering to social sciences, many areas of human knowledge have greatly benefited from these innovations. However, despite the excitement surrounding the potential of deep learning, growing concerns about the explainability and interpretability of these complex models persist among both the public and researchers. Moreover, explainability is not only crucial for end-users but also for regulators and policymakers. For instance, the European Union has been actively working on regulations such as the Artificial Intelligence Act (AI Act) [155], which includes provisions for the "*right to explanation*". In response to these needs, considerable efforts have been made to establish the foundations of Explainable Artificial Intelligence [8, 58] (XAI). Among the various XAI techniques proposed, *CounterFactual Explanations* (CFEs) have emerged as one of the most promising methods for explaining model predictions [176]. The primary goal of a CFE is to elucidate a model's prediction for a given instance by identifying minimal changes to the input features that would change the model's output. Therefore, CFEs are designed to answer "what if" questions, helping users comprehend the inner logic of a complex model in the form: "*If A had been different, B would* ***not*** *have occurred*". This capability is particularly valuable in sensitive domains such as finance, healthcare, and justice, where understanding the reasoning behind a model's prediction is essential for building trust and ensuring accountability. Existing CFE methods have played a crucial role in interpreting and validating predictions from various machine learning models [42, 130, 163, 187]. Recently, however, there has been a growing need to extend these techniques to accommodate diverse data types and model architectures. Among these, Graph Neural Networks (GNNs) have emerged as particularly effective for tasks involving graph-structured data, such as node classification, graph classification, and link prediction. GNNs have delivered significant benefits across multiple sectors; for instance, in the financial industry, they underpin (semi-)automated fraud detection systems [70]. Furthermore, advancements in GNNs have expanded their applicability to fields such as chemistry and biology. In Wong et al. [207], for example, GNNs were employed to predict the chemical properties of molecules, thereby bypassing the need for costly and time-intensive experimental screenings of large chemical libraries. Due to the unique characteristics of GNNs, developing methods to explain their predictions is, therefore, critical. In this paper we propose a unified framework, UCExplainer, to find the optimal counterfactual explanations for GNN models. Unlike prior methods, UCExplainer balances modifications to edges, edge attributes, and node features by jointly optimizing these perturbations (see Figure 3.1). Through extensive experiments, we show that this approach improves standard quality metrics for explanations across many different datasets.

3.2.1 Related Work

Counterfactual explanations have become a prominent tool for interpreting predictions of black-box models, including GNNs. Unlike factual explainers, which identify the graph's most influential components for the current model prediction, counterfactual methods ask: *what minimal change would alter the model's decision?*

To answer this question, these methods typically employ search or optimization algorithms to identify minimal and realistic graph edits that flip the model's prediction to a different class. For

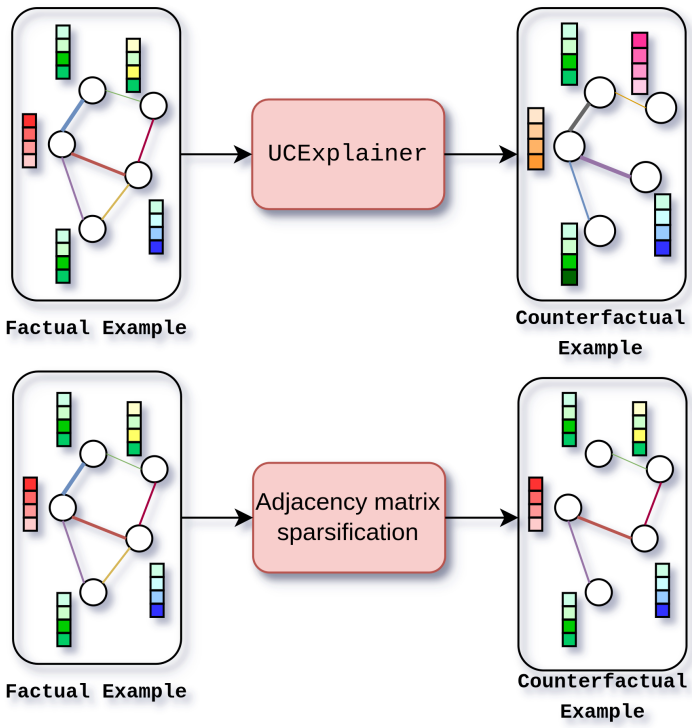


Figure 3.1: Comparison of two approaches for generating counterfactual explanations in Graph Neural Networks (GNNs): Adjacency matrix sparsification vs. UCExplainer. The adjacency matrix sparsification (*bottom*) modifies the graph structure by perturbing the adjacency matrix through edge removal, leaving unaltered both node features and edge attributes (CF-GNNExplainer). UCExplainer (*top*) takes a unified approach, balancing node features, edge attributes, and structural perturbations to find optimal counterfactual explanations.

instance, CF-GNNExplainer [128], formulates explanation as a graph structure optimization problem, learning a discrete edge mask that, when applied, leads to a label change sparsifying the adjacency matrix. However, it only removes edges and does not modify node features or add structure. RCE-GNN [10] introduces robustness by approximating the GNN's decision boundary via linear classifiers in latent space and optimizing perturbations that generalize across similar inputs. GNN-MOExp [124] adopts a multi-objective evolutionary strategy to jointly optimize counterfactuals for fidelity and sparsity, even though it limits explanations to acyclic subgraphs of fixed size. CFF [180] blends factual and counterfactual perspectives through causal metrics, namely, Probability of Necessity and Sufficiency, yielding explanations that are both sufficient for the original prediction and necessary to flip it. More recent work expands the scope of counterfactual explainability. UNR-Explainer [108] extends the paradigm to unsupervised node embeddings, identifying perturbations that alter a node's neighborhood in embedding space.

Generative approaches such as CLEAR [133] employ variational autoencoders to synthesize diverse, valid graph-level counterfactuals, although these often involve larger perturbations. D4Explainer [36] instead uses discrete diffusion models to generate structured perturbations in the adjacency matrix, producing realistic counterfactual explanations.

Other methods tackle graph-level tasks or aim at broader generalization. MACCS [205] targets molecular graphs, producing chemically valid counterfactual molecules via SELFIES mutations. GCFExplainer [100] addresses global model understanding by identifying a set of prototypical counterfactual graphs that together provide recourse for a wide range of instances. Meanwhile, NSEG [27] offers a principled causal explanation by maximizing the probability of necessity and sufficiency through differentiable edge masks. InduCE [192] is an inductive and model-agnostic framework that uses RL to efficiently find minimal graph changes for generating GNN counterfactuals. Its key characteristics include the ability to add edges.

Although originally designed as a factual explainers, both GNNExplainer [218] and PGExplainer [132] can be adapted for counterfactual analysis by interpreting the removal of important edges or node features as hypothetical interventions that flip the prediction. However, this adaptation does not guarantee that such removals always result in a label change, making it unsuitable for generating valid counterfactuals in general.

Our work differs from existing methods in three key aspects. First, it unify edge, edge attributes, and node feature modifications within a single optimization framework, providing more expressive counterfactuals. Second, it supports flexible perturbation constraints on values, allowing fine-grained control over explanation plausibility. Third, it allows nodes or edges to be excluded from the perturbation process. Finally, we demonstrate the method's applicability across node-level and graph-level tasks, setting a new standard for generality in graph counterfactual explanation.

3.2.2 Background

Graph Neural Networks. GNNs extend deep learning to graph-structured data. Formally, let $G = (V, E)$ be a graph with n nodes, and m edges, we can decompose the structure of such graph into smaller constituents, in particular we use the most general case in which the graph has both node features and edge attributes. The structure of G is encoded by its edge index matrix $\mathbf{E} \in \mathbb{N}^{2\times m}$, node features matrix $\mathbf{X} \in \mathbb{R}^{n\times j}$, and edge attributes matrix $\mathbf{EA} \in \mathbb{R}^{m\times z}$, where j and z are the dimensions of node features and edge attributes respectively. Generally speaking,

a GNN g generates node embeddings through a process known as message passing, updating this representation based on the node features and edge attributes of its neighbors. Formally, let $\mathbf{h}_u^l$ denote the embedding of node $u \in V$ at the l-th layer of g and $\mathcal{N}_u$ the set of u's neighbors. The updated node embedding for u at layer $l+1$ is then computed as follows.

$$\begin{aligned}\mathbf{h}_u^{(l+1)} &= g(\mathbf{E}_u^{(l+1)}, \mathbf{X}_u^{(l+1)}, \mathbf{EA}_u^{(l+1)}) = \\ &= \sigma\left(\mathrm{AGG}\left(\left\{\mathbf{h}_v^{(l)} : v \in \mathcal{N}(u)\right\}\right), \mathbf{h}_u^{(l)}\right)\end{aligned}$$

where: $\mathbf{E}_v^{(l+1)}$, $\mathbf{X}_v^{(l+1)}$, and $\mathbf{EA}_u^{(l+1)}$ are the edge index, the node features, and edge attributes matrices of the subgraph G_u^l of G, induced by u and its l-hop neighbors; $\mathrm{AGG}(\cdot)$ is a function combining neighbor embeddings (e.g., sum, mean, or attention); $\sigma(\cdot)$ is an activation function like ReLU.

Counterfactual Explanations. The general formulation for the counterfactual explanation problem in a classification task can be defined as follows:

Definition 3.2.1 (The Counterfactual Explanation Problem (CEP))**.** *Given an input sample $\boldsymbol{x}$ and a classifier f – hereinafter referred to as oracle – the problem to find a counterfactual sample $\boldsymbol{x'}$ such that $f(\boldsymbol{x}) \neq f(\boldsymbol{x'})$ while minimizing the distance $d(x, x')$ is formally defined as:*

$$\begin{aligned}\boldsymbol{x'} &= \arg\min_{\widetilde{\boldsymbol{x}}} \; d(\boldsymbol{x}, \widetilde{\boldsymbol{x}}) \\ &\textit{s.t.: } f(\boldsymbol{x}) \neq f(\widetilde{\boldsymbol{x}}).\end{aligned}$$

The function d enforces similarity between the counterfactual sample $\boldsymbol{x'}$ and the original factual sample $\boldsymbol{x}$. Note that Definition 3.2.1 is general enough and also encompasses a *targeted* version of the counterfactual explanation problem, where the objective is not only to ensure $f(\boldsymbol{x}) \neq f(\widetilde{\boldsymbol{x}})$ but also to enforce a specific target prediction, i.e., $f(\widetilde{\boldsymbol{x}}) = y_t$.

3.2.3 Proposed Method

Problem Formulation

Tackling the CEP outlined in Definition 3.2.1 within the context of GNNs introduces unique challenges. To formalize this, we follow the notation used by Romero et al. [157], replacing the original input instance $\boldsymbol{x}$ with the graph $G(V, E)$ and the counterfactual sample $\boldsymbol{x'}$ as $G'(V', E')$. Since G and G' can be represented by their constituents as shown in Section 3.2.2, the *targeted* counterfactual explanation problem for GNNs can be defined as follows.

Definition 3.2.2 (The Targeted Counterfactual Explanation Problem (TCEP) For GNN)**.** *Given an input graph $G(E, X, EA)$, a GNN g parametrized by θ, a classifier f, and a distance metric d, the targeted counterfactual explanation problem for GNN can be expressed as:*

$$\begin{aligned}\mathbf{E}', \mathbf{X}', \mathbf{EA}' = \arg\min_{\widetilde{\mathbf{E}}, \widetilde{\mathbf{X}}, \widetilde{\mathbf{EA}}} &\left\{ d(\mathbf{E}, \widetilde{\mathbf{E}}) + d(\mathbf{X}, \widetilde{\mathbf{X}}) + d(\mathbf{EA}, \widetilde{\mathbf{EA}}) \right\} \\ \textit{s.t.: } f(g_\theta(\mathbf{E}, \mathbf{X}, \mathbf{EA})) &\neq f(g_\theta(\widetilde{\mathbf{E}}, \widetilde{\mathbf{X}}, \widetilde{\mathbf{EA}})) \quad \wedge \\ f(g_\theta(\widetilde{\mathbf{E}}, \widetilde{\mathbf{X}}, \widetilde{\mathbf{EA}})) &= y_t\end{aligned}$$

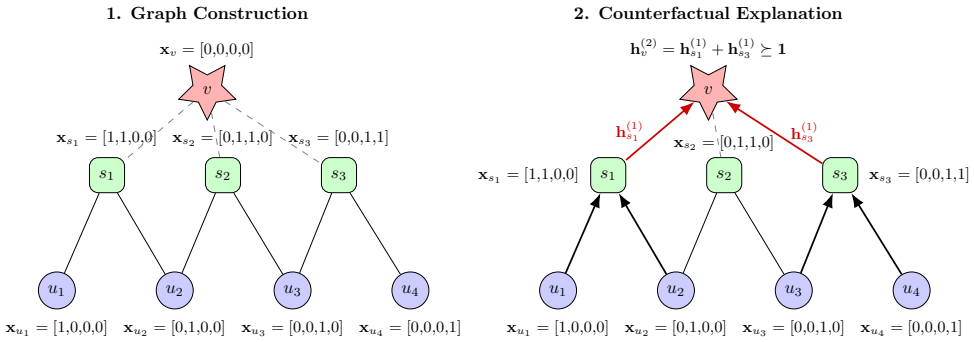


Figure 3.2: Visualization of Theorem 3.2.1: on the left side is depicted the graph construction, on the right side is depicted the edge perturbation (as edge addition)

where $d(\mathbf{E}, \widetilde{\mathbf{E}})$ *captures the structural distance between the original graph* G *and its counterfactual* G', $d(\mathbf{X}, \widetilde{\mathbf{X}})$ *and* $d(\mathbf{EA}, \widetilde{\mathbf{E}}\mathbf{Attr})$ *measure respectively the distance between the original node features and edge attributes and the new ones needed to build the counterfactual graph* G', *and* y_t *denotes the desired target prediction for the counterfactual example.*

Note that we differentiated the classifier f and the GNN g to decouple the representation from the task at hand. For instance, if f operates on a node embedding, the task corresponds to node classification. If f processes the graph embedding, it performs a graph classification task.

Since it has been demonstrated from Verma et al. [192] that the counterfactual reasoning for GNN is NP-Hard using only edges modification, we can say that The Targeted Counterfactual Explanation Problem in Graph Neural Networks (Definition 3.2.2) is NP-Hard as well because it is a more general version, in fact, it includes also node features and edge attributes.

Theorem 3.2.1 (The TCEP NP-Hardness)**.** *The Targeted Counterfactual Explanation Problem (Definition 3.2.2) is NP-Hard*

Proof. We extend the proof given by Verma et al. [192] for the node classification case to account also for node features and egde attributes. First of all, we construct a graph $G = (V, E)$ with the set of nodes $V = \{v\} \cup \mathcal{S} \cup \mathcal{U}$, where v is the target node, $\mathcal{S} = \{s_1, \ldots, s_m\}$ contains one node per subset, and $\mathcal{U} = \{u_1, \ldots, u_n\}$ contains one node per universe element. The edge set E includes all edges (s_i, u_j) for $u_j \in S_i$. The edges (v, s_i) are not present in E initially but are allowed as perturbations. Let the node feature dimension be $d = n$. For each $u_j \in \mathcal{U}$, define $\mathbf{x}_{u_j} = \mathbf{e}_j$, the j-th standard basis vector in $\mathbb{R}^n$. For each $s_i \in \mathcal{S}$, set $\mathbf{x}_{s_i} = \sum_{u_j \in S_i} \mathbf{x}_{u_j}$. The target node v has initial feature $\mathbf{x}_v = \mathbf{0}$. Similarly, each edge (s_i, u_j) has attribute $\mathbf{a}_{s_i u_j} = \mathbf{e}_j$. The potential edges (v, s_i) initially have attribute vectors $\mathbf{a}_{v s_i} = \mathbf{0}$ but can be perturbed. Let g_θ be a two-layer edge-aware GNN with ReLU activations, aggregating messages as follows:

$$\mathbf{h}_{s_i}^{(1)} = \text{ReLU}\left(\sum_{u_j \in \mathcal{N}(s_i)} (\mathbf{x}_{u_j} + \mathbf{a}_{s_i u_j})\right)$$

$$\mathbf{h}_{v}^{(2)} = \text{ReLU}\left(\sum_{s_i \in \mathcal{N}(v)} (\mathbf{h}_{s_i}^{(1)} + \mathbf{a}_{v s_i})\right)$$

Let the final prediction rule be:

$$f(g_\theta(G))[v] = \begin{cases} 1 & \text{if } \mathbf{h}_v^{(2)} \succeq \mathbf{1}, \\ 0 & \text{otherwise.} \end{cases}$$

The GNN predicts 1 if the aggregated vector at v has at least 1 in all coordinates. This occurs if v is connected to a set of s_i nodes that, through their neighbors and associated features and attributes, allow v to reconstruct all standard basis vectors $\mathbf{e}_j$, $j = 1, \ldots, n$.

To satisfy $\mathbf{h}_v^{(2)} \succeq \mathbf{1}$ (the same process can be followed for the other case), the counterfactual reasoner must aggregate enough information to cover all n dimensions. There are two different way to do that. It can select a minimal subset of s_i nodes such that the union of their support covers $\mathcal{U}$ (adding new edges) but this is equivalent to solving a Set Cover problem. Or, it can perturb node features or edge attributes but constructing such perturbations would still need to solve Set Cover to minimize the number of changes because the counterfactual reasoner must still cover all the n dimensions using the fewest modifications. Thus, finding the minimal set of perturbations such that $f(g_\theta(G^*))[v] \neq f(g_\theta(G))[v]$ is equivalent to solving Set Cover, which is NP-hard. See Figure 3.2 to visualize the proof. □

The proof above can be easily extended to the case of graph classification.

To solve the TCEP for GNN we can define a loss function $\mathcal{L}_{total}$ (Equation 3.1) and minimize it using gradient based optimization.

$$\mathcal{L}_{total} = \mathcal{L}_{CF} + \mathcal{L}_E + \mathcal{L}_X + \mathcal{L}_{EA} \tag{3.1}$$

To instantiate Equation 3.1 properly we say that $\mathcal{L}_{CF}$ penalizes when the counterfactual goal is *not* met, i.e., when the modified input instance does *not* lead to an actual classification change; $\mathcal{L}_E$ enforces minimal structural perturbations of the input graph; $\mathcal{L}_X$ controls the magnitude of node features perturbations and finally, $\mathcal{L}_{EA}$ controls the perturbation for the edge attributes.

Since we focus on classification tasks only, the first term ($\mathcal{L}_{CF}$) can be expressed as:

$$\eta = \mathbb{1}[f(g_\theta(E, X, EA)) = f(g(E', X', EA'))]$$
$$\mathcal{L}_{CF} = \eta \cdot \mathrm{CE}\left(f(g(E, X, EA)), f(g(E', X', EA'))\right)$$

where CE is the cross-entropy loss between the original and the counterfactual predictions and η is an indicator variable ensuring this term is considered only when the classification change has not yet occurred. The graph structural loss measuring the differences due to edge sparsification, $\mathcal{L}_E$, is defined as:

$$\mathcal{L}_E = \alpha \cdot \sum_{i \in |E|} \|E_i - E'_i\|_1$$

The node features loss $\mathcal{L}_X$ can be expressed as:

$$\mathcal{L}_{\mathbf{X}} = \beta \cdot \left(\sum_{j \in \mathcal{D}_{\mathbf{X}}} \|\mathbf{X}_{:,j} - \mathbf{X}'_{:,j}\|_1 + \sum_{j \in \mathcal{C}_{\mathbf{X}}} \|\mathbf{X}_{:,j} - \mathbf{X}'_{:,j}\|_2^2 \right)$$

In the same way we can express the edge attributes loss:

$$\mathcal{L}_{\mathbf{EA}} = \lambda \cdot \left(\sum_{j \in \mathcal{D}_{\mathbf{E}}} \|\mathbf{EA}_{:,j} - \mathbf{EA}'_{:,j}\|_1 + \sum_{j \in \mathcal{C}_{\mathbf{E}}} \|\mathbf{EA}_{:,j} - \mathbf{EA}'_{:,j}\|_2^2 \right)$$

where $\mathcal{D}_X$ and $\mathcal{C}_X$ index the discrete and continuous node features, and $\mathcal{D}_E$, $\mathcal{C}_E$ index the discrete and continuous edge attributes, respectively.

Finally, $\alpha, \beta,$ and λ are adjustable trade-off hyperparameters that balance between features and structural modifications. The impact of $\alpha, \beta,$ and λ will be further analyzed in Section 3.2.4.

Optimization With Discrete Values

To optimize the loss function defined in Equation 3.1 using gradient-based methods, we introduce three differentiable perturbation matrices: the node feature perturbation matrix **P** responsible for modifying node features, the edge perturbation matrix **EP** that governs changes in the graph topology by modifying edge values, and finally the edge attributes perturbation matrix **EAP** that is responsible for modifying the edge attrbutes.

Given that the graph structure, node features, and edge attributes often consist of discrete values, we adopt the approach proposed by Srinivas et al. [175] to preserve differentiability. Specifically, we apply a *tanh*-based transformation to discrete node features and edge attributes, and a sigmoid activation to edge perturbations. This ensures that modifications remain within valid bounds while allowing gradients to propagate effectively during optimization. The differentiable versions of these matrices are used to update model parameters via backpropagation, whereas their corresponding thresholded (non-differentiable) versions are ultimately employed to generate the final counterfactual examples.

Efficient Edges Sparsification

Our edge sparsification process leans on the approach introduced by Lucic et al. [128], while addressing scalability challenges inherent to their method. Specifically, Lucic et al. represent edges using a full adjacency matrix $A \in \{0,1\}^{n \times n}$, which significantly limits the applicability of their algorithm for graphs containing more than 30–35 nodes. To overcome this limitation, we leverage a novel strategy that exploits the edge weight vector, that can be seamlessly integrated into various graph convolutional layers, such as *ChebConv* [53], *GCNConv* [113], *GINEConve* [97], and *GraphConv* [141].

Given a generic graph $G(V, E)$ with n nodes and m edges, our approach introduces a perturbation vector $\mathbf{EP} \in \{0,1\}^{1 \times m}$ that effectively cancels out edges by feeding it into the GNN. We formally show that this technique is equivalent to performing edge deletion using the full adjacency matrix. For *GCNConv*, this equivalence is trivial and follows directly from Theorem 3.2.2.

Theorem 3.2.2. *[Equivalence of Edge Weight Nullification and Adjacency Matrix Edge Removal in GCNConv layers].*

Let $G = (V, E)$ be a generic graph with n nodes and m edges. Let $\mathbf{X} \in \mathbb{R}^{n \times j}$ be the node features matrix. Consider a GCNConv layer parameterized by a weight matrix $\mathbf{W} \in \mathbb{R}^{d \times d'}$. Setting an edge weight to zero in the GCNConv's edge weight vector is equivalent to removing the corresponding edge in the adjacency matrix representation.

Proof. The proof trivially follows from Equation 3.2 representing the GCNConv convolution. We can represent the edges of the graph using an edge index matrix $\mathbf{E} \in \mathbb{R}^{2\times m}$ or a full adjacency matrix $A \in \{0,1\}^{n\times n}$. Let $\mathbf{EP} \in \mathbb{R}^{1\times m}$ be the vector of edge weights.

Setting a generic $e_{(i,j)} \in \mathbf{EP}$ to 0 removes the contribution of the corresponding edge (i,j) in the message-passing process (Equation 3.2).

$$\mathbf{h}'_i = \sigma\left(\mathbf{W}^\top \sum_{j\in\mathcal{N}(i)} \frac{e_{j,i}}{\sqrt{\hat{d}_j \hat{d}_i}} \mathbf{x}_j\right) \tag{3.2}$$

Removing, instead, a generic edge (i,j) from the full adjacency matrix (setting $A_{ij} = 0$), ensure that node j will no longer be in the neighborhood of node i. Formally this means that in Equation 3.2 $j \notin \mathcal{N}(i)$, thus, node j will not contribute to the representation of node i.

Since both methods eliminate the contribution of edge (i,j), we obtain that the intermediate representation h'_i will be equal in both cases.

□

The impact of this approach on the execution time and the memory occupation has been studied empirically in Section 3.2.4.

The UCExplainer Algorithm

In this section, we detail UCExplainer, our proposed counterfactual explanation algorithm (Algorithm 7), which minimizes the objective defined in Equation 3.1 via gradient-based optimization to solve the TCEP (Definition 3.2.2). To accommodate datasets where node features or edge attributes are unavailable, these components can be turned off without affecting the process. Furthermore, to make the notation easier, from now on, the GNN g_θ that produces the representation for nodes or graphs will also include the classifier f, and we will omit the parametrization θ.

The algorithm takes as input a graph $G(E, X, EA)$, a pre-trained GNN model g with fixed parameters, an integer k representing the maximum number of optimization epochs, a target class y_t, two vectors $\mathbf{M}_d$ and $\mathbf{M}_d^{attr}$ serving as a mask for discrete node features and discrete edge attributes respectively and a learning rate γ.

During the initialization phase (lines 1–10) the perturbation matrices for node features, edge index, and edge attributes are initialized as $\mathbf{P}^0$, $\mathbf{EP}^0$ (as in Section 3.2.3), and $\mathbf{EAP}^0$, respectively. The oracle g computes an initial prediction, y, for the input graph $G(E, X, EA)$ and sets the epoch counter to 1.

The algorithm, then, extracts the constituents, namely, edge index, node features, and edge attributes from the original graph for subsequent processing. To establish a baseline for optimization, the best loss is initialized to infinity and the placeholder for the counterfactual sample is set to null.

The optimization process begins in line 11. From line 12 up to line 14, the algorithm perturbs the edge index (Algorithm 9), the nodes features (Algorithm 8), and the edge attributes (Algorithm 11). In line 15 and 16, the algorithm computes the new prediciton for both the differentiable and the non differentiable versions of the pertubed componets getting the differentiable prediction (y_{new}) and the non-differentiable one (y^{nd}_{new}).

In line 17 and 18 the algorithm computes the loss function using Algorithm 10. Finally, from lines 19–26 the perturbation matrices are updated to minimize the loss. If the model's prediction changes with the newly perturbed components while achieving the lowest loss observed so far, the algorithm identifies a valid counterfactual sample. Below, we describe the algorithms used to compute all the perturbations as well as the loss function that drives the optimization process at the core of UCExplainer.

Algorithm 7 UCExplainer

Input: $\mathbf{G}(\mathbf{A}, \mathbf{X}, \mathbf{EA})$: Graph to explain, $\mathbf{g}$: Oracle, $\mathbf{k}$: maximum number of epochs, y_t: target class, $\mathbf{M_d}$ discrete features mask vector, $\mathbf{M_d^{attr}}$ discrete features mask vector, γ: learning rate, $\mathbf{R}_m$ and $\mathbf{R}_M$: vectors containing lower and upper bound for each feature

Output: The counterfactual sample G'

$\mathbf{P}^0 \leftarrow \mathbf{0}^{n \times j}$ $\mathbf{EP}^0 \leftarrow \mathbf{1}^{1 \times m}$ $\mathbf{EAP}^0 \leftarrow \mathbf{0}^{m \times z}$ $y \leftarrow g(G(\mathbf{E}, \mathbf{X}, \mathbf{EA}))$ $epoch \leftarrow 1$ $\mathbf{E}, \mathbf{X}, \mathbf{EA} \leftarrow G(\mathbf{E}, \mathbf{X}, \mathbf{EA})$

$G' \leftarrow \emptyset$ $\mathcal{L}_{best} \leftarrow +\infty$

while $epoch \leq k$ **do**

- $\mathbf{E}_p^{nd}, \mathbf{E}_p \leftarrow \texttt{perturb\_edges}\left(\mathbf{E}, \mathbf{EP}^t\right)$
- $\mathbf{X}_p^{nd}, \mathbf{X}_p, \mathbf{X}_d, \mathbf{X}_c \leftarrow \texttt{pert\_nodes\_feat}\left(\mathbf{R}_m, \mathbf{R}_M, \mathbf{X}, \mathbf{M}_d, \mathbf{P}^t\right)$
- $\mathbf{EA}_p^{nd}, \mathbf{EA}_p, \mathbf{EA}_d, \mathbf{EA}_c \leftarrow \texttt{pert\_edges\_attrs}\left(\mathbf{R}_m, \mathbf{R}_M, \mathbf{EA}, \mathbf{M}_d^{attr}, \mathbf{EAP}^t\right)$
- $y_{new} \leftarrow g\left(G\left(\mathbf{X}_p, \mathbf{E}_p, \mathbf{EA_p}\right)\right)$
- $y_{new}^{nd} \leftarrow g\left(G\left(\mathbf{X}_p^{nd}, \mathbf{E}_p^{nd}, \mathbf{EA_p^{nd}}\right)\right)$
- $params \leftarrow \left\{y_{new}, y_t, \mathbf{EP}^t, \mathbf{X}, \mathbf{X}_p, \mathbf{M}_d, \mathbf{EA}, \mathbf{EA}_p, \mathbf{M}_d^{attr}, epoch\right\}$
- $\mathcal{L}_{total} \leftarrow \texttt{get\_loss}(params)$
- $\mathbf{P}^{t+1} \leftarrow \mathbf{P}^t - \gamma \cdot \nabla_P\left(\mathcal{L}_{total}\right)$
- $\mathbf{EP}^{t+1} \leftarrow \mathbf{EP}^t - \gamma \cdot \nabla_{EP}(\mathcal{L}_{total})$
- $\mathbf{EAP}^{t+1} \leftarrow \mathbf{EAP}^t - \gamma \cdot \nabla_{EAP}(\mathcal{L}_{total})$
- **if** $y_{new} = y_t \wedge \mathcal{L} < \mathcal{L}_{best}$ **then**
  - $G' \leftarrow G\left(\mathbf{E}_p, \mathbf{X}_p, \mathbf{EA}_p\right)$
  - $\mathcal{L}_{best} \leftarrow \mathcal{L}_{total}$
- **end**
- $epoch \leftarrow epoch + 1$

end

return G'

Node Features and Edge Attributes Perturbation. Algorithms 11 and 8 outline the perturbation mechanism applied to node features and edge attributes. The process begins by applying the $\mathbf{P}_{block}$ ($\mathbf{EA}_{block}$) mask to the perturbation matrix to prevent entire nodes (edges) or single features (attributes) from being perturbed (line 1) following the user's directives. Then the algorithm computes the continuous mask and a scaled perturbation matrix $\mathbf{P}_s^{(t)}$ ($\mathbf{EAP}_s^{(t)}$), to perturb the discrete node features (edge attributes), using a tanh-based transformation, ensuring that the values remain within the predefined feature bounds (line 2-3). Next, discrete features are updated by applying the Hadamard product between the scaled perturbation and the discrete feature mask $\mathbf{M}_d$ ($\mathbf{M}_d^{attr}$), followed by summation with the original features $\mathbf{X}$ ($\mathbf{EA}$), and subsequently clamped within the feature range (line 3). Similarly, continuous features are updated without any transformation on the perturbation and are clamped accordingly (line 4). The final perturbed matrix is obtained by summing the contributions of discrete and continuous feature updates, yielding $\mathbf{X}_p = \mathbf{X}_d + \mathbf{X}_c$ ($\mathbf{EA}_p = \mathbf{EA}_d + \mathbf{EA}_c$) (line 5). In line 6 instead, the algorithms compute the discretized version of the matrix transforming to integers discrete values, and, finally in line 7 the functions returns

Algorithm 8 Perturb nodes features (`pert_nodes_feat`)

Input: $\mathbf{R}_m$ and $\mathbf{R}_M$: Vectors containing lower and upper bound for each feature, $\mathbf{X}$: Original nodes features matrix, $\mathbf{M}_d$: discrete features mask, $\mathbf{P}^{(t)}$: Node feature perturbation matrix at the t-th iteration, $\mathbf{P}_{block}$: Mask to prevent node features perturbation

Output: $\mathbf{X}_p^{nd}$: the non-differentiable nodes features matrix, $\mathbf{X}_p$: the differentiable nodes features matrix, $\mathbf{X}_d$: perturbed discrete features, $\mathbf{X}_c$: perturbed continuous features

$\mathbf{P}^t \leftarrow \mathbf{P}^t \cdot \mathbf{P}_{block}$

$\mathbf{M_c} \leftarrow \mathbf{1} - \mathbf{M_d}$

$\mathbf{P}_s^t = tanh\left(\mathbf{P}^t\right) \cdot (\mathbf{R}_m + (\mathbf{R}_M - \mathbf{R}_m))$

$\mathbf{X}_d \leftarrow \texttt{clamp}\left(\mathbf{X} + \left(\mathbf{M}_d \odot \mathbf{P}_s^t\right)\ ;\ \mathbf{R}_m\ ;\ \mathbf{R}_M\right)$

$\mathbf{X}_c \leftarrow \texttt{clamp}\left(\mathbf{X} + \left(\mathbf{M}_c \odot \mathbf{P}^t\right)\ ;\ \mathbf{R}_m\ ;\ \mathbf{R}_M\right)$

$\mathbf{X}_p \leftarrow \mathbf{X}_d + \mathbf{X}_c$

$\mathbf{X}_p^{nd} \leftarrow \texttt{clamp}\left(\mathbf{X}_c + \left(\mathbf{M}_d \odot \texttt{to\_int}\left(\mathbf{P}_s^t\right)\right)\ ;\ \mathbf{R}_m\ ;\ \mathbf{R}_M\right)$

return $\mathbf{X}_p^{nd}, \mathbf{X}_p, \mathbf{X}_d, \mathbf{X}_c$

Algorithm 9 Perturb edges (`perturb_edges`)

Input: $\mathbf{E}$: Original edges matrix, $\mathbf{EP}^t$: Edge perturbation vector at the t-th iteration, $\mathbf{E}_{block}$: Mask to prevent edge perturbation

Output: $\mathbf{E}_p^{nd}$: the non-differentiable edge matrix, $\mathbf{E}_p$: the differentiable edge matrix

$\mathbf{EP}^t \leftarrow \mathbf{EP}^t \cdot \mathbf{E}_{block}$ $\mathbf{E}_p \leftarrow \mathbf{E} \cdot \sigma\left(\mathbf{EP}^t\right)$ $\mathbf{E}_p^{nd} \leftarrow \mathbb{1}\left[\sigma\left(\mathbf{EP}^t\right) > 0.5\right]$

return $\mathbf{E}_p^{nd}, \mathbf{E}_p$

Algorithm 10 Loss Computation (`get_loss`)

Input: y_{new}^{nd}: the new prediction, y_t: counterfactual target class, $\mathbf{EP}$: The edge perturbation vector, $\mathbf{X}$: The original nodes features matrix, $\mathbf{X}_p$: The perturbed nodes features matrix, $\mathbf{M}_d$: discrete features mask, $\mathbf{EA}$: The original edges attributes matrix, $\mathbf{EA}_p$: The perturbed edges attributes matrix, $\mathbf{M}_d^{attr}$: discrete edges attributes features mask, *epoch*: the current epoch

Output: The loss value $\mathcal{L}_{total}$

$\eta \leftarrow \mathbb{1}\left[argmax\left(y_{new}\right) = y_t\right]$ $\mathcal{L}_{CE} \leftarrow \mathrm{CE}\left(y_{new}\ ;\ y_t\right)$

$\mathcal{L}_E \leftarrow \sum |\sigma\left(\mathbf{EP}_x\right) - \mathbf{1}|$

$\mathcal{L}_d^X \leftarrow \mathrm{L1}(\mathbf{X} \odot \mathbf{M}_d\ ;\ \mathbf{X}_p \odot \mathbf{M}_d)$

$\mathcal{L}_c^X \leftarrow \mathrm{MSE}\left(\mathbf{X} \odot (1 - \mathbf{M}_d)\ ;\ \mathbf{X}_p \odot (1 - \mathbf{M}_d)\right)$

$\mathcal{L}_d^{EA} \leftarrow \mathrm{L1}\left(\mathbf{EA} \odot \mathbf{M}_d\ ;\ \mathbf{EA}_p \odot \mathbf{M}_d^{attr}\right)$

$\mathcal{L}_c^{EA} \leftarrow \mathrm{MSE}\left(\mathbf{EA} \odot \left(1 - \mathbf{M}_d^{attr}\right)\ ;\ \mathbf{EA}_p \odot \left(1 - \mathbf{M}_d^{attr}\right)\right)$

$\mathcal{L}_X \leftarrow \mathcal{L}_d^X + \mathcal{L}_c^X$ $\mathcal{L}_{EA} \leftarrow \mathcal{L}_d^{EA} + \mathcal{L}_c^{EA}$

$\alpha, \beta, \lambda \leftarrow \texttt{get\_balance\_factors}(epoch)$ $\mathcal{L}_{total} = \eta\mathcal{L}_{CE} + \alpha\mathcal{L}_E + \beta\mathcal{L}_X + \lambda\mathcal{L}_{EA}$

return $\mathcal{L}_{total}$

the non-differentiable features matrices $\mathbf{X}_p^{nd}$ ($\mathbf{EA}_p^{nd}$), the differentiable features matrices $\mathbf{X}_p$ ($\mathbf{EA}_p$), and both the continuous and discrete feature matrices $\mathbf{X}_d, \mathbf{X}_c$ ($\mathbf{EA}_d, \mathbf{EA}_c$). We employ both the differentiable and the non-differentiable versions of the matrices to maintain gradient flow during optimization while ultimately obtaining the final discrete prediction. Specifically, the differentiable matrices $\mathbf{X}_p$ and $\mathbf{EA}_p$ are used to produce a continuous output from the model, which is essential for computing the loss via backpropagation. In contrast, the non-differentiable matrices, $\mathbf{X}_p^{nd}$ and $\mathbf{EA}_p^{nd}$, are derived by thresholding $\mathbf{EAP}^{(t)}$ and $\mathbf{P}_s^{(t)}$, thereby yielding the definitive prediction. It is important to note that the final counterfactual sample is constructed from these *non-differentiable* components, ensuring that the discrete nature of the graph is preserved in the final output.

Algorithm 11 Edge attributes perturbation function (`pert_edges_attrs`)

Input: $\mathbf{R}_m$ and $\mathbf{R}_M$: Vectors containing lower and upper bound for each feature, $\mathbf{EA}$: Original edges attirbutes matrix, $\mathbf{M}_d^{attr}$: discrete features mask, $\mathbf{EAP}^t$: Edge attributes perturbation matrix at the t-th iteration, $\mathbf{EA}_{block}$: Mask to prevent attributes perturbation

Output: $\mathbf{EA}_p^{nd}$: the non-differentiable edges attributes matrix, $\mathbf{EA}_p$: the differentiable edges attributes matrix, $\mathbf{EA}_d$: perturbed discrete edges attributes matrix, $\mathbf{EA}_c$: perturbed continuous edges attributes matrix

$\mathbf{EAP}^t \leftarrow \mathbf{EAP}^t \odot \mathbf{EA}_{block}$
$\mathbf{M}_c^{attr} \leftarrow \mathbf{1} - \mathbf{M}_d^{attr}$
$\mathbf{EAP}_s^t = tanh\left(\mathbf{EAP}^t\right) \cdot (\mathbf{R}_m + (\mathbf{R}_M - \mathbf{R}_m))$
$\mathbf{EA}_d \leftarrow \texttt{clamp}\left(\mathbf{EA} + \left(\mathbf{M}_d^{attr} \odot \mathbf{EAP}_s^t\right)\ ;\ \mathbf{R}_m\ ;\ \mathbf{R}_M\right)$
$\mathbf{EA}_c \leftarrow \texttt{clamp}\left(\mathbf{EA} + \left(\mathbf{M}_c^{attr} \odot \mathbf{EAP}^t\right)\ ;\ \mathbf{R}_m\ ;\ \mathbf{R}_M\right)$
$\mathbf{EA}_p \leftarrow \mathbf{EA}_d + \mathbf{EA}_c$
$\mathbf{EA}_p^{nd} \leftarrow \texttt{clamp}\left(\mathbf{X}_c + \left(\mathbf{M}_d^{attr} \odot \texttt{to\_int}\left(\mathbf{EAP}_s^t\right)\right)\ ;\ \mathbf{R}_m\ ;\ \mathbf{R}_M\right)$
return $\mathbf{EA}_p^{nd}, \mathbf{EA}_p, \mathbf{EA}_d, \mathbf{EA}_c$

Edge Perturbation. To increase control on the edge sparsification mechanism, the algorithm exploits the $\mathbf{E}_{block}$ matrix to prevent edges from being perturbed, accommodating the user's directives (line 1). To perturb the edges in the factual graph G Algorithm 9 computes the differentiable edge matrix $\mathbf{E}_p$ by applying a sigmoid activation σ to the edge perturbation vector $\mathbf{EP}^{(t)}$, ensuring that the resulting values are in the range $(0, 1)$ (line 2). The non-differentiable edge matrix $\mathbf{E}_p^{nd}$ is obtained by thresholding the sigmoid output at 0.5, enforcing a hard binary decision on edge presence (line 3). Also in this case, we preserve differentiability using a differentiable and non-differentiable version of the matrices.

Loss Function. The loss function is computed in Algorithm 10. In line 1 the indicator variable η determines whether the cross-entropy loss $\mathcal{L}_{CE}$ should be applied (line 2). This ensures that the classification loss is only considered when the counterfactual prediction does not yet match the counterfactual target class y_t. The *edge distance loss* $\mathcal{L}_E$ is then computed in line 3 by summing the absolute differences between the sigmoid-transformed edge perturbation values and 1, encouraging minimal modifications to the graph structure.

In lines 4 and 5 , the *node feature loss* $\mathcal{L}_X$ is computed separately for discrete and continuous features. The discrete feature loss $\mathcal{L}_d$ is defined using the L1 loss between the masked original feature values and their perturbed versions, while the continuous feature loss $\mathcal{L}_c$ is computed using the Mean Squared Error (MSE). The same is applied for the edge attributes in lines 7 and 8. These losses are then combined to form the total feature loss $\mathcal{L}_X = \mathcal{L}_d + \mathcal{L}_c$ and edge attribute loss $\mathcal{L}_{EA} = \mathcal{L}_d^{EA} + \mathcal{L}_c^{EA}$.

Dataset Name	#Graphs	#Nodes	#Edges	#Features	#Edge Attrs.	#Classes
Cora (N) [217]	1	2,708	5,429	1,433	n.a.	7
CiteSeer (N) [217]	1	3,312	4,732	3,703	n.a.	6
PubMed (N) [217]	1	19,717	44,338	500	n.a.	3
AIDS (G) [103, 164]	2000	15.69	16.20	4	n.a.	2
Cuneiform (G) [118]	267	21.27	44.80	3	2	30

Table 3.1: Datasets used to carry out the experiments in Section 3.2.4

Next, in line 7, the function `get_balance_factors` determines the value of the weighting parameter α, β and λ, which regulates the trade-off between edge, node features, and edge attributes modifications. To enhance the flexibility of the framework, we introduce the ability to select from various *scheduling policies*, allowing these values to dynamically evolve over training epochs. This adaptive approach ensures that the model progressively refines its focus on different parts of the original graph, leading to a more effective optimization process and improving the quality of the generated counterfactual explanations. Finally, in line 8, the total loss $\mathcal{L}_{total}$ is computed as specified in Equation 3.1. This formulation ensures that perturbations remain minimal while enforcing the desired classification change.

Computational Complexity Analysis

The time complexity of the UCExplainer algorithm is primarily determined by its iterative optimization process over k training epochs. Let n and m denote the number of nodes and edges in the graph, respectively. Moreover, let j be the number of node features and z be the number of edge attributes. Therefore, each iteration consists of:
(1) **Perturbation Step:** $O(nj + mz + m)$ for updating node features, edge attributes, and edge perturbations;
(2) **GNN Forward Pass:** $O(L(nj + mz))$, where L is the number of GNN layers;
(3) **Loss Computation:** $O(nj + mz + m)$ for feature and edge losses;
(4) **Gradient Update:** $O(nj + mz + m)$ for updating perturbation vectors.

Overall, the worst-case complexity is, therefore, $\mathcal{O}(knj + kmz + km)$. This implies that the algorithm scales *linearly* with the number of nodes, edges, features, and training epochs.

3.2.4 Experiments

Below, we outline our experimental setup and evaluation methodology to assess the effectiveness of our proposed approach. ***Tasks, Datasets, and Models.*** Our method is tested on two tasks – *node classification* and *graph classification* – using a diverse collection of real-world datasets to ensure robustness and generalizability across domains such as citation networks, web page classification, and social network analysis (see Table 3.1).

Specifically, the dataset collection includes the Planetoid datasets (CiteSeer, Cora, and PubMed) ***Baselines.*** We evaluate our approach against a set of baselines that span naive strategies (random-edges, random-features, and ego-graph) and state-of-the-art techniques such as: CF-GNNExplainer [128], CounterFactual and Factual Explainer (CFF) [180], UNR [108], GNNExplainer [218], CLEAR [133], D4Explainer [36], PGExplainer [132], InduCE [192], RCExplainer [200]. Since the code repositories are not always available or easily integrable for some of the results we took the values directly from the original papers or added just the metrics in the original repositories. Baselines such as random-edges and random-features perturb the graph's components by applying random modifications, whereas the ego-graph method extracts a subgraph centered on a given node.

Settings. The experiments were conducted on two machines, each equipped with an Nvidia GTX 4090 GPU, 64 GB of RAM, and an AMD Ryzen 9 7900 processor. To ensure robustness, we employed 4-fold cross-validation. Additionally, we performed an extensive hyperparameter tuning process to identify the optimal parameter configurations for our method, which were set as follows: learning rate of 0.1, and 500 training epochs. For the baseline models, we used the same hyperparameters specified in the original papers.

Metrics

To assess the quality of generated counterfactuals, we consider several key metrics.
Validity (Ω): also known as correctness, as defined by Guidotti et al. [79, 80], indicates whether the explainer is capable of producing a valid counterfactual explanation. Formally, given the original instance x, the candidate counterfactual sample x', and oracle g validity is defined as:

$$\Omega(x, x') = \mathbb{1}[g(x) \neq g(x')]$$

Fidelity (Ψ): as defined in Romero et al. [157] measures how faithful the explanations are to the oracle considering their correctness. Given the input x, its true label y_x, and its counterfactual x', fidelity is defined as:

$$\Psi(x, x') = \chi(x) - \mathbb{1}\left[g\left(x'\right) = y_x\right]$$
$$\chi(x) = \mathbb{1}\left[g(x) = y_x\right]$$

where $\chi(x)$ is an indicator function that describes the oracle's accuracy. Notice that $\Psi(x, x')$ can assume three values. A value of 1 implies that both the explainer and oracle are working correctly. It is straightforward to confirm that $\chi(x)$ must output 1 (i.e., $g(x) = y_x$) while the indicator should yield 0 (i.e., $g(x') \neq y_x$). Values of 0 or -1 signal an issue with either the explainer or the oracle. However, it is not possible to determine whether the fault lies with the explainer or the oracle. This reflects a limitation of fidelity, as it evaluates correctness based on the true label y_x rather than the prediction $g(x)$.
Node Sparsity ($\Delta_{\mathbf{X}}$): this metric measures the sparsity of node features in a counterfactual graph compared to a factual graph. It quantifies how many node attributes have been modified between the factual and counterfactual node attribute matrices. The number of modified attributes is computed as follows:

$$\Delta_{\mathbf{X}} = \frac{\sum(\mathbf{X} \neq \mathbf{X}_p)}{|\mathbf{X}|}$$

where $\mathbf{X}$ and $\mathbf{X}_p$ are the node features matrices of the factual and counterfactual graphs, respectively.
Edge Sparsity ($\Delta_{\mathbf{E}}$): It quantifies how many edges have been modified between the factual and counterfactual graph. The number of modified edges is computed by comparing the edge index matrices of the factual and counterfactual graphs, namely:

$$\Delta_{\mathbf{E}} = \frac{\sum(\mathbf{E} \neq \mathbf{E}_p)}{|\mathbf{E}|}$$

where $\mathbf{E}$ and $\mathbf{E}_p$ are the factual and counterfactual edge index matrices.
Edge Attributes Sparsity ($\Delta_{\mathbf{EA}}$): similarly to the node sparsity, the edge attributes sparsity

measures the differences in the edge attributes matrices between the counterfactual graph compared to a factual graph:

$$\Delta_{\mathbf{EA}} = \frac{\sum(\mathbf{EA} \neq \mathbf{EA}_p)}{|\mathbf{EA}|}$$

where $\mathbf{EA}$ and $\mathbf{EA}_p$ are the factual and counterfactual edge attributes matrices.

Distribution Distance (d_v): To better understand how counterfactual samples are distributed within the original data, we defined a cluster-like distance measure. Given a graph $G = (V, E)$ we compute the distribution distance as follows:

$$d_v = \|\mathbf{G}_h - \mu_G\|_2, \quad \text{where,}$$
$$\mathbf{G}_h = mean(g(G)), \quad \mu_G = \frac{1}{|D|}\sum_{G \in D} mean(g(G))$$

G_h is the embedded graph representation for G, and μ_G is the distribution mean value computed over all the samples in the dataset D.

Explainer	Validity ↑	Fidelity ↑	Distr. Distance ↓	Node Spars. ↓	Edge Spars. ↓
	Dataset: CiteSeer - Model: GCNConv - Task: Node Classification				
UCExplainer$_{sin}$	**1.000**(±**0.000**)	**0.755**(±**0.007**)	5.620(±0.212)	**0.031**(±**0.002**)	**0.025**(±**0.01**)
EGO	0.012(±0.003)	0.250(±0.500)	1.995(±0.206)	*n.a.*	0.937(±0.003)
Random Edges	0.123(±0.009)	0.476(±0.099)	**1.716**(±**0.089**)	*n.a.*	0.393(±0.014)
Random Features	0.514(±0.065)	0.721(±0.024)	32.926(±0.291)	0.489(±0.000)	*n.a.*
CFF	0.010(±0.004)	0.125(±0.629)	2.325(±1.323)	*n.a.*	0.594(±0.194)
CF-GNNExplainer	0.108(±0.014)	0.534(±0.103)	1.783(±0.134)	*n.a.*	0.070(±0.022)
UNR	0.047(±0.012)	0.202(±0.162)	2.389(±0.336)	*n.a.*	0.186(±0.068)
GNNExplainer	0.000(±0.000)	*n.a.*	*n.a.*	*n.a.*	*n.a.*
induCE [1]	0.785(±0.010)	0.523(±0.085)	3.224(±0.250)	*n.a.*	0.155(±0.043)
PGExplainer [2]	0.245(±0.021)	0.010(±0.004)	5.453(±0.583)	*n.a.*	0.653(±0.023)
	Dataset: AIDS - Model: GCNConv - Task: Graph Classification				
UCExplainer$_{sin}$	**0.972**(±**0.003**)	**0.768**(±**0.015**)	13.681(±3.149)	0.782(±0.011)	**0.018**(±**0.010**)
EGO	0.015(±0.008)	0.562(±0.315)	**2.301**(±**0.101**)	*n.a.*	0.892(±0.025)
Random Edges	0.378(±0.003)	−0.033(±0.014)	2.566(±0.012)	*n.a.*	0.292(±0.004)
Random Features	0.183(±0.035)	0.652(±0.091)	26.421(±1.401)	0.944(±0.009)	*n.a.*
CFF	0.018(±0.003)	0.042(±0.946)	4.325(±3.382)	*n.a.*	0.666(±0.194)
CF-GNNExplainer	0.458(±0.019)	−0.034(±0.034)	2.618(±0.022)	*n.a.*	0.076(±0.005)
GNNExplainer	0.253(±0.060)	0.401(±0.154)	4.602(±0.932)	**0.128**(±**0.002**)	0.024(±0.005)
PGExplainer [2]	0.218(±0.019)	0.026(±0.008)	3.921(±0.472)	*n.a.*	0.498(±0.035)

Table 3.2: Results for the CiteSeer dataset (node classification task) and the AIDS dataset (graph classification task)

Scheduling Policies

We explored several scheduling policies to control the trade-off between edge, edge attributes, and node features modifications in our loss function. These different strategies result in multiple variants of our UCExplainer method. Specifically, we consider the following distinct approaches:

The *linear* policy (UCExplainer$_{lin}$): it lets α decrease linearly from 1.0 to 0.0 and transfers the weight on β, and λ over the course of training.

[1] Results obtained using directly the code provided by the authors https://github.com/idea-iitd/InduCE including the missing metrics and dataset

[2] Results obtained from the version implemented in Torch Geometric adapted for counterfactual generation

The *exponential* policy (UCExplainer$_{exp}$): it implements an exponential annealing schedule that shifts focus from node perturbations to edges and edge attributes as follows:

$$\alpha(t) = e^{\frac{-t}{\delta}}, \quad \beta(t) = \lambda(t) = \frac{\left(1 - e^{\frac{-t}{\delta}}\right)}{2}$$

The primary weight α decays from 1 towards 0, t is the current epoch and δ is a decay-rate hyperparameter. The remaining weight is then distributed equally between the edge structure and edge attribute components.

The *sinusoidal* policy (UCExplainer$_{cos}$): it sets $\alpha, \beta,$ and λ as follows:

$$\alpha = \frac{1}{2} \cdot \left(1 + \cos\left(\pi \cdot \frac{epoch}{epochs_{max}}\right)\right), \quad \beta = \lambda = \frac{\left(1 - \alpha^t\right)}{2}$$

resulting in a cosine-shaped decay.

The *dynamic* policy (UCExplainer$_{dyn}$) adjusts each value based on the relative magnitudes of the edge, node feature, and edge attributes loss, setting the lowest loss value to 0.

The *constant* policy (UCExplainer$_{def}$) set all the values as user chosen constants.

Results

In this section, we analyze the experimental results in Table 3.2. The other results are reported in Appendix A.1.2.

The first observation that stands out is the consistently high *validity* of UCExplainer. This indicates that UCExplainer can easily generate counterfactual explanations. In contrast, other methods exhibit much lower validity, often falling below 0.5. Turning our attention to *fidelity*, which measures how closely the generated counterfactuals align with the decision boundary, we see that UCExplainer performs remarkably well. While methods such as CFF occasionally achieve slightly higher fidelity in specific settings, they often suffer from reduced validity or increased sparsity. UCExplainer consistently ranks among the best-performing methods in fidelity, reinforcing its ability to generate explanations that are not only valid but also faithful to the underlying model.

Distribution Distance is another important metric, as lower values indicate that the generated counterfactuals remain within the natural data distribution. Although EGO and CF-GNNExplainer achieve competitive results in some cases, their poor validity makes these results less meaningful. UCExplainer, while not always achieving the lowest distribution distance, maintains a strong balance by ensuring both validity and fidelity remain high. This balance is critical in real-world applications, where counterfactuals must not only be feasible but also realistic. When considering *sparsity*, both in terms of nodes and edges, UCExplainer once again demonstrates its superiority generating explanations with minimal perturbations. A deeper comparison with baseline methods reveals further insights. Random Features, for instance, occasionally achieves high validity, but its counterfactuals are highly unrealistic, as indicated by their excessively high distribution distance.

Random Edges, instead, tends to perform poorly in fidelity and sparsity, demonstrating that randomly modifying graph structures does not produce meaningful counterfactual explanations. Among structured methods, CF-GNNExplainer exhibits relatively low distribution distance and reasonable edge sparsity, but its lower validity and fidelity scores limit its overall usefulness. CFF,

on the other hand, achieves the highest fidelity in some cases, but at the cost of poor validity and increased sparsity. This indicates that while CFF can produce highly faithful explanations, they are often unrealistic or overly complex.

EGO, while getting a low distribution distance in some instances, suffers from extremely low validity scores. The adaptation of GNNExplainer instead seems to perform poorly in almost every case. It has low validity, and often cannot find counterfactual samples at all.

Overall, we attribute these outstanding results to our method's ability to optimally balance node feature and edge perturbations, leading to superior counterfactual explanations.

Ablation Study

We performed an ablation study to understand how different perturbations impact the counterfactual generation. We tested out explainer on the Cuneiform dataset using the GINEConv model. We tried all the possible combinations, namely: nodes features perturbation only (UCExplainer$_{\mathbf{X}}$), edge attributes perturbation only (UCExplainer$_{\mathbf{EA}}$), edge perturbation only (UCExplainer$_{\mathbf{E}}$), nodes features and edge attributes perturbation (UCExplainer$_{\mathbf{EA+X}}$), nodes features and edge perturbation (UCExplainer$_{\mathbf{X+E}}$), edge attributes and edge perturbation (UCExplainer$_{\mathbf{E+EA}}$), and all three together (UCExplainer$_{\mathbf{E+X+EA}}$). In order to deactivate the different perturbations we exclude the perturbation matrices from the gradient computation, otherwise we put the corresponding weight to 1. The results in Table 3.3 highlight a key trade-off between perturbing node features and edges. Specifically, perturbing only node features yields the highest validity. In contrast, perturbing only the edges produces a lower (better) distribution distance, a trend consistent with other explainers that alter the graph structure. Furthermore, our analysis of sparsity shows that when perturbations are applied to multiple graph components simultaneously, the changes are distributed, resulting in lower sparsity for any individual component. Across all configurations, however, fidelity remained consistently low.

Explainer	Validity ↑	Fidelity ↑	Distr. Distance ↓	Node Spars. ↓	Edge Spars. ↓	Edge Attr. Spars. ↓
UCExplainer$_{\mathbf{EA}}$	0.395(±0.029)	0.003(±0.0001)	515.335(±158.069)	*n.a.*	*n.a.*	0.204(±0.008)
UCExplainer$_{\mathbf{X}}$	**0.784(±0.008)**	0.021(±0.0001)	1043.245(±60.207)	0.334(±0.002)	*n.a.*	*n.a.*
UCExplainer$_{\mathbf{E}}$	0.098(±0.002)	0.005(±0.0001)	**131.859(±35.004)**	*n.a.*	0.245(±0.019)	*n.a.*
UCExplainer$_{\mathbf{EA+X}}$	0.771(±0.007)	0.018(±0.003)	1021.977(±26.032)	0.312(±0.002)	*n.a.*	0.397(±0.001)
UCExplainer$_{\mathbf{E+EA}}$	0.272(±0.016)	0.003(±0.0001)	372.094(±87.821)	*n.a.*	0.021(±0.07)	**0.122(±0.001)**
UCExplainer$_{\mathbf{X+E}}$	0.732(±0.011)	**0.022(±0.0004)**	941.005(±61.039)	0.291(±0.006)	0.014(±0.008)	*n.a.*
UCExplainer$_{\mathbf{E+X+EA}}$	0.692(±0.01)	0.012(±0.0002)	911.390(±11.719)	**0.259(±0.003)**	**0.003(±0.0001)**	0.345(±0.004)

Table 3.3: Ablation study using GINEConv on the Cuneiform dataset

Comparison with generative approaches

Recently, generative approaches have been used to build counterfactual explanations for graphs. In particular CLEAR and D4Explainer. Contrary to these approaches, UCExplainer takes a non-generative, training-free approach. Specifically, our framework generates counterfactuals via direct optimization, without the need to train a separate generative model. This makes UCExplainer immediately applicable, which is particularly advantageous in scenarios with limited data (where training a VAE or diffusion models is impractical) or large datasets (where training time becomes prohibitive). We included CLEAR and D4Explainer in our experimental evaluation using the OGBG-MolHIV dataset.

Explainer	Validity ↑	Fidelity ↑	Distr. Distance ↓	Node Spars. ↓	Edge Spars. ↓
UCExplainer$_{sin}$	**0.88**(±**0.000**)	**0.893**(±**0.007**)	4.934(±0.212)	**0.464**(±**0.002**)	**0.001**(±**0.000**)
PGExplainer [2]	0.413(±0.18)	0.287(±0.023)	7.312(±0.86)	*n.a.*	0.52(±0.072)
CLEAR [3]	0.502(±0.001)	0.01(±0.007)	22.48(±0.002)	1.000(±0.000)	0.8092(±0.03)
D4Explainer [4]	0.765(±0.023)	0.561(±0.007)	**2.481**(±**0.002**)	*n.a.*	0.231(±0.03)

Table 3.4: Dataset: OGBG-MolHIV - Model: GCNConv - Task: Node Classification.

Results in Table 3.4 show that CLEAR performs poorly in fidelity (0.01) and has a relatively low validity (0.50), high sparsity, and high distribution distance (22.49), leading to less interpretable and less realistic explanations. D4Explainer instead reaches the lowest distribution distance with the second highest validity (0.765) and fidelity (0.561), with fairly low edge sparsity. UCExplainer offers instead high validity (0.88) and fidelity (0.89) with less sparse explanations and a relatively low distribution distance (4.93) indicating that counterfactuals are close to the data manifold.

The Impact of the Scheduling Policy

The results obtained with different scheduling policies for $\alpha, \beta,$ and λ in Table 3.5 show that the policy has an impact on every measured metrics. In particular with this dataset and model the *sin* policy reaches the highest validity, and a comparatively good fidelity, distribution distance, node and edge sparsity. Anyhow, we cannot conclude that one policy is better than the others since there are clear tradeoff between policies. The optimal choice of $\alpha\,\beta,$ and λ should therefore be carefully tuned based on the complexity of the graph data and the interpretability objectives of the counterfactual explanations. In Figure 3.3 it is possible to see the trends for the non-trivial policies.

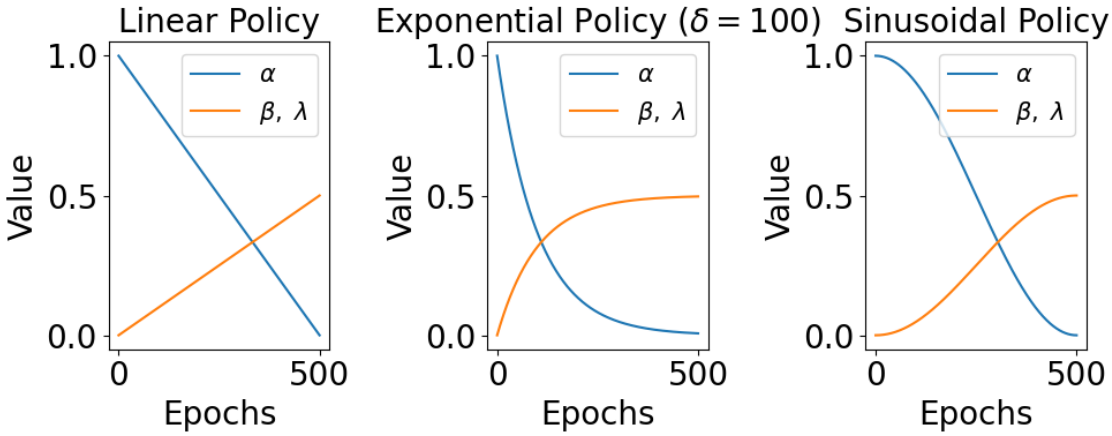


Figure 3.3: Scheduling policies trends

Feasibility of our Method

In this section, we discuss the feasibility of our method in comparison with the baselines.

In Table 3.6, for example, we compare the execution time and the memory needed for each explainer on the Citeseer dataset.

[3]Results obtained using directly the code provided here https://github.com/jma712/GraphCFE including the missing metrics

[4]Results obtained using directly the code provided by the authors https://github.com/Graph-and-Geometric-Learning/D4Explainer including the missing metrics and dataset

Explainer	Validity ↑	Fidelity ↑	Distr. Distance ↓	Node Spars. ↓	Edge Spars. ↓
UCExplainer$_{def}$	0.955(±0.006)	0.764(±0.015)	13.393(±3.070)	0.822(±0.021)	0.252(±0.063)
UCExplainer$_{dyn}$	0.932(±0.014)	0.771(±0.018)	13.268(±3.051)	0.820(±0.021)	0.010(±0.002)
UCExplainer$_{exp}$	0.873(±0.035)	**0.775(±0.005)**	11.926(±2.970)	0.828(±0.021)	**0.002(±0.003)**
UCExplainer$_{lin}$	0.960(±0.009)	0.767(±0.010)	13.891(±3.006)	**0.781(±0.007)**	0.013(±0.006)
UCExplainer$_{sin}$	**0.972(±0.003)**	0.768(±0.015)	13.681(±3.149)	0.782(±0.011)	0.018(±0.010)

Table 3.5: Scheduling policies study, Dataset: AIDS, Task: Graph Classification, Model: GCNConv

Explainer	Time (*Seconds*)
UCExplainer	7.665(±0.245)
CF-GNNExplainer	45.576(±1.306)
Random Features	1.287(±0.025)
Random Edges	1.350(±0.017)
EGO	0.011(±0.002)
CFF	8.936(±0.204)
UNR	0.266(±0.327)
GNNExplainer	1.67(±0.31)
PGExplainer	0.733(±0.0012)
InduCE	8.352(±1.653)

Table 3.6: Execution time comparison for different explainers on the Citeseer dataset.

The effectiveness of our edge weight sparsification technique is evident when analyzing the execution time results in Tables 3.6 and 3.7. Notably, CF-GNNExplainer requires 45.576 seconds on average to generate explanations, whereas UCExplainer completes the same process in just 7.665 seconds. This substantial improvement in runtime efficiency highlights the scalability advantage introduced by our sparsification technique. By reducing the computational overhead associated with handling edge deletions, UCExplainer maintains high explainability performance while significantly lowering execution time. Generative approaches instead are generally much faster during inference, for example, CLEAR takes 0.03 seconds to generate an explanation but requires 450 seconds of training on an RTX 4090 before being able to produce counterfactuals. The same observation holds for D4Explainer. PGExplainer instead has a much lower training time thus is faster than our approach but has shown in Section 3.2.4 and Appendix A.1.2 the performances are lower.

3.2.5 Conclusion and Future Work

In this work, we introduced UCExplainer, a unified counterfactual explainer for Graph Neural Networks (GNNs) that integrates both node feature, edge attributes, and structural perturbations. Through extensive experiments across various datasets, tasks, and architectures, we demonstrated that UCExplainer effectively balances key evaluation metrics, ensuring high validity while minimizing modifications to the graph structure and node features to maintain realism.

We also proposed a novel edge weight sparsification technique, which significantly improves computational efficiency without compromising explainability. Our comparative analysis showed that UCExplainer operates more efficiently and with lower computational costs than existing methods following a similar approach, such as CF-GNNExplainer. Additionally, we explored different scheduling policies for balancing node and edge perturbations, further highlighting the flexibility and generalizability of UCExplainer across diverse scenarios.

In summary, UCExplainer represents a state-of-the-art counterfactual explanation framework for

Explainer	**Time (*Seconds*)**
UCExplainer	5.34(±0.098)
PGExplainer	0.598(±0.0012)
CLEAR	0.429(±0.023)
D4Explainer	0.981(±0.01)

Table 3.7: Execution time comparison between generative based explainers and UCExplainer on the OGBG-MolHIV dataset.

GNNs, offering a comprehensive and computationally efficient approach that aligns with real-world interpretability requirements.

Future work will focus on extending our approach to broader graph-based tasks, including link prediction, as well as adapting the counterfactual framework to dynamic and heterogeneous graph structures.

marion

3.3 Graphs Explainability and Large Language Models

In the previous section, we introduced a new method for generating counterfactual explanations for GNN. In this section, we investigate a further step toward human understanding of such explanations, namely, the creation of narratives for graph counterfactual explanations. In recent years, Machine Learning has become deeply embedded in various facets of our daily lives, influencing decisions ranging from personalized recommendations to critical judgments in healthcare and finance. This pervasive integration has raised significant concerns regarding transparency and accountability, leading to legislative actions such as the European Union's General Data Protection Regulation (GDPR) [194], which emphasizes the right to explanation for automated decision-making processes. Similarly, the upcoming EU Artificial Intelligence Act aims to regulate AI systems, mandating explainability and interpretability in high-risk applications [60].

In the realm of eXplainable Artificial Intelligence (XAI), numerous techniques have been proposed to address this need. Among them, *counterfactual explanations* [195] stand out as one of the most promising *post-hoc* methods. The core idea is to explain a model's prediction by identifying a *counterfactual example* – i.e., the minimal change to the input that leads to a different prediction. Counterfactual explanations have been successfully applied to unveil the inner workings of predictive models having various degrees of complexity, ranging from ensembles of decision trees [185, 188] and multi-layer perceptrons [140] to more sophisticated models like transformers [35] or model-agnostic techniques such as [41, 42, 145].

Despite these advancements, a significant challenge remains: translating algorithmically generated explanations into a "format" that is accessible and comprehensible to end-users who may not possess technical expertise. [67] takes a first step in this direction by leveraging the generative capabilities of Large Language Models (LLMs) to produce user-friendly counterfactual explanations in natural language.

Building on the work above, we tackle the more ambitious challenge of generating natural language explanations from counterfactual examples tailored for graph neural networks (GNNs). Indeed, counterfactual examples derived from graph inputs are inherently more complex than those from tabular data, given the intricate relationships and structures they capture (e.g., nodes, edges, and their dependencies). This complexity presents unique challenges that demand a specialized approach to translate them into universally comprehensible natural language explanations.

Moreover, graph data is prevalent across various domains such as financial decision-making [30], fraud detection [197], structured text extraction [29], and biology [64], where GNN-based models have shown remarkable predictive performance by encoding high-order structural relationships within their learned graph representations.

Specifically, we consider the counterfactual examples output by a generic graph counterfactual explainer designed for node classification tasks using GNNs. Then, we instruct several open-source LLMs to translate these "raw" counterfactual examples into coherent natural language explanations that are also accessible to non-expert users. To evaluate the quality of the generated explanations, we introduce a set of novel metrics that measure how accurately our method maps the counterfactual examples to their corresponding natural language descriptions. As openly generated text might be hard to evaluate using automatic evaluation metrics [45], we also validate this evaluation through human judgment. Extensive experiments conducted using two graph counterfactual explainers for

node classification – CF-GNNExplainer [128] and its extension for node features, CF-GNNFeatures [74] – across several graph datasets and multiple open-source LLMs demonstrate that our method can effectively support decision-making processes in critical domains through the generation of natural language explanations.

3.3.1 Related Works

Counterfactual explainability has emerged as a pivotal approach for interpreting complex machine learning models by illustrating how changes in input variables can lead to different outcomes. Despite this, a significant challenge persists: making these explanations accessible and understandable to a broad audience, particularly non-technical users who may struggle with abstract mathematical concepts and technical jargon. The intricacy of counterfactual explanations often hinders their comprehension among laypersons, limiting their practical utility in real-world applications where user trust and transparency are paramount.

The advent of large language models (LLMs), such as GPT-4[4], Qwen2.5 [184], LLama [189], Mistral [106], and their successors [9], has revolutionized the field of natural language processing. These models possess an unparalleled capacity for understanding and generating human-like text, making them invaluable tools for translating complex technical information into plain language. Their widespread availability and ease of integration into various platforms further enhance their appeal for tasks requiring sophisticated language generation and interpretation.

Leveraging the immense capabilities of LLMs presents a promising solution to the accessibility problem in counterfactual explainability. By converting intricate data-driven explanations into natural language narratives, LLMs can make the insights derived from machine learning models more digestible for non-technical users. This translation not only aids in user comprehension but also fosters greater trust in automated systems by promoting transparency.

In this context, the pioneering work done by [67] marks a significant milestone. Their research laid the foundations for utilizing LLMs to transform data-based counterfactual explanations into coherent, user-friendly language. After generating the counterfactual examples, the researchers aimed to identify the primary causal factors deduced from these examples that led to the user's differing classification. To achieve this, they input both the set of counterfactual examples and the original user data into a Large Language Model (LLM), instructing it to produce a list of the main reasons why the user was classified differently. Once the LLM generated this list, they meticulously verified its accuracy and identified the most relevant causes contributing to the explanation.

Subsequently, they synthesized all the information produced and leveraged the LLM once more to generate a final explanation articulated in plain language. This explanation was crafted to emphasize actionable steps that the user could take to alter their input data or behavior, thereby changing their classification to the desired category. By doing so, they not only provided a transparent rationale behind the classification but also offered practical guidance for the user to achieve a favorable outcome. However, their work focuses solely on the relatively straightforward case of translating counterfactual examples derived from a single, well-known tabular dataset. Furthermore, they used a proprietary LLM model (GPT-4o), making it hard to replicate their approach.

To the best of the authors' knowledge, there are no papers in the State-of-the-Art that address the problem of translating the explanations generated via counterfactual explainers into natural language explanations; for this reason, we firmly believe that our work can be useful to the com-

munity.

3.3.2 Background

Graph Neural Networks (GNNs) have emerged as a powerful class of machine learning models specifically designed to handle graph-structured data. Graphs are a natural representation for many real-world problems, such as social networks, recommendation systems, molecular chemistry, and knowledge graphs, where entities are represented as nodes and their relationships as edges. Unlike traditional neural networks, GNNs explicitly account for the relational structure of data, making them particularly effective for tasks like node classification, link prediction, and graph classification.

At the core of GNNs lies the concept of message passing or neighborhood aggregation, where nodes iteratively aggregate information from their neighbors to learn contextualized and high-order representations. The resulting node embeddings capture both the local structure and node attributes, enabling downstream tasks on graph data. This framework was formalized in seminal works like Graph Convolutional Networks (GCN) by [114], which introduced a spectral perspective on graph convolutions.

Since the introduction of GCNs, several variants have been developed to address specific limitations or enhance capabilities. For example, Graph Attention Networks (GAT), introduced by [191], employ an attention mechanism to weigh the contributions of neighboring nodes dynamically. GraphSAGE by [87] enables inductive learning by aggregating sampled neighbor information through mean, LSTM, or pooling operations. [214] proposed Graph Isomorphism Network (GIN), which focused on designing a more expressive GNN capable of distinguishing graph structures that previous architectures could not. Finally, extensions like R-GCN by [171], address edge types and heterogeneous graph structures, which are common in knowledge graphs and multi-relational data.

The versatility of GNNs has led to their application across a range of domains, such as community detection and friend recommendations ([158]) in social networks, personalized suggestions based on complex user-item interaction graphs ([201]) in recommender systems, molecular property prediction, and drug discovery ([72]) in biochemistry, and entity linking and relation extraction ([199]) from knowledge graphs. For a comprehensive survey on GNNs, we invite the reader to refer to [210].

3.3.3 From Counterfactual Examples to Natural Language Narratives

This section begins by formalizing the counterfactual explanation problem in a classification task, with a particular focus on graph-structured inputs. It then introduces our proposed method, which generates counterfactual examples using a dedicated explainer and subsequently translates these examples into natural language explanations via a pre-trained large language model (LLM). Finally, the section presents a new evaluation framework that defines a suite of metrics—such as target node identification, counterfactual class identification, and target node feature and neighbor extraction—to quantitatively assess the quality and coherence of the generated natural language explanations.

The Counterfactual Explanation Problem

The counterfactual explanation problem in a classification task is described as follows. Given a sample x and a predictive model $f_{\boldsymbol{\theta}}$ parametrized by $\boldsymbol{\theta}$ – hereinafter referred to as *oracle* – the goal is to find a sample $x' \neq x$ such that $f_{\boldsymbol{\theta}}(x) \neq f_{\boldsymbol{\theta}}(x')$. This x' is called a *counterfactual example* for x. Among all potential counterfactual examples (if any), we assume the existence of a counterfactual example generator g, which takes as input the original instance x and returns a counterfactual x^*, where the distance $d(x, x^*)$ is minimized, or $\perp$ if no valid counterfactual example exists. The distance function $d(\cdot, \cdot)$ ensures that the counterfactual sample x^* remains as close as possible to the original factual sample x.

Note that, in this work, we focus on graph inputs. Specifically, each sample x and its counterfactual(s) x' can be represented as $G(V, E)$ and $G'(V', E')$, respectively. Therefore, a counterfactual example for an input graph G will be a new graph G', where either the node features, the structural links, or both differ from the original, while still satisfying the counterfactual criterion of altering the model's prediction. This introduces additional challenges as the modifications can affect both the node-level properties and the overall graph topology.

Proposed Method

We assume to have an oracle $f_{\boldsymbol{\theta}}$ for a node classification task, specifically a graph neural network trained on a given input graph. For any node instance x, we know both its predicted label $\hat{y}$, such that $f_{\boldsymbol{\theta}}(x) = \hat{y}$, and its optimal counterfactual example $x^* = g(x)$, which is generated by the counterfactual explainer g.

To generate the natural language explanation ($e(x^*)$) associated with the generic counterfactual example (x^*), we feed a pre-trained LLM m with the factual (x) and counterfactual (x^*) instances along with a specific prompt p, i.e., $e(x^*) = m(p, x, x^*)$.

One of the critical challenges of this approach is how to validate the quality of the generated natural language explanations through the LLM. In the following section, we introduce the new evaluation framework proposed in this work.

A New Evaluation Framework

Given the novelty of the problem we are tackling, to the best of our knowledge, no established quality metrics exist in the literature to rigorously evaluate explanations generated by LLMs for graph counterfactuals. The only commonly adopted approach is based on subjective human judgment, which may introduce variability and bias in the assessment process. To address this gap and ensure a more systematic evaluation, we have developed a suite of novel metrics that provide a quantitative and objective assessment of the language model's capability to articulate pairs of factual and counterfactual graphs into coherent and informative natural language explanations.

For clarity, we denote the factual graph as G and the corresponding counterfactual graph as G'. To compute these metrics effectively, we structured our prompts to include a request for the language model to populate a predefined dictionary with essential graph information. This dictionary encompasses the target node of the classification task, its original class in the factual graph, its modified class in the counterfactual graph, the neighborhood of the target node in both G and G', and the set of features associated with the target node in each scenario.

By doing so, we aim to evaluate the language model's ability to discern and convey the critical elements of the graph structure and its transformations, thereby assessing the model's comprehension of how the changes in the graph influence the classification outcome for the target node. This framework allows us to objectively measure the performance of the language model in translating structural and attribute-based differences between G and G' into precise and meaningful natural language descriptions.

Target Node Identification (TNI) This metric assesses the ability of the LLM to accurately identify and reference the target node within the graph structure. Given the importance of the target node as the focal point of the classification task, correctly pinpointing it is crucial for generating valid explanations. Formally, given a graph $G(V, E)$, the target node $t \in V$, and a node $v \in V$ predicted by the LLM, we define the Target Node Identification (TNI) metric as:

$$\text{TNI}(v) = \begin{cases} 1 & \text{if } v = t, \\ 0 & \text{otherwise.} \end{cases}$$

Counterfactual Class Identification (CCI) This metric evaluates the capacity of the LLM to correctly comprehend and express the change in class assignment of the target node from the factual graph G to the counterfactual graph G'. Let $G' = (V, E')$ be the factual, such that $f_{\boldsymbol{\theta}}(G') = c'$ where c' is the counterfactual class of the target node t such that $f_{\boldsymbol{\theta}}(G') \neq f_{\boldsymbol{\theta}}(G)$, and let c_{LLM} be the counterfactual class predicted by the LLM, CCI is defined as:

$$\text{CCI}(c_{\text{LLM}}) = \begin{cases} 1 & \text{if } c_{\text{LLM}} = c', \\ 0 & \text{otherwise.} \end{cases}$$

Factual Target Node Feature (FTNF) This metric examines the LLM's ability to accurately recognize and describe the features associated with the target node in the factual graph G. Correctly identifying these features is essential, as they are pivotal in determining the node's initial classification. Let $G = (V, E)$ be the factual graphs. Let $t \in V$ be the target node, and $\mathbf{x}_t \in \mathbb{R}^d$ denote the feature vectors of node t in the factual graph G. We also define the vector of features predicted by the LLM for the target node t as $\mathbf{x}_{\mathbf{LLM}}$, the metric is computed as:

$$\text{FTNF}(\mathbf{x}_{\mathbf{LLM}}) = \begin{cases} 1 & \text{if } \mathbf{x}_{\mathbf{LLM}} = \mathbf{x}_t, \\ 0 & \text{otherwise.} \end{cases}$$

Counterfactual Target Node Feature (CFTNF) Similar to FTNF, this metric evaluates the LLM's capability to identify and articulate the features associated with the target node in the counterfactual graph G'. This metric is critical for assessing whether the LLM captures the differences in node attributes that lead to a shift in classification. Let $G' = (V', E')$ be the counterfactual graphs. Let $t \in V'$ be the target node, and $\mathbf{x}'_t \in \mathbb{R}^d$ denote the feature vectors of node t in the counterfactual graph G'. We also define the vector of features predicted by the LLM for the target

node t as $\mathbf{x_{LLM}}$, the metric is computed as:

$$\text{CFTNF}(\mathbf{x_{LLM}}) = \begin{cases} 1 & \text{if } \mathbf{x_{LLM}} = \mathbf{x}'_t, \\ 0 & \text{otherwise.} \end{cases}$$

Counterfactual Target Node Neighbors (CFTNN) This metric measures the LLM's capacity to correctly identify the set of neighboring nodes for the target node in the counterfactual graph G'. Understanding these neighbors in the counterfactual context is essential, as changes in the neighborhood structure may directly influence the target node's classification shift. Accurately capturing the neighbors in G' helps the LLM articulate how the local connectivity of the target node has been modified and how this alteration affects the node's role and classification within the network. Let $G'(V', E')$ be a counterfactual graph, let $t' \in V'$ be the target node in the graph G', and let $\mathcal{N}(t') = u \in V' \mid (t', u) \in E'$ denote the set of neighbors of the target node t. Given a set of neighbors $\mathcal{N}_{\text{LLM}}(t')$ predicted by the LLM for the target node t', we define the Counterfactual Target Node Neighbors (CFTNN) metric as:

$$\text{CFTNN}(\mathcal{N}_{\text{LLM}}(t')) = \begin{cases} 1 & \text{if } \mathcal{N}_{\text{LLM}}(t') = \mathcal{N}(t'), \\ 0 & \text{otherwise.} \end{cases}$$

Overall, these metrics provide a comprehensive framework for evaluating the language model's capacity to interpret and describe the structural and feature-based transformations between factual and counterfactual graphs. By assessing these distinct aspects of graph understanding, we can quantitatively measure the model's ability to generate coherent and insightful explanations that accurately reflect the graph dynamics and their implications on classification outcomes. We decided to select the explanations that score 1 on at least 5 out of 6 metrics and test them to human judgment.

3.3.4 Experiments

Given the general framework described in Section 3.3.3, this section presents the experimental setup and evaluation of our approach for generating counterfactual explanations in graph neural networks using large language models (LLMs). Specifically, we investigate the problem of translating counterfactual modifications into human-interpretable natural language descriptions.

To comprehensively assess the effectiveness and generalizability of our method, we conduct experiments on two distinct tasks: node classification and graph classification. In the node classification setting, the goal is to generate counterfactual explanations that describe changes necessary to alter the classification of an individual node within a larger graph. In contrast, in the graph classification task, counterfactuals are generated at the graph level, identifying modifications that would change the overall label assigned to the entire graph. To ensure a robust evaluation, we employ real-world datasets widely used in graph-based machine learning.

Node Classification Graph Counterfactual Explainers

As outlined in Section 3.3.3, our proposed pipeline requires the integration of a counterfactual explainer for node classification, denoted as g. In this study, we employed two state-of-the-art

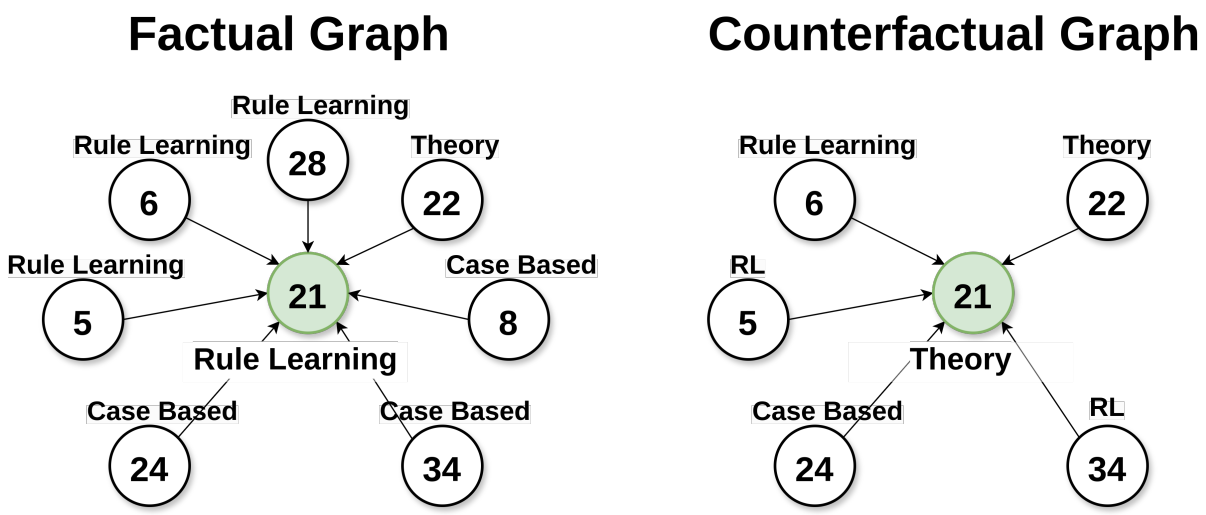


Figure 3.4: On the left, the factual graph. On the right, the counterfactual graph is computed using CF-GNNExplainer. Each graph contains node ids and classes

explainers, namely, CF-GNNExplainer and CF-GNNFeatures, allowing us to investigate the impact of distinct graph modifications on the LLM's ability to generate natural language explanations. Specifically, CF-GNNExplainer modifies the graph structure by perturbing the adjacency matrix, while CF-GNNFeatures focuses on altering the node attributes. This dual approach enables us to assess how variations in both structural and feature-based properties influence the quality and coherence of the generated explanations.

CF-GNNExplainer is a perturbation-based counterfactual explainer. It defines $\bar{\mathbf{A}}_v = \mathbf{P} \odot \mathbf{A}_v$, where $\mathbf{P}$ is a binary perturbation matrix that sparsifies $\mathbf{A}_v$. The goal is to find $\mathbf{P}$ for a given node v such that $f_{\boldsymbol{\theta}}(\mathbf{A}_v, x) \neq f_{\boldsymbol{\theta}}(\mathbf{P} \odot \mathbf{A}_v, x)$. To find $\mathbf{P}$, CF-GNNExplainer exploits a technique to train sparse neural networks to zero out entries in the adjacency matrix (i.e., removing edges). This results in the deletion of the edge between node i and node j.

CF-GNNFeatures is a node features perturbation-based counterfactual explainer. Given a graph $G(V, E)$, two matrices are defined, namely: $\mathbf{V}_x$, the node features matrix representing the features for every node in G and $\mathbf{P}_x$, the feature perturbation matrix. Initially, the matrix $\mathbf{P}_x$ is filled with ones to maintain the original sets of attributes. Given an oracle $f_{\boldsymbol{\theta}}$, parameterized by θ, we fix all the weights and train $\mathbf{P}_x$ to change the attribute matrix $\mathbf{V}_x$ that is fed into the oracle, multiplying the current feature matrix with the perturbation matrix. In Figure 3.4 you can see an example of a factual and its counterfactual graph computed using CF-GNNExplainer.

Large Language Models

In this study, we utilized a family of state-of-the-art LLMs, namely Qwen2.5 [184], to generate natural language explanations. We experimented with different model sizes in terms of parameters, specifically employing the 0.5B, 1.5B, 3B, 7B, and 14B variants.

The model configuration used in this work includes several hyperparameters that influence its behavior during text generation:

- temperature: 0.1,

- top-p: 0.8,
- top-k: 10,
- repetition penalty: 1.05,
- max output tokens: 2048

In order to reduce the memory footprint of the models, we used GPTQ [66] quantization with 4-bit integer precision (`int4`).

Overall, the chosen configuration allows the model to generate meaningful counterfactual explanations for complex graph structures while maintaining computational efficiency. The setup leverages the power of modern LLMs along with advanced quantization techniques to produce high-quality outputs, making it an ideal choice for applications requiring detailed textual descriptions of graph perturbations.

Given the factual graph: ***$factual graph description$*** and given the counterfactual example: ***$counterfactual graph description$*** and given the knowledge base about the dataset: ***$dataset knowledge$***, fill the dictionary and provide an explanation about the change in classification for the target node, please evaluate also the influences of neighbors nodes.

Prompt used to obtain explanations.

The target node (node 25) was originally classified as **Genetic Algorithms** with a rich set of attributes. By removing certain attributes like *'adapt', 'constrain', 'audio', 'interoper', 'dataflow', 'visibl', 'see', 'gateway', 'realtim', 'exploratori', 'skill', 'almost', 'adversari', 'confid', and 'chapter'*, the node's representation became simpler and more aligned with the characteristics of nodes classified as **Neural Networks**. Additionally, the neighboring nodes (24 and 41) also changed their classification from **Genetic Algorithms** to **Neural Networks**, further influencing the reclassification of node 25.
The original target node features were: *['schedul', 'adapt', 'constrain', 'audio', 'interoper', 'account', 'dataflow', 'gestur', 'light', 'visibl', 'see', 'intrus', 'gateway', 'realtim', 'exploratori', 'skill', 'almost', 'adversari', 'confid', 'oracl', 'gigabit', 'chapter']*

Response generated by Qwen2.5-14B with CF-GNNFeatures.

Figure 3.5: Prompt–response pair for natural-language counterfactual explanations.

Prompting The design and structure of the prompt is critical, as the performances of LLMs are highly influenced by how the graph data is presented. To this end, we adopted the incident representation framework proposed by [62], with some adjustments to better suit the requirements of our task. To provide all the necessary information for the LLM, we utilize an initial system prompt that provides essential background information on the challenges of counterfactual explainability, particularly in the context of graph-based data. Subsequently, the counterfactual prompt is introduced (Figure 3.5), which instructs the LLM to generate a coherent and contextually appropriate natural language explanation based on both the factual and counterfactual graph samples.

Experimental Setup

The experiments were conducted on a machine equipped with 64GB of RAM, an NVIDIA RTX 4090, and an AMD Ryzen 9 7900 processor. As oracle, we used a 2-layer GCN trained for 500 epochs with a learning rate of 0.001.

# Parameters	TNI ↑	CCI ↑	FTNF ↑	CFTNF ↑	FTNN ↑	CFTNN ↑
0.5B	0.000	0.000	0.000	0.000	0.000	0.000
1.5B	0.048	0.048	0.022	0.007	0.030	0.030
3B	0.391	0.336	0.092	0.074	0.221	0.258
7B	0.292	0.284	0.162	0.085	0.007	0.007
14B	**0.720**	**0.720**	**0.668**	**0.565**	**0.528**	**0.524**

Table 3.8: Evaluation metrics using CF-GNNFeatures and Cora dataset for graph understanding.

# Parameters	TNI ↑	CCI ↑	FTNF ↑	CFTNF ↑	FTNN ↑	CFTNN ↑
0.5B	0.000	0.000	0.000	0.000	0.000	0.000
1.5B	0.053	0.053	0.039	0.026	0.000	0.013
3B	0.237	0.197	0.105	0.132	0.066	0.105
7B	0.171	0.118	0.053	0.118	0.026	0.000
14B	**0.618**	**0.618**	**0.579**	**0.579**	**0.539**	**0.500**

Table 3.9: Evaluation metrics using CF-GNNExplainer and Cora dataset for graph understanding.

Datasets

To ensure a comprehensive evaluation, we employed two well-known citation network datasets: Cora and CiteSeer. The CiteSeer dataset comprises 3,312 scientific publications categorized into six distinct classes, connected by 4,732 citation links. Each publication is represented by a binary word vector, where each entry indicates the absence or presence of a specific word from a dictionary of 3,703 unique terms. Similarly, the Cora dataset contains 2,708 scientific publications classified into seven distinct classes, with 5,429 citation links. Each publication is also represented by a binary word vector corresponding to a dictionary of 1,433 unique terms.

After the counterfactuals had been found, we translated the original node feature vocabulary using actual words rather than binary vectors to facilitate the language model's understanding of the relationship between a node's classification and its features. Since predefined vocabularies for node features are not provided for these datasets, we adhered to the original feature extraction instructions to generate the vocabularies for both datasets.

Results

To provide an appropriate evaluation, we tested multiple counterfactual methods and LLMs on different graph datasets. As shown in Tables from 3.8 to 3.11, the results for the graph understanding as defined in Section 3.3.3 are generally good for models with more parameters. In particular, a clear trend emerges across all tables: as the number of parameters in the LLM increases, performance improves significantly across all metrics, as expected. The smallest model, with 0.5 billion parameters, consistently scores zero across all metrics and datasets, indicating its inability to generate meaningful explanations. With an increase to 1.5 billion parameters, there is a minimal improvement, but performance remains substantially low.

Notable improvements are observed with the 3-billion-parameter model, particularly in metrics such as Target Node Identification (TNI) and Counterfactual Class Identification (CCI). However, it's the largest model, with 14 billion parameters, that achieves the highest scores across all metrics and datasets. This demonstrates the importance of model size when dealing with complex tasks

# Parameters	TNI ↑	CCI ↑	FTNF ↑	CFTNF ↑	FTNN ↑	CFTNN ↑
0.5B	0.000	0.000	0.000	0.000	0.000	0.000
1.5B	0.273	0.280	0.096	0.003	0.174	0.174
3B	0.488	0.394	0.112	0.037	0.320	0.348
7B	0.450	0.447	0.180	0.171	0.314	0.314
14B	**0.879**	**0.873**	**0.705**	**0.481**	**0.758**	**0.758**

Table 3.10: Evaluation metrics using CF-GNNFeatures and Citeseer dataset for graph understanding

# Parameters	TNI ↑	CCI ↑	FTNF ↑	CFTNF ↑	FTNN ↑	CFTNN ↑
0.5B	0.000	0.000	0.000	0.000	0.000	0.000
1.5B	0.000	0.038	0.000	0.000	0.000	0.000
3B	0.385	0.192	0.038	0.077	0.115	0.115
7B	0.385	0.346	0.154	0.269	0.077	0.038
14B	**0.615**	**0.615**	**0.346**	**0.346**	**0.577**	**0.538**

Table 3.11: Evaluation metrics using CF-GNNExplainer and Citeseer dataset for graph understanding.

such as generating natural language explanations from graph counterfactuals.

In Table 3.8, results show that using CF-GNNFeatures on the Cora dataset, the TNI metric increases from 0.000 with the smallest model to 0.720 with the largest model. Similarly, in Table 3.10, using the CiteSeer dataset, the TNI metric reaches 0.879 with the 14-billion-parameter model. These improvements indicate that larger models can understand and generate accurate descriptions of complex graph structures.

Comparing performances across datasets The LLMs generally perform better on the CiteSeer dataset, especially when using the CF-GNNFeatures explainer. For example, in Table 3.10 (CF-GNNFeatures on CiteSeer), the 14-billion-parameter model achieves a TNI of 0.879, higher than the 0.720 achieved on the Cora dataset in Table 3.8.

Several factors contribute to this difference in performance. The datasets have different levels of complexity, differences in feature distributions, or inherent properties that make one more amenable to the LLM's processing capabilities. CiteSeer has more straightforward and distinctive features that the LLM can more readily associate with classification outcomes, aiding in the generation of coherent explanations.

Comparison between explainers The two counterfactual explainers used in the study, CF-GNNFeatures and CF-GNNExplainer, focus on different aspects of the graph. CF-GNNFeatures modifies node features, while CF-GNNExplainer modifies the graph structure.

For instance, in Table 3.8 (CF-GNNFeatures on Cora), the FTNF (Factual Target Node Feature) and CFTNF (Counterfactual Target Node Feature) scores at 14 billion parameters are 0.668 and 0.565, respectively. In contrast, in Table 3.9 (CF-GNNExplainer on Cora), these scores are lower, at 0.579 for both metrics at the same model size.

This suggests that LLMs find it easier to generate explanations when the modifications involve changes in node features rather than structural changes in the graph. Explaining structural changes may require a deeper understanding of the graph's topology and how it influences the classification, which appears more challenging for the LLMs.

Question	CF-GNNE.	CF-GNNF.
Q1	3.00 ± 1.60	4.44 ± 0.80
Q2	2.87 ± 1.47	4.28 ± 0.83
Q3	2.90 ± 1.51	3.56 ± 1.23
Q4	2.92 ± 1.52	3.68 ± 1.20
Q5	3.00 ± 1.55	3.80 ± 1.11

Table 3.12: Average scores (1 to 5) from the human evaluation for the explanations generated using Qwen2.5-14B using counterfactuals from CF-GNNExplainer (CF-GNNE.) and CF-GNNFeatures (CF-GNNF) on the Cora dataset

Human Evaluation To accurately assess the explanations, we conducted a human evaluation study involving a sample of 15 graduate students with diverse skills and backgrounds. Participants were asked to complete a questionnaire designed to evaluate the quality and clarity of the counterfactual explanations. The questionnaire consisted of the following five questions, each rated on a scale from 1 to 5:

- Q1: Is the terminology and language used in the explanation appropriate and easy to understand?
- Q2: How clear and easy to understand is the provided counterfactual explanation?
- Q3: How clearly does the explanation describe the changes in node connections (graph structure) that led to the counterfactual outcome?
- Q4: Are the changes in features and structure easy to interpret and make sense in the context of the original graph?
- Q5: What is your overall assessment of the clarity and coherence of the counterfactual explanation?

The results of the human evaluation can be seen in Table 3.12. An example of natural language translation is illustrated in Figure 3.5. The findings indicate that explanations generated by the LLM that focus on node features are more effective and meaningful to human users than those based on adjacency matrix perturbations. Participants found feature-based explanations to be more accessible in terms of language, clearer in conveying the reasoning, and more interpretable within the context of the graph.

3.3.5 Practical Implications

The proposed framework for generating natural language counterfactual explanations using LLMs holds significant promise for enhancing interpretability and transparency in graph-based machine learning models. This approach has several practical implications across domains that utilize complex graph structures, including financial analysis, healthcare, cybersecurity, and social network analysis. By translating complex counterfactual explanations into natural language, our method makes these explanations more accessible to non-technical stakeholders, including business managers, policymakers, and end-users. Moreover, with the European Union's General Data Protection Regulation (GDPR) and the EU Artificial Intelligence Act, explainability and transparency of AI systems are becoming mandatory, particularly in high-stakes applications. Our method addresses

this requirement by ensuring that counterfactual explanations are technically sound and easily interpretable, thereby aiding compliance with legal and ethical standards. In summary, our framework offers a comprehensive solution for bridging the gap between technical counterfactual explanations and human comprehension, thereby enabling the broader adoption of AI technologies in high-stakes, complex domains. The method promotes transparency and trust, enhancing the utility of counterfactual explanations as a tool for model evaluation, debugging, and refinement.

3.3.6 Conclusions and Future Work

In this paper, we presented a novel framework for generating natural language counterfactual explanations for graph-based models using state-of-the-art LLMs. Our approach leverages the inherent generative capabilities of LLMs to transform complex and technical counterfactual examples into coherent and accessible natural language descriptions. By doing so, we address a critical gap in the literature: the lack of intuitive, user-friendly counterfactual explanations for GNNs. We validated our method across several GNN-based counterfactual explainers and multiple graph datasets, demonstrating its effectiveness through a suite of newly proposed metrics and human evaluation. Our findings show that our framework not only captures the underlying structural and attribute-based transformations within the graphs but also produces explanations that are easily interpretable by non-expert users, thereby enhancing the transparency and usability of graph-based machine learning models.

Our framework does not depend upon any particular counterfactual explainer, so it can be used with any approach. We developed additional metrics that quantify the quality of the natural language output, such as graph understanding. Finally, our experiments were conducted using open-source LLMs without task-specific fine-tuning. Future research could explore the impact of fine-tuning the LLMs on graph-specific tasks or developing domain-specific LLMs to further enhance the quality and relevance of the generated explanations.

Overall, we believe that our framework represents a significant step toward bridging the gap between complex algorithmic explanations and human comprehension in the domain of graph-based machine learning.

3.4 Graph Counterfactual Narratives

In this section, we continue to explore the topic of counterfactual graph narrative generation, extending the work done in the previous section to graph classification.

In recent years, machine learning has become a pervasive force in modern computing, underpinning applications that span personalized recommendation systems [148], natural language processing [139], medical diagnostics [6], and financial forecasting [204]. These models, particularly deep neural networks, have demonstrated exceptional performance by automatically learning complex patterns from large-scale data. However, their increasing integration into decision-making pipelines has raised concerns about their interpretability, trustworthiness, and alignment with human reasoning [71, 215]. As the use of opaque, high-capacity models increases, so does the need for methods that can explain their outputs in a manner understandable to human stakeholders, especially in high-stakes contexts where decisions may have significant social or economic consequences. To this end, regulatory bodies have introduced legal frameworks designed to promote responsible AI. For example, the European Union's General Data Protection Regulation (GDPR) explicitly enshrines a *'right to explanation'* for individuals subject to automated decision-making [194]. Similarly, the upcoming EU Artificial Intelligence Act proposes stricter requirements for AI systems, particularly in high-risk settings, including mandates for explainability and interpretability [60].

Within the field of eXplainable Artificial Intelligence (XAI), various techniques have been proposed to make ML models more transparent. Among them, *counterfactual explanations* [195] gained substantial attention as intuitive and user-aligned post-hoc interpretability tools. The core idea behind counterfactual reasoning is to identify a minimally perturbed version of the input that would have led the model to produce a different output. In doing so, counterfactuals provide insight into a model's decision boundaries by answering *"what if"* questions in a concrete and actionable form. They have been applied to a wide array of model architectures, ranging from ensembles of decision trees [185, 188] and multi-layer perceptrons [140] to large-scale transformers [35], as well as through model-agnostic methods [41, 42, 145].

Despite this progress, a key limitation remains: most counterfactual explanations are expressed in abstract feature space (e.g., "*feature X changes from 0.3 to 0.7*"), which can be difficult for non-technical users to interpret or act upon. This challenge has sparked recent interest in generating natural language counterfactuals, where explanations are automatically translated into textual narratives that align more closely with human understanding. For instance, Fredes et al. [67] explored the use of Large Language Models (LLMs) to verbalize counterfactuals for tabular data, offering a promising first step toward more accessible forms of explanation.

In this work, we extend this line of inquiry to the more complex setting of graph-structured data. This kind of data is ubiquitous across a variety of knowledge-driven domains, including financial services (e.g., fraud detection and credit scoring) [30, 197], biomedical research (e.g., protein interaction networks) [64], and natural language processing (e.g., structured relation extraction) [29]. Graph Neural Networks (GNNs) have emerged as a dominant paradigm for learning from such data, achieving state-of-the-art results in tasks like node classification, graph classification, and link prediction. However, the structural complexity of graphs and their interdependencies make the generation and interpretation of graph counterfactuals significantly more challenging compared to simpler data modalities, such as images or tabular records. This structural complexity poses unique

challenges, necessitating dedicated strategies to translate graph-based counterfactuals into intuitive and easily interpretable natural language explanations.

In this paper, we tackle the problem of translating graph counterfactual explanations into natural language narratives using LLMs. Specifically, we consider counterfactual examples produced by graph-based explainers for both node and graph-level classification tasks. We then prompt LLMs to transform these structured counterfactuals into coherent, user-friendly textual narratives. This two-stage process bridges the gap between algorithmic explainability and human interpretability, making counterfactual reasoning more accessible to the end user.

To assess the quality of the generated explanations, we propose a set of novel evaluation metrics that quantify how accurately the natural language outputs reflect the underlying counterfactual examples. Given the known limitations of automatic metrics for evaluating open-ended text generation [45], we further validate our evaluation using human judgment.

We conduct extensive experiments using three graph counterfactual explainers: two for node classification, namely, CF-GNNExplainer [128] and its node-feature extension CF-GNNFeatures [74], and one for graph classification, GNNExplainer [218]. These experiments span multiple graph datasets and open-source LLMs, and demonstrate that our approach can effectively generate natural language explanations, supporting decision-making processes in critical, knowledge-driven domains.

3.4.1 Related Work

Recent advances in narrative-driven Explainable AI (XAI), particularly those leveraging Large Language Models (LLMs), have significantly improved the interpretability of complex machine learning models across various data modalities. A key area of interest within XAI is counterfactual explainability, which highlights how small changes to input variables can result in different model outputs. While counterfactuals are intuitively appealing, their technical complexity often makes them difficult to understand, especially for non-expert users, limiting their applicability in real-world, human-centered settings where interpretability and transparency are paramount.

The emergence of powerful LLMs, such as GPT-4 [4], Qwen2.5 [184], LLaMA [189], Mistral [106], and their successors [9], has opened new opportunities to bridge this gap. These models are capable of generating coherent, human-like language from complex inputs, making them ideal tools for translating technical content—including counterfactual explanations—into narratives that are accessible to general audiences. Their integration into XAI pipelines promises not only better user comprehension but also increased trust in AI-driven decision-making.

A notable contribution in this direction is the work by Fredes and Vitrià [67], which marks one of the first attempts to use LLMs to generate natural language explanations from sets of counterfactual examples. Their method—applied to tabular data—leverages GPT-4 to extract causal insights and suggest actionable changes that could alter classification outcomes. Similarly, Martens et al. [137] introduce XAIstories, a method that leverages LLMs to provide narrative explanations for both SHAP values and counterfactuals in tabular and image data. Although their results are compelling, the scope of their study is limited to a single, relatively simple data set (Adult Income), and it relies on a proprietary model, raising questions about reproducibility and generalizability.

Several recent studies illustrate the growing interest in natural language explanations for graph-based models. Giorgi et al. [74] generate counterfactual explanations for GNNs and utilize open-source LLMs, such as Qwen2.5, to produce human-readable outputs, demonstrating effectiveness

on datasets including Cora and CiteSeer. Cedro and Martens [34] introduce GraphXAIN, a model-agnostic approach that translates explanatory subgraphs into natural language narratives, showing high user satisfaction in surveys with researchers and practitioners. Extending these ideas to text-attributed graphs, Pan et al. [154] propose TAGExplainer, which utilizes pseudo-labels and iterative training to generate concise and faithful explanations, targeting applications such as social networks and recommendation systems.

Outside the graph domain, other notable contributions include Slack et al.'s TalkToModel [173], an interactive system that enables users, especially in healthcare settings, to engage with model explanations through dialogue, outperforming traditional dashboard-based interfaces. Complementing these efforts, the work of Jacovi et al. [104] provides a foundational discussion on the definition and evaluation of faithfulness in interpretable NLP systems, arguing for a graded rather than binary notion of this crucial property. More recently, Ichmoukhamedov et al. [102] propose quantitative metrics—such as faithfulness and human similarity—to evaluate LLM-generated explanations, offering much-needed rigor to the evaluation of narrative XAI.

Finally, although slightly orthogonal to our setting, He et al. [91] design an LLM-guided counterfactual generator to explain GNNs for molecular property prediction, addressing issues such as hallucination and domain knowledge gaps. The work by Zytek et al. [224] further underscores the potential of LLMs in XAI, outlining future research directions for "explaining explanations" and improving their interpretability.

Taken together, these works underscore a growing momentum toward making AI explanations more user-centric and interpretable through natural language. However, none of the existing approaches tackle the translation of *already generated* graph counterfactual explanations into natural language narratives using LLMs. Our work aims to fill this gap by introducing a novel method that combines graph-level counterfactual reasoning with accessible language generation, supported by both automatic and human-centered evaluations.

3.4.2 From Counterfactual Examples to Natural Language Narratives

The counterfactual explanation problem in classification tasks can be formally defined as follows.

Definition 3.4.1 (Counterfactual Explanation Problem)**.** *Given an input instance x and a predictive model $f_{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta}$—hereafter referred to as the oracle—the objective is to identify a modified instance $x^* \neq x$ such that $f_{\boldsymbol{\theta}}(x^*) \neq f_{\boldsymbol{\theta}}(x)$. The instance x^* is referred to as a counterfactual example for x.*

Among the set of possible counterfactuals (assuming at least one exists), we posit the existence of a counterfactual generator g that, given the original instance x, returns a counterfactual x^* minimizing the distance $d(x, x^*)$, or returns $\perp$ if no valid counterfactual can be found. The distance function $d(\cdot, \cdot)$ serves to constrain the generated counterfactual such that it remains as close as possible to the original instance, thereby ensuring minimal perturbation.

In this work, we focus specifically on inputs represented as graphs. That is, each instance x and its corresponding counterfactual x^* are represented as graphs $G = (V, E)$ and $G' = (V', E')$, respectively. A counterfactual example for an input graph G thus corresponds to a modified graph G' in which changes may affect the node features, the graph structure (i.e., the edge set), or both. These modifications must still satisfy the counterfactual condition, that is, they must induce a

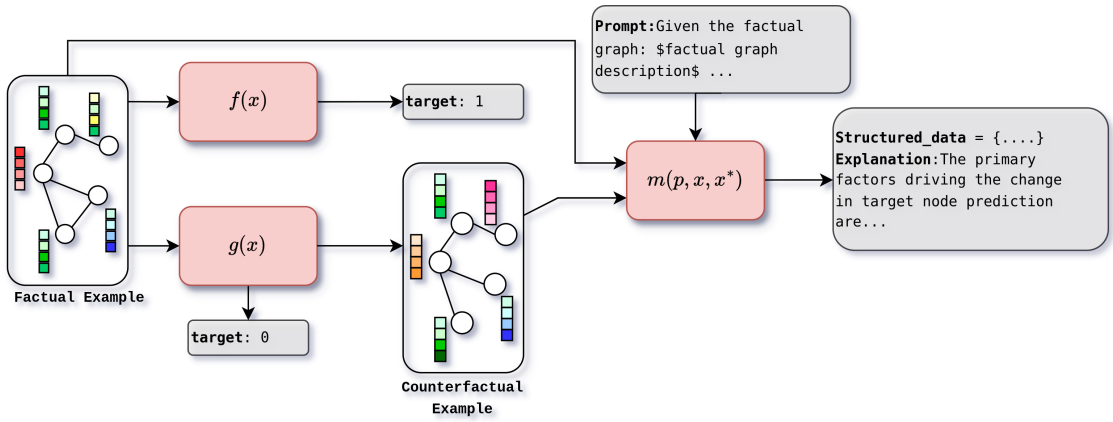


Figure 3.6: An illustration of our two-stage process for counterfactual narrative generation. In the first stage, given a factual graph x and an oracle model f, a counterfactual explainer g generates a modified graph x^* that flips the original prediction (from target 1 to 0). In the second stage, the factual-counterfactual pair (x, x^*) is encoded into a prompt and fed to a Large Language Model (LLM) m, which produces a natural language explanation (ε) and structured data summarizing the key changes.

change in the oracle's prediction. The graph-based setting introduces additional complexity, as changes may alter not only local node-level properties but also the global topology and relational dependencies of the graph.

While generating a counterfactual example provides a concrete instance that alters the model's decision, this output remains within the domain of data structures, and unfortunately, the set of modifications done by the explainer may lack direct interpretability. To bridge the gap between technical data modification and a human-understandable explanation, we introduce a second, crucial step: generating a natural language narrative. This leads us to the formal definition of the counterfactual narrative generation problem, which builds directly upon the output of the counterfactual explanation process. The problem of generating narratives from factual-counterfactual samples has been defined in [76] as follows:

Definition 3.4.2 (The Counterfactual Narrative Generation Problem)**.** *Given a generic pair of factual-counterfactual samples* $(\boldsymbol{x}, \boldsymbol{x}^*)$ *where the counterfactual sample* $\boldsymbol{x}^*$ *is a solution of the Counterfactual Explanation Problem, a generic function* m*, and a prompt* p *we want to generate a natural language narrative* ε *associated with the generic factual-counterfactual pair* $(\boldsymbol{x}, \boldsymbol{x}^*)$*, namely,* $\varepsilon = m(p, \boldsymbol{x}, \boldsymbol{x}^*)$ *also known as counterfactual narrative.*

Where the function m is defined generically to accommodate different instantiations, including deterministic mappings, probabilistic generative models, or ensembles thereof.

Proposed Method

We assume access to an oracle $f_{\boldsymbol{\theta}}$, specifically, a graph neural network (GNN) trained for either node or graph classification tasks. For any instance x, we consider its predicted label $\hat{y}$, such that $f_{\boldsymbol{\theta}}(x) = \hat{y}$, as well as its corresponding counterfactual example $x^* = g(x)$, where g is a counterfactual explainer that returns an optimal counterfactual instance for x.

Our goal is to generate a narrative ε that conveys the rationale behind the counterfactual example x^*. To achieve this, we prompt a pre-trained Large Language Model (LLM) m with the factual instance x, its counterfactual x^*, and a tailored prompt p, resulting in $\varepsilon = m(p, x, x')$. An overview of this process is illustrated in Figure 3.6.

A central challenge of this approach lies in assessing the quality of the generated natural language narratives. In the following section, we introduce a novel evaluation framework specifically designed to address this issue and provide a rigorous assessment of the generated outputs.

A New Evaluation Framework

Given the novelty of the task at hand, to the best of our knowledge, no established quantitative evaluation metrics exist for assessing natural language narratives generated by LLMs in the context of graph-based counterfactuals. The predominant approach in the literature relies on human judgment, which, while valuable, can introduce variability and subjective bias. To address this limitation and enable a more systematic and replicable evaluation, we propose a set of novel, task-specific metrics that quantitatively assess the language model's ability to articulate the differences between factual and counterfactual graph instances in a coherent and informative manner.

We denote the factual graph as G and its corresponding counterfactual as G'. To compute our metrics effectively, we design prompts that instruct the language model to populate a structured dictionary (see Figure 3.7) capturing key information from both G and G'. The content of this dictionary is tailored to the specific classification task under consideration.

In the case of node classification, the dictionary includes the identity of the target node, its predicted class in both the factual and counterfactual graphs, the node's neighborhood in each graph, and the set of features associated with the node in both scenarios. For graph classification tasks, the dictionary records the predicted class of the entire graph in both G and G', along with the nodes and edges that differ between the two instances.

After generating the structured dictionary, we parse it to extract key information about the feature changes between the factual instance G and its counterfactual G'. The core assumption is that a model's ability to populate this structured format correctly reflects its underlying understanding of the narrative. Therefore, errors in the dictionary signal potential inaccuracies in the subsequent natural language narrative. This technique provides a reliable and automated method for assessing the factual accuracy of the generated text, quantifying how accurately the narratives align with the ground truth factual and counterfactual feature changes. Since our study addresses

Graph Classification Dictionary

```
{
"Factual_Target_Graph_Class": None ,
"CounterFactual_Target_Graph_Class": None ,
"Node_Differences": [],
"Edge_Differences": [] ,
"Natural_Language_Explanation": None
}
```

Node Classification Dictionary

```
{
Target_Node_ID:'',
Factual_Target_Node_Neighbors: [],
Factual_Target_Node_Features: [],
Counterfactual_Target_Node_Neighbors: [],
Counterfactual_Target_Node_Features: [],
Factual_Target_Node_Class: '',
Counterfactual_Target_Node_class: '',
Target_Node_Features_Changed_From_Factual_To_Counterfactual: [],
Natural_Language_Explanation: ''
}
```

Figure 3.7: Structured output dictionaries for narratives in graph classification (left) and node classification (right) tasks.

two distinct tasks, namely, node classification and graph classification, we developed two separate evaluation protocols, each tailored to the specific structure and requirements of the task. The following sections provide a detailed description of these two frameworks.

Node Classification To evaluate the ability of LLMs to understand and explain counterfactual reasoning in node classification tasks, we introduce a dedicated set of binary metrics. These metrics are designed to assess whether the LLM can accurately identify and describe the structural and feature-level transformations of a target node between the factual and counterfactual graph instances. Each metric captures a specific dimension of the LLM's understanding, and collectively, they provide a rigorous framework for quantifying the fidelity and informativeness of the generated explanations.

Target Node Identification (TNI) This metric evaluates the LLM's ability to accurately identify the target node within the graph. Formally, let $G = (V, E)$ be the factual graph, $t \in V$ the true target node, and $v \in V$ the node identified by the LLM. The TNI metric is defined as $\text{TNI}(v) = \mathbb{1}[v = t]$

Counterfactual Class Identification (CCI) This metric assesses the LLM's ability to correctly identify the target node's new class in the counterfactual graph. Given a factual graph G and a counterfactual graph G' such that their predictions differ, $f_{\boldsymbol{\theta}}(G) \neq f_{\boldsymbol{\theta}}(G')$, let $c' = f_{\boldsymbol{\theta}}(G')$ be the counterfactual class. Denoting the class identified by the LLM for the instance G' as c_{LLM}, the CCI metric is defined as $\text{CCI}(c_{LLM}) = \mathbb{1}\left[c_{LLM} = c'\right]$

Factual Target Node Features (FTNF) This metric measures the extent to which the LLM accurately captures the feature vector of the target node in the original graph. Let $t \in V$ be the target node in the factual graph $G = (V, E)$, and let $\mathbf{x}_t \in \mathbb{R}^d$ be its associated feature vector. Let $\mathbf{x}_{\text{LLM}}$ denote the vector predicted by the LLM. The FTNF metric is defined as $\text{FTNF}(\mathbf{x_{LLM}}) = \mathbb{1}[\mathbf{x_{LLM}} = \mathbf{x}_t]$

Counterfactual Target Node Features (CFTNF) Analogous to FTNF, this metric evaluates whether the LLM correctly identifies the feature vector of the target node in the counterfactual graph. Let $G' = (V', E')$ be the counterfactual graph, and let $\mathbf{x}'t \in \mathbb{R}^d$ represent the true features of the target node $t \in V'$. The LLM's predicted vector is denoted as $\mathbf{x}_{\text{LLM}}$. The CFTNF metric is defined as $\text{CFTNF}(\mathbf{x_{LLM}}) = \mathbb{1}[\mathbf{x_{LLM}} = \mathbf{x}'_t]$

Counterfactual Target Node Neighbors (CFTNN) This metric assesses the LLM's ability to correctly identify the set of neighboring nodes of the target node in the counterfactual graph. This is particularly relevant, as changes in the local neighborhood may influence the classification outcome. Let $G' = (V', E')$ be the counterfactual graph, and let $t' \in V'$ denote the target node. The true neighborhood of t' is given by $\mathcal{N}(t') = u \in V' \mid (t', u) \in E'$. Let $\mathcal{N}_{\text{LLM}}(t')$ be the set of neighbors predicted by the LLM. The CFTNN metric is defined as $\text{CFTNN}(\mathcal{N}_{\text{LLM}}(t')) = \mathbb{1}[\mathcal{N}_{\text{LLM}}(t') = \mathcal{N}(t')]$

These metrics collectively provide a comprehensive framework for evaluating the LLM's ability to interpret and articulate the underlying structural and attribute-level transformations that occur between the factual and counterfactual graph instances. By quantifying the model's performance along these multiple dimensions, we can more precisely evaluate its ability to generate accurate narratives.

Graph Classification To evaluate the effectiveness of LLMs in generating narratives for graph-level classification tasks, we extend our analysis to the *AIDS dataset*, a widely used benchmark for identifying chemical compounds in antiviral drug discovery. In this setting, we employ *GNNExplainer* to generate counterfactual examples at the graph level. Given the shift from node to graph-level reasoning, we introduce a new set of evaluation metrics designed to capture the LLM's ability to understand and describe structural and class-level transformations.

Our framework introduces four evaluation metrics: *Factual Target Class (FTC)*, *Counterfactual Target Class (CTC)*, *Node Difference (NDIFF)*, and *Edge Difference (EDIFF)*. These metrics measure whether the LLM can accurately identify the predicted class labels and detect structural changes between the factual and counterfactual graphs. Formally, let the dataset consist of N graph instances. Each original (factual) graph is denoted as $G_i = (V_i, E_i, X_i)$, where V_i is the set of nodes, E_i the set of edges, and X_i the node feature matrix. The corresponding counterfactual graph, generated by GNNExplainer, is represented as $G_i' = (V_i', E_i', X_i')$. The LLM is prompted to predict class labels and structural differences, which are then evaluated against the ground truth.

Factual Target Class (FTC) This metric assesses whether the LLM accurately identifies the ground-truth class of the factual graph G_i. Let $\hat{y}_i$ denote the LLM's predicted class and y_i the true class label. FTC is defined as $FTC_i = \mathbb{1}[\hat{y}_i = y_i]$.

Counterfactual Target Class (CTC) This metric assesses whether the LLM correctly predicts the target class of the counterfactual graph G_i'. Let $\hat{y}_i'$ be the class predicted by the LLM and y_i' the ground truth counterfactual class. CTC is defined as $CTC_i = \mathbb{1}[\hat{y}_i' = y_i']$

Nodes Differences (NDIFF) This metric measures the LLM's ability to identify differences in node features between the factual and counterfactual graphs. Let $\Delta X_i = X_i' - X_i$ denote the true node-level difference, and $\Delta\hat{X}_i$ the difference as predicted by the LLM. NDIFF is defined as:

$$NDIFF_i = \begin{cases} \frac{|\Delta X_i \cap \Delta\hat{X}_i|}{|\Delta X_i|} & \text{if } |\Delta X_i| > 0 \\ 1 & \text{if } |\Delta X_i| = 0 \text{ and } |\Delta\hat{X}_i| = 0 \\ 0 & \text{Otherwise} \end{cases}$$

Edges Differences (EDIFF) This metric evaluates the LLM's ability to accurately detect changes in the graph structure. Let $\Delta E_i = E_i' - E_i$ denote the true edge-level difference, where $E_i \in G$ and $E_i' \in G'$ are respectively the factual and counterfactual edges, and $\Delta\hat{E}_i$ the difference as predicted by the LLM. EDIFF is defined as:

$$\text{EDIFF}_i = \begin{cases} \frac{|\Delta E_i \cap \Delta\hat{E}_i'|}{|\Delta E_i|} & \text{if } |\Delta E_i| > 0 \\ 1 & \text{if } |\Delta E_i| = 0 \text{ and } |\Delta\hat{E}_i'| = 0 \\ 0 & \text{Otherwise} \end{cases}$$

Aggregation of Metric Values

To obtain a robust measure of overall performance, we first computed each metric for every instance in the dataset and subsequently averaged these scores. To account for variability arising

from stochastic elements in the model or data processing, the evaluation has been repeated across distinct random seeds. The final performance is reported as the mean and standard deviation of the aggregated scores over these runs. This rigorous aggregation protocol enables a quantitative assessment of the LLM's understanding of graph-level and node-level transformations, thereby establishing a reliable benchmark for the quality of explanations.

Human Evaluation Procedure

To evaluate the quality of the generated counterfactual narratives, we employed a questionnaire-based protocol that has been previously used in [73]. The evaluation compared narratives from three distinct sources: the top-performing open-source model, the leading closed-source model, and a simple fixed-template baseline.

The study began with a demographic section to profile the participants. Data was collected on participant age, followed by their self-assessed expertise and professional profile. Expertise level was determined using the question: *"Which is your expertise level on the topic of molecular biology, medicine, and physiology? (1) Weak: No knowledge or limited knowledge about the topics. (2) Medium: Robust knowledge about the topics. (3) Strong: Very detailed knowledge about the topics at hand."*.

Participant profiles were categorized using the question: *"Select the option that best fits your profile: (1) Student (Non-expert): You are currently enrolled in a university course, but these topics are not related to your current study plan, or you are currently enrolled in a related course, but you still have not acquired knowledge about the topics at hand. (2) Student (Expert): You are currently enrolled in a related university course, and you already studied related subjects deeply, or you are enrolled in a non-related course, but you studied related subjects deeply before. (3) Professional (Non-expert): You are not a student, and you have never studied or worked in related subjects. (4) Professional (Expert): You are not a student, and you are currently involved professionally in related subjects, or you have been involved in them in the past as part of your professional activity."*

Upon completion of the demographic survey, the evaluation phase commenced. In this phase, each participant was tasked with assessing three narratives. These narratives were randomly sampled from a larger pool under the constraint that each set of three contained exactly one narrative from each of the three models under study. To mitigate bias, the presentation was double-blinded: the model source for each narrative was anonymized, and the order of the narratives was randomized for each participant. Participants were instructed to read all three narratives before proceeding to the evaluation instrument. The questionnaire then prompted participants to rate each narrative on a 5-point Likert scale across the following five dimensions:

- **Q1 (Terminology):** Is the terminology and language used in the explanation appropriate and easy to understand?
- **Q2 (Clarity):** How clear and easy to understand is the provided counterfactual explanation?
- **Q3 (Structural Description):** How clearly does the explanation describe the changes in node connections (graph structure) that led to the counterfactual outcome?
- **Q4 (Interpretability):** Are the changes in features and structure easy to interpret and make sense in the context of the original graph?

- **Q5 (Overall Assessment):** What is your overall assessment of the clarity and coherence of the counterfactual explanation?

The collected responses provide qualitative insights into the perceived clarity, interpretability, and overall effectiveness of the explanations generated by different models.

3.4.3 Experiments

Given the general framework described in Section 3.4.2, this section presents the experimental setup and evaluation of our approach for generating narratives using LLMs. To comprehensively assess the effectiveness and generalizability of our method, we conduct experiments on two distinct tasks: node classification and graph classification. In the node classification setting, the goal is to generate counterfactual explanations that describe changes necessary to alter the classification of an individual node within a larger graph. In contrast, in the graph classification task, counterfactuals are generated at the graph level, identifying modifications that would change the overall label assigned to the entire graph. To ensure a robust evaluation, we employ real-world datasets widely used in graph-based machine learning.

Counterfactual Explainers

As outlined in Section 3.4.2, our proposed pipeline requires a counterfactual explainer g but is otherwise agnostic to its specific choice. To rigorously evaluate the robustness of our framework, we tested it with three state-of-the-art graph explainers that use distinct modification strategies. This comparative approach enables us to systematically assess how the nature of the input counterfactuals—whether structural or feature-based—influences the quality and coherence of the generated natural language explanations.

CF-GNNExplainer is a perturbation-based counterfactual explainer. It defines $\bar{\mathbf{A}}_v = \mathbf{P} \odot \mathbf{A}_v$, where $\mathbf{P}$ is a binary perturbation matrix that sparsifies $\mathbf{A}_v$. The goal is to find $\mathbf{P}$ for a given node v such that $f_{\boldsymbol{\theta}}(\mathbf{A}_v, x) \neq f_{\boldsymbol{\theta}}(\mathbf{P} \odot \mathbf{A}_v, x)$. To find $\mathbf{P}$, CF-GNNExplainer exploits a technique to train sparse neural networks to zero out entries in the adjacency matrix (i.e., removing edges). This results in the deletion of the edge between node i and node j. The hyperparameters selected for this explainer are: $\alpha = 0.01$, $\beta = 0.5$, $k = 500$ (iterations), and as optimizer Adam.

CF-GNNFeatures is a node features perturbation-based counterfactual explainer. Given a graph $G(V, E)$, two matrices are defined, namely: $\mathbf{V}_x$, the node features matrix representing the features for every node in G, and $\mathbf{P}_x$, the feature perturbation matrix. Initially, the matrix $\mathbf{P}_x$ is filled with ones to maintain the original sets of attributes. Given an oracle $f_{\boldsymbol{\theta}}$, parameterized by θ, we fix all the weights and train $\mathbf{P}_x$ to change the attribute matrix $\mathbf{V}_x$ that is fed into the oracle, multiplying the current feature matrix with the perturbation matrix. The hyperparameters selected for this explainer are the same as CF-GNNExplainer: $\alpha = 0.01$, $\beta = 0.5$, $k = 500$ (iterations), and as optimized Adam. Here we also clipped the perturbation gradient norm to 3.0.

GNNExplainer is a model-specific explainer originally designed for generating post-hoc explanations for GNN predictions. While not inherently counterfactual, it can be adapted to generate counterfactual instances by identifying the minimal subgraph and feature set most influential to the model's decision, and then altering them to induce a change in prediction. In our study, we leverage GNNExplainer for graph classification tasks. Given a graph $G = (V, E, X)$ and a

trained GNN model $f_{\boldsymbol{\theta}}$, GNNExplainer learns a mask over edges and node features to uncover the substructures most relevant for the predicted class. To generate counterfactuals, we invert the learned mask, removing or modifying the most influential components, such that the perturbed graph $G' = (V', E', X')$ results in a different class prediction $f_{\boldsymbol{\theta}}(G') \neq f_{\boldsymbol{\theta}}(G)$. Since we used the `torch_geometric` implementation we selected just the learning rate as $\alpha = 0.05$.

Large Language Models

In this study, we evaluated a suite of state-of-the-art open-source LLMs to generate natural language explanations for counterfactual graph instances. Our selection encompasses models from various architectural families and parameter scales, providing a broad perspective on the impact of model size on performance.

The evaluated models are: `Qwen2.5` [184], `Mistral-7B-Instruct-v0.3` [106], `Llama-3.2-3B-Instruct` and `Llama-3.1-8B-Instruct` [189], `DeepSeek-R1` [84], and `Phi-3.5-mini-Instruct` [3], and Grok4 [211]. These models vary not only in scale, from compact, open-source models (0.5B up to 14B) up to a larger, closed-source model like Grok4 (estimated to have up to 1700B [182, 183]), but also in architecture and pretraining strategies, allowing us to comprehensively benchmark their ability to generate accurate and user-friendly explanations. To ensure reproducibility, we applied a controlled set of hyperparameters for each model. Table 3.13 summarizes the specific configurations used in our experiments. All the open source models have been quantized using `GPTQ` [66] with 4-bit integer precision (`int4`) to minimize memory usage and enable efficient inference.

Table 3.13: Hyperparameter configurations used for each language model. The columns represent: **Model** – the name and size of the LLM; **Temp.** – sampling temperature; **Top-p** – nucleus sampling threshold; **Top-k** – the number of highest-probability tokens considered; **Rep. Pen.** – repetition penalty; and **Max Out. T.** – the maximum number of output tokens.

Model	Temp.	Top-p	Top-k	Rep. Pen.	Max Out. T.
DeepSeek Family	0.1	0.90	40	1.10	2048
Grok-4-1700B	n.a.	n.a.	n.a.	n.a.	2048
Llama-3.1 Family	0.1	0.85	40	1.05	2048
Llama-3.2-3B	0.2	0.80	50	1.00	2048
Mistral-v0.3-7B	0.1	0.90	40	1.10	2048
Phi-3.5-3B	0.2	0.85	50	1.10	2048
Qwen2.5 Family	0.1	0.80	10	1.05	2048

Prompting Methodology The performance of LLMs is highly sensitive to the structure and content of the input prompt. Therefore, we developed a two-stage prompting strategy to guide the models in generating high-quality counterfactual explanations. Our methodology consists of two distinct components: (1) a global system prompt and (2) an instance-specific prompt.

First, a comprehensive system prompt is provided to the LLM, as shown in Figure 3.9. This initial prompt establishes the model's persona as an expert analyst in Explainable AI (XAI) for Graph Neural Networks. It equips the model with essential background knowledge, defining counterfactual explanations in the context of graph data and providing a detailed description of the GNN explainer method used. Furthermore, it specifies the required output format, including a structured Python dictionary schema that the model must populate with its findings.

Second, for each explanation to be generated, an instance-specific prompt is supplied (Figure 3.8). This prompt contains the concrete data for a single case. To effectively serialize the factual and counterfactual graphs for the model, we adapted the incident representation framework proposed by Fatemi et al. [62]. Crucially, the prompt also provides a detailed set of instructions that guide the LLM through the analytical process. These instructions direct the model to identify changes between the graphs, translate abstract structural and feature modifications into meaningful domain-specific concepts, and weave these insights into a coherent, causal narrative, before presenting the final structured data. This dual-prompt structure ensures that the LLM has both the general context and the specific guidance necessary to perform the complex task of generating narratives.

Prompt

Your Turn
Dataset Knowledge
The dataset used is the following: The AIDS dataset is a collection of molecular graphs used for the graph classification task. The goal is to predict whether a given molecular compound is active or inactive against HIV. Each graph in the dataset represents a single molecule. Graph Structure Nodes: Represent individual atoms within a molecule. Edges: Represent the chemical bonds between atoms. Node Attributes Each atom (node) has features describing its chemical properties and spatial location: Atom Symbol: The chemical symbol (e.g., C, O, N) is encoded as an integer label. Chemical Properties: A feature vector capturing specific chemical characteristics. Charge: The electrical charge of the atom. Coordinates: x and y spatial coordinates define the atom's position. Node Labels The nodes are labeled with atom symbols, mapped to integers: 0: C (Carbon) * 1: O (Oxygen) * 2: N (Nitrogen) * 3: Cl (Chlorine) * 4: F (Fluorine) * 5: S (Sulfur) * 6: Se (Selenium) * 7: P (Phosphorus) * 8: Na (Sodium) * 9: I (Iodine) * 10: Co (Cobalt) * 11: Br (Bromine) * 12: Li (Lithium) * 13: Si (Silicon) * 14: Mg (Magnesium) * 15: Cu (Copper) * 16: As (Arsenic) * 17: B (Boron) * 18: Pt (Platinum) * 19: Ru (Ruthenium) * 20: K (Potassium) * 21: Pd (Palladium) * 22: Au (Gold) * 23: Te (Tellurium) * 24: W (Tungsten) * 25: Zn (Zinc) * 26: Bi (Bismuth) * 27: Pb (Lead) * 28: Ge (Germanium) * 29: Sb (Antimony) * 30: Sn (Tin) * 31: Ga (Gallium) * 32: Hg (Mercury) * 33: Ho (Holmium) * 34: Cr (Chromium) * 35: Ni (Nickel) * 36: Mn (Manganese) * 37: V (Vanadium) Edge Attributes Each bond (edge) is labeled according to its valence to distinguish between different types of chemical bonds: 0: Single bond * 1: Double bond * 2: Triple bond Class Labels The molecules are classified into two categories: 1: Active against HIV (a) * 0: Inactive against HIV (i)
Original Graph This graph is classified as class i. The graph contains the following nodes: The node 0 has as attributes chem = 4.0, charge = 0.0, x = 3.76, y = -1.23 and is labeled as N. It is connected with the nodes 1, 4, 7. The node 1 has as attributes chem = 1.0, charge = 0.0, x = 2.95, y = -1.81 and is labeled as C. It is connected with the nodes 0, 2, 6. The node 2 has as attributes chem = 1.0, charge = 0.0, x = 3.26, y = -2.76 and is labeled as C. It is connected with the nodes 1, 3. The node 3 has as attributes chem = 1.0, charge = 0.0, x = 4.26, y = -2.76 and is labeled as C. It is connected with the nodes 2, 4. The node 4 has as attributes chem = 1.0, charge = 0.0, x = 4.56, y = -1.81 and is labeled as C. It is connected with the nodes 0, 3, 5. The node 5 has as attributes chem = 2.0, charge = 0.0, x = 5.52, y = -1.5 and is labeled as O. It is connected with the nodes 4. The node 6 has as attributes chem = 2.0, charge = 0.0, x = 2.0, y = -1.5 and is labeled as O. It is connected with the nodes 1. The node 7 has as attributes chem = 1.0, charge = 0.0, x = 3.76, y = -0.23 and is labeled as C. It is connected with the nodes 0, 8. The node 8 has as attributes chem = 1.0, charge = 0.0, x = 4.62, y = 0.26 and is labeled as C. It is connected with the nodes 9, 7. The node 9 has as attributes chem = 1.0, charge = 0.0, x = 4.62, y = 1.26 and is labeled as C. It is connected with the nodes 8, 10. The node 10 has as attributes chem = 1.0, charge = 0.0, x = 5.49, y = 1.76 and is labeled as C. It is connected with the nodes 9, 11. The node 11 has as attributes chem = 1.0, charge = 0.0, x = 5.49, y = 2.76 and is labeled as C. It is connected with the nodes 10
Counterfactual Graph This graph is classified as class i. The graph contains the following nodes: The node 0 has as attributes chem = 4.0, charge = 0.0, x = -1.24, y = -1.23 and is labeled as N. It is connected with the nodes 1, 4, 7. The node 1 has as attributes chem = 1.0, charge = 0.0, x = -0.33, y = -1.81 and is labeled as C. It is connected with the nodes 0, 6. The node 2 has as attributes chem = 1.0, charge = 0.0, x = 3.26, y = 2.61 and is labeled as C. It is connected with the nodes 3. The node 3 has as attributes chem = 1.0, charge = 0.0, x = 4.26, y = 3.57 and is labeled as C. It is connected with the nodes 2, 4. The node 4 has as attributes chem = 1.0, charge = 0.0, x = 4.56, y = 2.21 and is labeled as C. It is connected with the nodes 0, 3, 5. The node 5 has as attributes chem = 2.0, charge = 0.0, x = -3.64, y = -1.5 and is labeled as O. It is connected with the nodes 4. The node 6 has as attributes chem = 2.0, charge = 0.0, x = -0.98, y = -1.5 and is labeled as O. It is connected with the nodes 1. The node 7 has as attributes chem = 1.0, charge = 0.0, x = -1.55, y = -0.23 and is labeled as C. It is connected with the nodes 8, 0. The node 8 has as attributes chem = 1.0, charge = 0.0, x = -1.7, y = 0.26 and is labeled as C. It is connected with the nodes 9, 7. The node 9 has as attributes chem = 1.0, charge = 0.0, x = -1.2, y = 1.26 and is labeled as C. It is connected with the nodes 8, 10. The node 10 has as attributes chem = 1.0, charge = 0.0, x = -0.36, y = 1.76 and is labeled as C. It is connected with the nodes 9, 11. The node 11 has as attributes chem = 1.0, charge = 0.0, x = -0.75, y = 2.76 and is labeled as C. It is connected with the nodes 10.
Instructions Identify and State the Core Facts. Pinpoint the Classification Change: Clearly state the initial ("factual") class and the final ("counterfactual") class. For the AIDS dataset, this would be from inactive (0) to active (1), or vice versa. Provide a High-Level, Non-Technical Summary. Synthesize the findings from step 1 into a concise, easy-to-understand paragraph. This summary should immediately tell the user what happened without diving into technical jargon. For example: "The molecule was reclassified from inactive to active by changing the chemical properties of four specific atoms without altering its physical structure." Translate Graph Changes into Chemical Concepts. This is the most critical step. The model must map the abstract graph data to real-world chemistry. For Node Feature Changes: Identify the atom involved (e.g., Oxygen, Nitrogen, Sulfur). Explain the chemical role of that type of atom (e.g., "Oxygen and Nitrogen atoms are crucial as they often form hydrogen bonds, which act like molecular glue to anchor a drug to a protein"). Interpret the change in the chem attribute as a modification of the atom's electronic properties, such as its ability to act as a hydrogen bond donor or acceptor. For Edge Additions/Removals: State that edges represent chemical bonds which define the molecule's 3D shape, or conformation. Explain the consequence of breaking or forming a bond: it fundamentally alters the molecule's shape and flexibility. Use the "key and lock" analogy: Explain that the molecule's shape (the key) must precisely fit the protein's active site (the lock). Changing the bond structure "breaks" or "fixes" the key. Construct a Coherent Causal Narrative Weave the technical points from step 3 into a clear story. Connect the dots for the user. Start with the specific graph change, link it to the chemical consequence, and conclude with the impact on biological activity. Present the Structured Data Conclude by providing the clean, machine-readable dictionary that was requested. This allows for easy programmatic use of the findings and serves as a final, unambiguous summary of the exact changes.

Figure 3.8: An example of the instance-specific prompt. The factual and counterfactual graphs are encoded using the framework proposed by Fatemi et al. [62], and an explanation is generated for a sample instance using a GNNExplainer.

Baseline

To establish a clear and simple benchmark for comparison, we developed a template-based baseline. This system operates without a generative language model and functions by programmatically inserting the list of structural and feature modifications, as identified by the explainer, into a fixed sentence structure. The resulting output is deterministic and purely descriptive, serving as a reference to evaluate the improvements in narrative coherence, readability, and contextual depth

SystemPrompt

Instructions
You are an expert analyst specializing in Explainable AI (XAI) for Graph Neural Networks. Your task is to analyze two graphs: a 'Factual Graph' and a 'Counterfactual Graph'.
1. Analyze the differences: Identify the specific changes (node or edge modifications) that were made to the factual graph to create the counterfactual one.
2. Provide a summary: In a brief paragraph, explain what changed and how it relates to the concept of a counterfactual explanation.
3. Fill the dictionary: Populate the provided Python dictionary schema with the specific, structured information about the changes.
4. Avoid schematic or list formats: Your summary should be in natural language, avoiding bullet points, numbered lists or emoji.
Knowledge
Counterfactual Explanation
Counterfactual Explanation: A counterfactual explanation refers to a type of explanation in machine learning and artificial intelligence that describes how altering certain input features can change the output of a model. It answers 'what if' scenarios by identifying minimal changes necessary to achieve a different desired outcome. Counterfactual explanations provide insights into the decision-making process of complex models, enhancing transparency and interpretability. For example, consider a credit scoring model that denies a loan application. A counterfactual explanation might be: 'If your annual income had been $50,000 instead of $45,000, your loan would have been approved.' This helps the applicant understand what specific change could lead to a different decision.
Counterfactual Explanations in Graphs
Graphs are data structures consisting of nodes (vertices) and edges that represent relationships between entities. Graphs are prevalent in various domains such as social networks, biological networks, and knowledge graphs. When machine learning models, particularly Graph Neural Networks (GNNs), are applied to graph data, interpreting their predictions becomes challenging due to the complexity of the graph structure. Counterfactual explanations in graphs aim to provide insights into how changes in the graph's structure or node features can alter the model's predictions. This involves identifying minimal modifications to the graph that would change the outcome for a specific node or the entire graph.
Counterfactual Explainer
Explainer Description: GNNExplainer
Core Idea: GNNExplainer is a model-agnostic explanation method designed specifically for Graph Neural Networks (GNNs). Its primary goal is to explain *any* GNN's prediction for a specific instance (a node, a graph, or a link) by identifying the most influential components of the graph that led to that prediction. It answers the question: "Which parts of the graph and its features were most important for this specific outcome?"
How It Works: The method works by learning a mask over the graph's structure and node features. It treats the explanation process as an optimization problem. The objective is to find a small subgraph and a minimal set of node features that maximize the mutual information with the GNN's original prediction.
In simpler terms, GNNExplainer searches for the smallest possible piece of the original graph that, if fed back into the GNN, would still produce the same (or a very similar) prediction. This is achieved through the following steps: 1. Identify a Computation Graph: For a given prediction (e.g., classifying a specific node), it first considers the k-hop neighborhood of that node, which is the part of the graph the GNN used to make its decision. 2. **Learn Masks**: It learns two types of "soft masks" (masks with continuous values, not just 0 or 1): * An **edge mask** to identify the most critical subgraph structure. * A **feature mask** to identify the most important input node features. 3. **Optimize**: These masks are optimized by maximizing the mutual information between the GNN's prediction and the masked (simplified) graph. The optimization process also includes terms that encourage the masks to be sparse, ensuring the final explanation is as small and concise as possible.
Key Characteristics: * **Model-Agnostic**: It can be applied to any GNN-based model (e.g., GCN, GAT, GraphSAGE) without needing to know its internal architecture. * **Instance-Level Explanations**: It provides local explanations for individual predictions, not global explanations for the model as a whole. * **Identifies Subgraph and Features**: The output is not just a set of important nodes. The explanation is a combination of a crucial **subgraph structure** (the most important connections) and the most salient **node features**. #### **Type of Explanation** * **Output**: The explanation is a subgraph of the original computation graph and a subset of node features. This is a form of **feature attribution**, where importance scores are assigned to edges and node features. * **Use Case**: It is used for post-hoc interpretability, helping users understand why a trained GNN made a specific decision for a node, link, or entire graph. This is useful for debugging models, verifying their logic, and discovering new patterns in the data.
Dictionary Schema
Here is the Python dictionary schema you should use to structure your findings:

```
infos={
        "Factual_Target_Graph_Class": <FACTUAL_CLASS_VALUE> , // Required integer
        "CounterFactual_Target_Graph_Class": <COUNTERFACTUAL_CLASS_VALUE> , // Required integer
        "Node_Differences": [<NODE_ID_1>, <NODE_ID_2>, ...] , // Required list of integers
        "Edge_Differences": [[<NODE_ID_A>, <NODE_ID_B>], [<NODE_ID_C>, <NODE_ID_D>], ...] , // Required list of lists of integers
    }
```

Figure 3.9: The system prompt provided to the model. It defines the model's persona, provides general background knowledge (e.g., on a GNN explainer), and specifies the output schema used in the task.

provided by the more sophisticated LLM-based approaches. See Figure 3.13 for a practical example on the AIDS dataset.

Experimental Setup

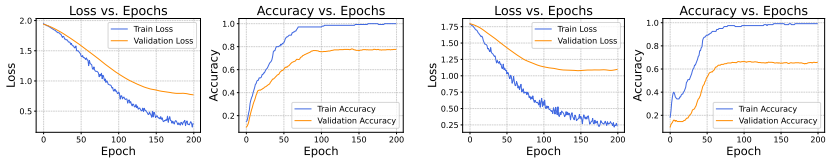
Our experimental framework utilized two distinct modalities for model access. Open-source models were evaluated on a local workstation equipped with an AMD Ryzen 9 7900 processor, 64GB of RAM, and a single NVIDIA RTX 4090 GPU. In contrast, the Grok-4 model was accessed via the xAI web service API. To ensure the robustness of our results, the narrative generation process was replicated three times for each model using different random seeds (42, 54, and 100). The findings presented in this paper are aggregated from these runs unless otherwise noted.

Oracles and Datasets Details To facilitate a comprehensive evaluation, our study utilizes three benchmark datasets for two distinct graph-based tasks: node classification and graph classification. For node classification, we employ the Cora and CiteSeer datasets. The Cora dataset comprises 2,708 scientific publications classified into seven categories, linked by 5,429 citation links. Similarly, the CiteSeer dataset comprises 3,312 publications across six classes with 4,732 links. In both datasets, each publication (node) is represented by a high-dimensional, sparse binary vector indicating the presence or absence of words from a vocabulary of 1,433 and 3,703 terms, respectively. To provide more informative features for the LLM, we reconstructed the original vocabularies to map these binary vectors back to human-readable tokens. For the graph classification task, we use

Table 3.14: Summary of model architecture and training hyperparameters for the implemented classification tasks.

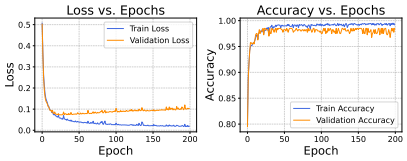
	Model Architecture					Training Hyperparameters						
Dataset	# Layers	Hidden	Activation	Readout	Dropout	Optimizer	LR	Weight Decay	Loss	Epochs	Batch Size	Split
GCN on Cora	3	16	ReLU	—	0.5	Adam	0.001	5e-4	Cross-Entropy	200	Full	Planetoid
GCN on CiteSeer	3	16	ReLU	—	0.5	Adam	0.001	5e-4	Cross-Entropy	200	Full	Planetoid
GCN on AIDS	3	64	ReLU	Mean Pool	0.5	Adam	0.0005	—	Cross-Entropy	200	64	80/20%

the AIDS dataset, which contains 2,000 molecular graphs. In this dataset, nodes represent atoms and edges correspond to chemical bonds. The objective is to classify each molecule as either "active" or "inactive" against HIV. The oracle models for all tasks are three-layer Graph Convolutional Networks (GCNs), trained for 200 epochs using the Adam optimizer. Detailed model architectures and training hyperparameters are summarized in Table 3.14. The training performance, including loss and accuracy curves for each oracle, is presented in Figure 3.10.



(a) Cora Oracle

(b) Citeseer Oracle



(c) AIDS Oracle

Figure 3.10: Train and test loss and accuracy values for all the datasets used

Graph Classification Results

In this section, we analyze the performance metrics for various LLM families evaluated on the AIDS dataset using graph neural network explanations. The results, visualized in Figure 3.11, highlight a clear trend of performance scaling with model size, which ranges from 0.5B to 1700B parameters. For the FTC metric, larger models generally exhibit higher values, indicating a positive correlation with model scale. For instance, the Grok-4 model achieves a score of 1, while Qwen2.5-14B scores around 0.8. In contrast, smaller models show markedly lower FTC: the 0.5B variant in the Qwen2.5 family and the 1.5B model in the DeepSeek family both score close to 0.0. The CTC metric follows a similar upward trend with model size. Large models, such as Grok-4 and both Qwen2.5-14B and DeepSeek-14B variants (respectively 0.8 and around 0.74), perform well. Conversely, smaller models decline significantly, with the smaller DeepSeek and Qwen2.5 variants scoring 0.05 and 0.01, respectively. NDIFF values are also generally higher for larger models. Examples such as Llama-3.2-3B and Phi-3.5-3B achieve scores of approximately 0.64 and 0.8. The largest model, Grok-4, achieves nearly 0.96, whereas smaller models, such as DeepSeek-1.5B, demonstrate substantially reduced scores, approaching 0.05 in this case as well. The EDIFF metric also shows a positive trend with model size, exhibiting the strongest correlation among all metrics (Pearson ρ: 0.90). While smaller models, such as the 0.5B variant of the Qwen2.5 family, score near zero, larger models achieve higher values, with Grok-4 peaking at approximately 0.38.

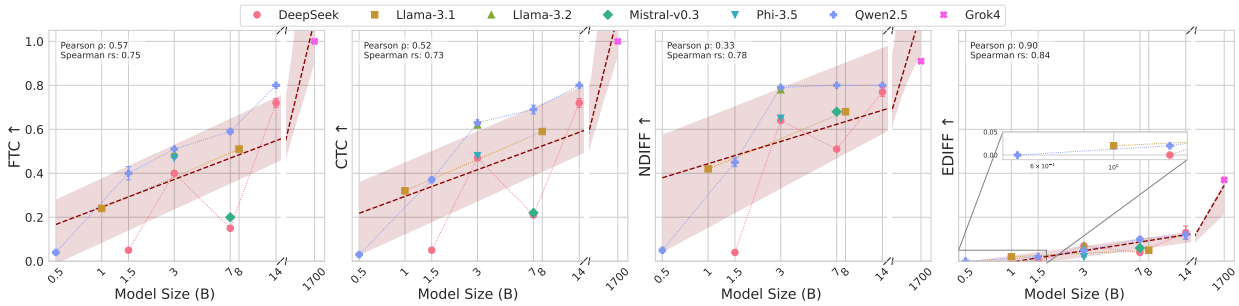


Figure 3.11: Results on the AIDS dataset using as explainer GNNExplainer. The plots have a broken x-axis using a log scale.

However, it is noteworthy that even this peak value remains relatively low in absolute terms. This suggests that while larger models are generally more adept at understanding graph changes, as indicated by the FTC, CTC, and NDIFF scores, identifying purely structural differences (i.e., edges) represents a more complex challenge for all models. This correlation with model size is discussed in detail in Section 3.4.3.

Human Evaluation To assess the quality of the generated explanations, we conducted a human evaluation on outputs from the simple baseline, deepseek-14B, and grok4. We gathered feedback from 23 participants who were asked to rate the explanations across five dimensions.

(a) Age Distribution **(b)** Expertise Level **(c)** Participant Profiles

Figure 3.12: Demographics of the 23 participants in the human evaluation study. Figure (a) shows the age distribution, (b) the self-reported expertise level, and (c) the distribution of participant profiles.

Details of the participant demographics are presented in Figure 3.12. The age distribution (Figure 3.12a) is skewed towards younger individuals, with a majority of participants aged between 18 and 28. The pie charts detail the participants' self-reported expertise and professional profiles. Results of the evaluation (Table 3.15) confirm that the baseline can be used to compose very simple narratives; however, the quality of them is judged poorly by the participants. The best open-source model we used instead (Qwen2.5-14B) received a better evaluation, but the closed-source model yielded the best results. In Figure 3.13, it is possible to see a narrative example for an element of the AIDS dataset.

Table 3.15: Average scores (1 to 5) from the human evaluation for explanations generated by different models. The comparison is shown for counterfactuals from GNNExplainer on the **AIDS** dataset (graph classification) and the **Cora** dataset (node classification).

	AIDS Dataset			Cora Dataset		
Question	**Baseline**	**Qwen2.5 - 14B**	**Grok-4**	**Baseline**	**Qwen2.5 - 14B**	**Grok-4**
Q1	1.0 ± 0.0	2.4 ± 0.5	4.3 ± 0.4	1,0 ± 0,0	2,0 ± 0,0	3,3 ± 0,5
Q2	1.2 ± 0.4	2.6 ± 0.4	4.5 ± 0.5	1,0 ± 0,0	2,1 ± 0,3	3,4 ± 0,5
Q3	1.0 ± 0.2	2.7 ± 0.4	4.6 ± 0.4	1,0 ± 0,0	2,3 ± 0,5	3,2 ± 0,4
Q4	1.0 ± 0.0	2.0 ± 0.2	4.4 ± 0.5	1,0 ± 0,0	2,0 ± 0,0	3,2 ± 0,4
Q5	1.0 ± 0.2	2.3 ± 0.4	4.3 ± 0.4	1,0 ± 0,0	1,8 ± 0,4	3,4 ± 0,5

Node Classification Results

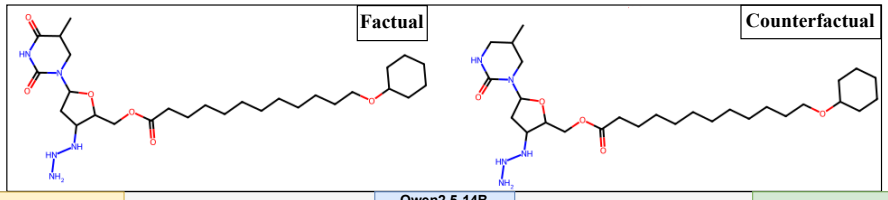
Our node classification experiments (Figure 3.14 and 3.15) demonstrate that the proposed method achieves high fidelity in capturing factual-counterfactual differences. Core explanation metrics, such as TNI and CCI, frequently surpassed scores of 0.8 even for smaller models (3B parameters), indicating strong performance in identifying the salient elements that differentiate the original and counterfactual predictions.

A critical finding emerged from the comparison between different counterfactual generation strategies. We observed that feature-oriented metrics, FTNF and CFTNF, performed better when the explainer (CF-GNNExplainer, top rows in both Figure 3.14 and 3.15) preserved the original node features. Conversely, performance on these metrics, particularly FTNF, declined when the explainer (CF-GNNFeatures, bottom rows in both Figure 3.14 and 3.15) modified node features. These results suggest a key insight: the evaluated LLMs are more proficient at verbalizing stable graph properties, those that remain unchanged between factual and counterfactual instances, than they are at articulating the specific modifications introduced by the explainer. The CFTNN metric followed a similar trend. Notably, the Grok4 model was an exception, showing resilience to these effects across datasets. A final observation is the remarkably low standard deviation (often near zero) in metric scores for most models, indicating high output consistency, with Qwen2.5 being the primary exception.

Human Evaluation To assess the quality of the generated explanations, we conducted a human evaluation on outputs from the simple baseline, DeepSeek-14B, and Grok-4. We gathered feedback from 29 participants who were asked to rate the explanations across five dimensions. Details of the participant demographics are presented in Figure 3.16. The age distribution (Figure 3.16a) is skewed towards younger individuals (age 18-28). The pie charts detail the participants' self-reported expertise and professional profiles. Notably, there is an apparent discrepancy between the two. While Figure 3.16b indicates that a large majority (58.6%) of participants considered their knowledge to be 'Weak', the profile breakdown in Figure 3.16c shows that the majority, 62.9%, fall into 'Non-Expert' categories, broken down as Student (44.4%) and Professional (18.5%). Results of the evaluation are reported in Table 3.15.

Impact of LLM Scale on Narratives

To investigate the influence of the LLM scale on the quantitative metrics defined in Section 3.4.2, we analyzed correlation values, namely, Spearman and Pearson. Across all configurations, a positive



Grok-4 Response

Core Facts and Classification Change
The factual graph represents a molecule classified as active (class 1) against HIV in the AIDS dataset. The counterfactual graph modifies this molecule, resulting in reclassification to inactive (class 0). This shift implies that the GNN model now predicts no inhibitory activity, likely due to reduced binding affinity to HIV proteins such as reverse transcriptase. No explicit confidence or probability values are provided in the graph data, but the binary flip from 1 to 0 suggests a decisive threshold crossing, potentially corresponding to a drop in predicted binding free energy (ΔG) of 2-5 kcal/mol, based on typical QSAR models for HIV inhibitors where small structural tweaks can toggle activity.
High-Level, Non-Technical Summary
The original active molecule against HIV becomes inactive through two subtle tweaks: one carbon atom in its core structure has its chemical properties adjusted, making it less reactive, and a key bond linking another carbon to an oxygen atom is severed, isolating that oxygen like a loose thread. These changes reshape the molecule just enough to prevent it from latching onto the virus's proteins effectively, turning a potential drug candidate into a dud—much like slightly bending a key so it no longer fits the lock.
Node Feature Changes
The only node with a modified feature is node 7, a carbon (C) atom. In the factual graph, its "chem" attribute is 1.0, which likely encodes a standard sp3-hybridized carbon with tetrahedral geometry, typical for aliphatic chains in organic molecules (Pauling electronegativity ~2.55, partial charge near 0, and four valence electrons available for bonding, often with implicit hydrogens to satisfy valence 4). This configuration allows it to contribute to flexible, hydrophobic regions that stabilize drug-protein interactions via van der Waals forces. In the counterfactual graph, the "chem" drops to 0.0, suggesting a shift to a less saturated or deactivated state—possibly mimicking deprotonation, radical formation, or a change to sp2 hybridization (e.g., like in an alkene or carbonyl precursor), reducing its electron density (partial charge shift by ~0.1-0.2 e) and hydrogen-bonding potential. Carbon atoms like this are backbone builders in drug molecules; altering their electronic properties diminishes their role in pi-stacking or sigma-bond donations, which are vital for HIV inhibitors that nestle into enzyme pockets. For node 7, connected to nodes 6 (C), 8 (C), and 13 (C), this tweak could raise its local polarity, increasing solvation entropy by ~5-10 J/mol·K and disrupting hydrophobic collapse needed for active-site binding. No other node features differ meaningfully; charges remain unchanged across all nodes (e.g., node 15 N at +1.0, node 16 N at -1.0, indicating a charged ammonium pair preserved for ionic interactions), and atom types (labels) are identical. The widespread shifts in x/y coordinates reflect a global conformational rearrangement—likely a rotation or flip in 2D projection—but do not alter intrinsic node identities, akin to the molecule adopting a higher-energy conformer with torsional strain increased by 1-3 kcal/mol, per rotational barriers in C-C bonds (~12 kcal/mol for ethane-like).
Edge Changes (Bond Modifications)
The sole edge difference is the removal of the bond between node 8 (C) and node 11 (O) in the counterfactual graph. In the factual graph, this is a C-O single bond (edge [8,11]), with node 11 as a terminal oxygen (degree 1, chem=2.0 indicating oxygen's high electronegativity ~3.44 and dual lone pairs for hydrogen bonding). Such C-O bonds, with dissociation energy ~85 kcal/mol (comparable to ethers or alcohols), often represent hydroxyl (-OH) or ether linkages in pharmaceuticals, enabling the oxygen to act as a hydrogen-bond acceptor (strength ~5-10 kcal/mol per bond) that "glues" the molecule to polar residues like serine or asparagine in HIV reverse transcriptase's active site. Removing this bond isolates node 11 (no connections in counterfactual), effectively detaching a functional group and shortening the molecule by ~1.43 Å (typical C-O length). This breaks the structural integrity, shifting node 8's hybridization (from sp3 to potentially planar sp2, exposing pi-orbitals) and altering the chain's flexibility—reducing rotational entropy by ~4-6 cal/mol·K and raising the energy barrier for conformational selection (from ~2 kcal/mol to 5-7 kcal/mol, per induced-fit models). In the "key and lock" analogy, this severs a critical prong on the key, misaligning the molecule in the protein pocket and weakening overall binding (ΔG penalty ~3-6 kcal/mol), as the lost oxygen can no longer form the 2-3 hydrogen bonds typical in active HIV inhibitors. No edges are added; the rest of the graph (e.g., the N-heterocycle around nodes 5,9,14-16 and the aliphatic tail from 19-38) remains intact, preserving core scaffold but compromising peripheral interactions.
Coherent Causal Narrative
Starting from the factual active molecule, a protease or reverse transcriptase inhibitor with a polar head (oxygens at 0,11,12,18,20,32) and charged tail (N+ at 15, N- at 16), the GNN predicts efficacy because the C8-O11 bond positions the oxygen for hydrogen bonding to the enzyme's Gln or Asn residues, while node 7's standard carbon (chem=1.0) maintains a hydrophobic linker that threads through the pocket without steric clash. By deactivating node 7's chem to 0.0, electron withdrawal reduces its nucleophilicity, subtly polarizing nearby bonds (e.g., C6-C7 lengthens by ~0.02 Å due to weakened sigma overlap) and increasing the molecule's overall dipole moment by ~1-2 Debye, which favors aqueous solvation over protein burial—tying into physical chemistry where higher polarity boosts entropy but cuts enthalpy-driven binding. Compounding this, severing the C8-O11 bond orphans the oxygen, collapsing a local loop and enforcing a rigid, extended conformation (via loss of ~60° rotational freedom), which misorients the charged N-pair away from the catalytic site. This dual hit—electronic tweak plus structural break—cascades to halve hydrogen-bond sites (TPSA drops ~20 Å²), elevate lipophilicity (logP rises ~0.3), and inflate van der Waals clashes, ultimately flipping activity by preventing the induced-fit docking essential for HIV inhibition. Interdisciplinary, this echoes how entropy penalties in confined enzyme pockets (e.g., ~TΔS = 0.6 kcal/mol at 300K) amplify small changes into total failure, as seen in clinical analogs like altered AZT derivatives.

Qwen2.5-14B Response

Analyzing the Differences Between the Factual and Counterfactual Graphs
Core Facts and Classification Change
The factual graph is classified as active (class 1) against HIV, while the counterfactual graph is classified as inactive (class 0).\n\n####
High-Level Summary
The molecule was reclassified from active to inactive by modifying the chemical properties of several atoms and altering some of the chemical bonds within the molecule. These changes disrupted the molecule's structure, making it less effective at binding to the target protein.
Detailed Analysis of Graph Changes
Node Feature Changes
- **Node 7**: The chem attribute changed from 1.0 to 0.0. This indicates a change in the electronic properties of the carbon atom, potentially affecting its ability to participate in hydrogen bonding or other interactions.
- **Node 8**: The chem attribute changed from 1.0 to 1.0 but its position and connections suggest a significant shift in its role within the molecule. This change likely altered its contribution to the overall molecular structure and function.
- **Node 11**: The chem attribute remained 2.0, but its position shifted significantly, indicating a relocation of the oxygen atom. This change could affect hydrogen bonding capabilities and disrupt the molecule's shape.
- **Node 37**: The chem attribute changed from 1.0 to 1.0, but its position and connections indicate a substantial alteration in its role. This change may have affected its interaction with neighboring atoms.
Edge Additions/Removals
Edge Removals: Several edges were removed, notably between nodes 7 and 8, and between nodes 11 and its previous connections. These removals signify broken chemical bonds, which fundamentally alter the molecule's 3D structure.\
Edge Additions: New edges were added between nodes that previously had no direct connection, such as between nodes 37 and its new neighbors. These additions suggest the formation of new bonds, further altering the molecule's shape and flexibility.
Coherent Causal Narrative
The molecule was initially classified as active because its structure allowed it to effectively bind to the target protein. However, when we modified the chemical properties of nodes 7 and 8, and relocated node 11, the molecule's shape became distorted. Additionally, breaking bonds between nodes 7 and 8, and forming new ones around node 37, disrupted the molecule's ability to fit into the protein's active site. These structural changes prevented the molecule from interacting properly with the protein, leading to a reclassification as inactive.

Baseline Response

The graph classification changed from Factual Class 1 to Counterfactual Class 0.

Regarding node attribute modifications: for Node 0, attribute 'x' changed from `16.33` to `-11.32`; for Node 1, attribute 'x' changed from `17.14` to `-2.61`; for Node 2, attribute 'x' changed from `17.95` to `-0.44`; for Node 3, attribute 'x' changed from `17.64` to `-2.2`; for Node 4, attribute 'x' changed from `16.64` to `-1.07`; for Node 5, attribute 'x' changed from `17.14` to `-8.85`; for Node 6, attribute 'x' changed from `18.01` to `-9.82`; for Node 7, attribute `chem` changed from `1.0` to `0.0`; for Node 8, attribute `y` changed from `-4.99` to `4.23`; for Node 9, attribute 'x' changed from `16.28` to `-8.35`; for Node 10, attribute 'x' changed from `16.28` to `-9.48`; for Node 11, attribute `y` changed from `-5.99` to `5.31`; for Node 12, attribute 'x' changed from `15.41` to `-8.66`; for Node 13, attribute `y` changed from `-4.99` to `5.97`; for Node 14, attribute 'x' changed from `18.23` to `-14.95`; for Node 15, attribute 'x' changed from `19.22` to `-3.95`; for Node 16, attribute 'x' changed from `20.22` to `-16.78`; for Node 17, attribute 'x' changed from `16.05` to `-6.72`; for Node 18, attribute 'x' changed from `15.06` to `-6.66`; for Node 19, attribute 'x' changed from `14.47` to `-7.34`; for Node 20, attribute 'x' changed from `14.88` to `-0.86`; for Node 21, attribute 'x' changed from `13.48` to `-10.31`; for Node 22, attribute 'x' changed from `12.89` to `-9.85`; for Node 23, attribute 'x' changed from `11.9` to `-10.73`; for Node 24, attribute 'x' changed from `11.31` to `-6.09`; for Node 25, attribute 'x' changed from `10.31` to `-3.38`; for Node 26, attribute 'x' changed from `9.73` to `-3.72`; for Node 27, attribute 'x' changed from `8.73` to `-2.6`; for Node 28, attribute 'x' changed from `8.14` to `-0.34`; for Node 29, attribute 'x' changed from `7.15` to `-4.12`; for Node 30, attribute `y` changed from `4.58` to `-2.05`; for Node 31, attribute `y` changed from `4.47` to `-1.2`; for Node 32, attribute `y` changed from `5.28` to `-0.89`; for Node 33, attribute `y` changed from `5.18` to `-1.62`; for Node 34, attribute `y` changed from `4.26` to `-2.08`; for Node 35, attribute `y` changed from `4.16` to `-2.77`; for Node 36, attribute `y` changed from `4.97` to `-2.79`; for Node 37, attribute `y` changed from `5.88` to `-4.28`; and for Node 38, attribute `y` changed from `5.99` to `-4.5`.

In terms of edge modifications, the edge connecting Node 8 and Node 11 was removed.

Figure 3.13: Narrative explanations for a GNNExplainer counterfactual on a sample from the AIDS dataset, as generated by Grok-4, Qwen2.5-14B, and a baseline model.

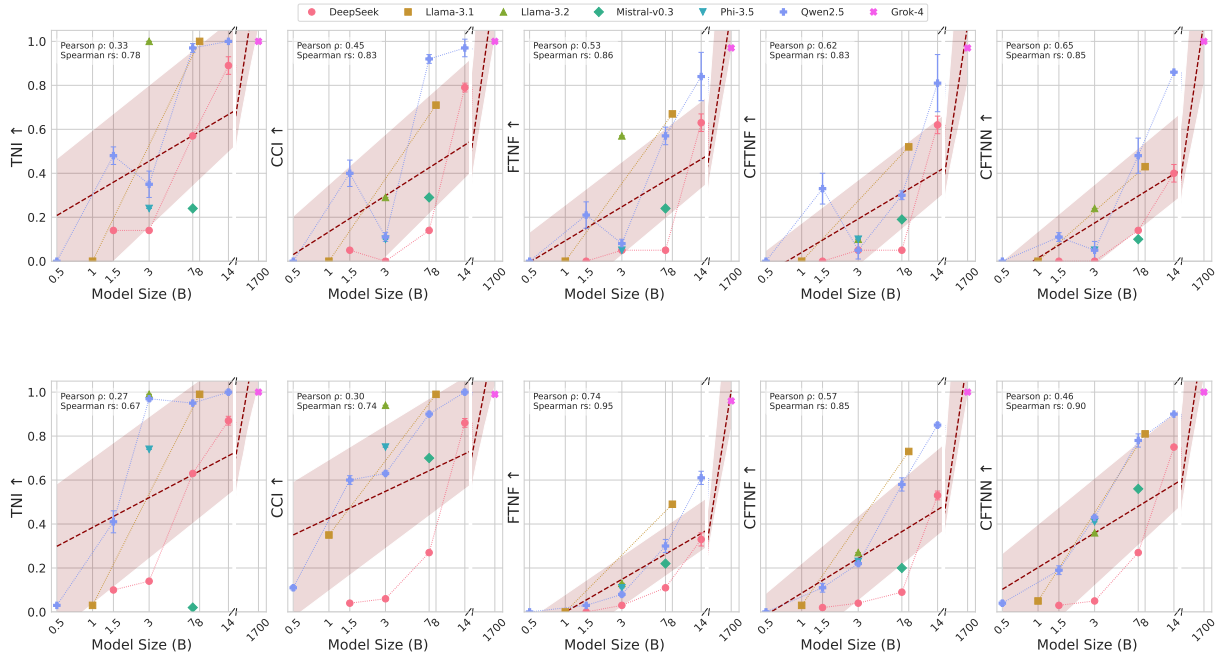


Figure 3.14: Top: Results for the Cora dataset using CF-GNNExplainer. **Bottom**: Results for the Cora dataset using CF-GNNFeatures. The plots have a broken x-axis using a log scale.

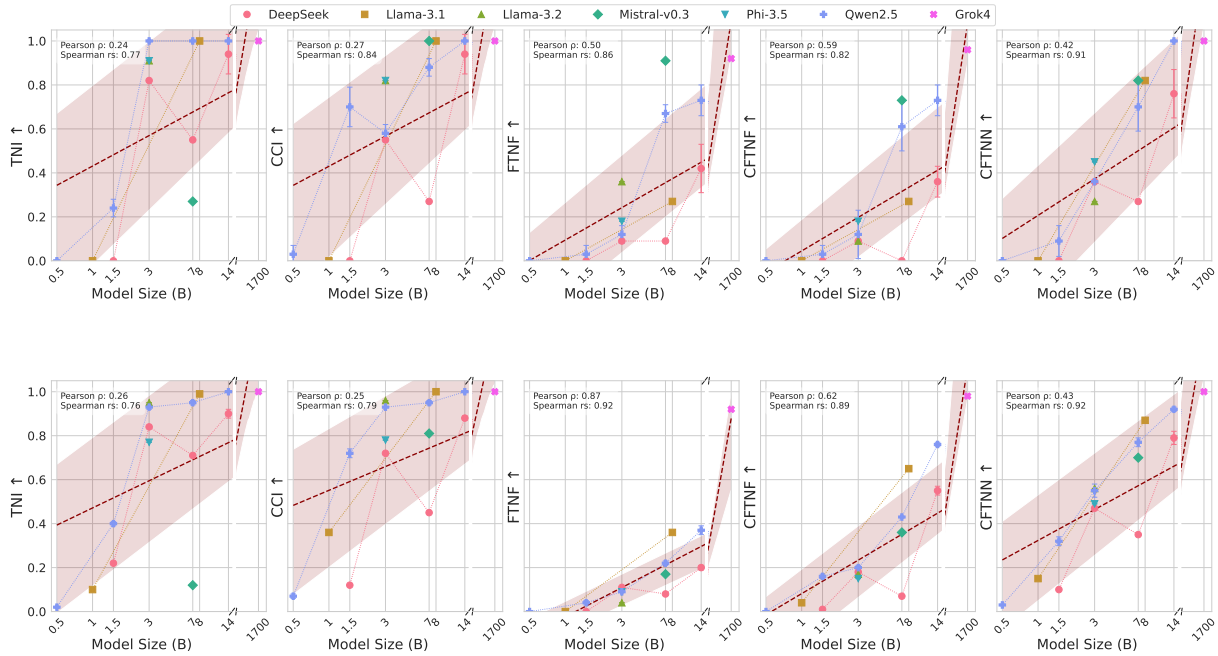
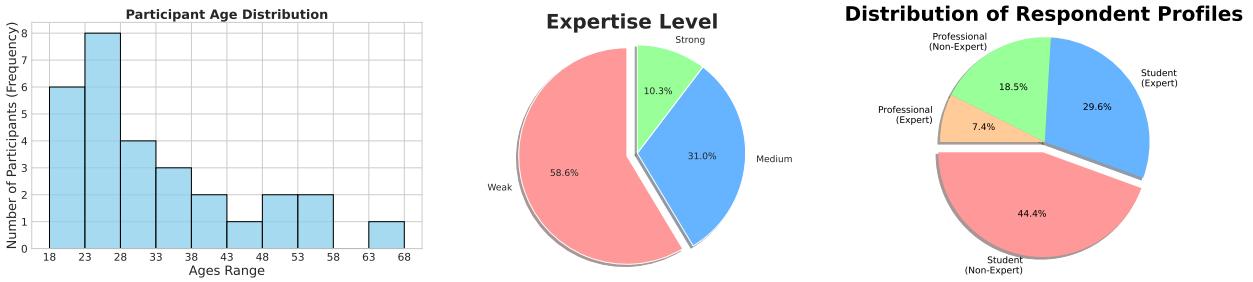


Figure 3.15: Top: Results for the Citeseer dataset using CF-GNNExplainer. **Bottom**: Results for the Citeseer dataset using CF-GNNFeatures. The plots have a broken x-axis using a log scale.

correlation emerges between model size and metric scores, with strong Spearman rank correlations ($rs \in [0.67, 0.95]$) indicating monotonic improvements, while Pearson correlations ($\rho \in [0.24, 0.90]$) reveal a non-linear relationship, suggestive of diminishing returns at extreme scales. For the AIDS dataset (Figure 3.11) in graph classification, the plots for FTC, CTC, NDIFF, and EDIFF demonstrate robust scaling effects, particularly pronounced for structural metrics. FTC and CTC exhibit moderate correlations (FTC: $\rho = 0.57$, $rs = 0.75$; CTC: $\rho = 0.52$, $rs = 0.73$). NDIFF exhibits weaker linearity ($\rho = 0.33$, $rs = 0.78$), with small models near 0 and large models at 0.6-0.8, indicating improved detection of perturbed nodes as the scale increases. Notably, EDIFF yields the



(a) Age Distribution **(b)** Expertise Level **(c)** Participant Profiles

Figure 3.16: Demographics of the 29 participants in the human evaluation study. Figure (a) shows the age distribution, (b) the self-reported expertise level, and (c) the distribution of participant profiles.

highest Pearson correlation across all plots ($\rho = 0.90$, $rs = 0.84$). Shifting to node classification, in the Cora dataset (Figure 3.14), the correlations are consistent but vary by explainer. Using CF-GNNExplainer, Spearman values range from 0.78 (TNI) to 0.86 (FTNF), and Pearson ρ from 0.33 (TNI) to 0.65 (CFTNN), showing strong monotonicity but moderate linearity. Small models cluster below 0.4 on TNI and CCI, with outliers like DeepSeek (0.5B) near 0.1, while large models exceed 0.9, particularly on feature metrics (FTNF: $\rho = 0.53$, $rs = 0.86$; CFTNF: $\rho = 0.62$, $rs = 0.83$). The structure-focused CFTNN metric displays the highest Pearson value (0.65), suggesting scale aids in articulating neighbor changes on Cora's dense citations. With CF-GNNFeatures, correlations strengthen for features (FTNF: $\rho = 0.74$, $rs = 0.95$ among the highest Spearman values), though TNI and CCI show lower Pearson values (0.27 and 0.30), with rs at 0.67-0.74. Finally, also CFTNF ($\rho = 0.57$, $rs = 0.85$) and CFTNN ($\rho = 0.46$, $rs = 0.90$) show positive correlation. On the CiteSeer dataset (Figure 3.15), patterns mirror those of Cora, but with nuanced differences such as lower average Pearson values. In Tables 3.16 and 3.17, it is possible to see the statistical significance analysis for the correlation values. These results confirm that larger LLMs improve explanation quality through superior reasoning and contextual understanding, with feature perturbations benefiting most from scaling. The divergence in correlation types highlights non-linear dynamics, warranting future explorations into fine-tuning for optimal efficiency in graph-to-text XAI tasks. For a discussion about the intrinsic limitations in the evaluation phase, see Section 3.4.4.

Table 3.16: Statistical significance analysis for correlations on the AIDS dataset (graph classification task). The table reports the p-values for the correlation between language model size and quantitative metrics, derived from counterfactuals identified by GNNExplainer. Both Pearson (linear) and Spearman (monotonic) correlations are presented, calculated using a sample of n = 15 models. Bold values indicate statistical significance at the $p \leq 0.05$ level.

Correlation	FTC	CTC	NDIFF	EDIFF
Pearson	**0.027**	**0.047**	0.230	**<0.001**
Spearman	**0.001**	**0.002**	**<0.001**	**<0.001**

Feasibility

To evaluate the practical viability of our approach, we measured the computational overhead in terms of inference latency and power consumption. The results, detailed in Table 3.18, confirm that generating counterfactual narratives with LLMs is feasible across a range of model scales. Analysis of inference times for locally-run models reveals a non-monotonic relationship between

Table 3.17: Statistical significance analysis for correlations on the Cora and CiteSeer datasets (node classification task). The table presents p-values for the correlation between model size and quality metrics, distinguishing between explanations based on CF-GNNExplainer and CF-GNNFeatures modifications. The analysis is based on counterfactuals generated by GNNExplainer (with n=15 models). Bold values are significant at the $p \leq 0.05$ level.

Dataset	Change Type	Correlation	TNI	CCI	FTNF	CFTNF	CFTNN
Cora	*CF-GNNExplainer*	Pearson	0.230	0.092	**0.042**	**0.014**	**0.009**
		Spearman	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**
	CF-GNNFeatures	Pearson	0.330	0.277	**0.002**	**0.027**	0.084
		Spearman	**0.006**	**0.002**	**<0.001**	**<0.001**	**<0.001**
CiteSeer	*CF-GNNExplainer*	Pearson	0.389	0.330	0.058	**0.021**	0.119
		Spearman	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**
	CF-GNNFeatures	Pearson	0.349	0.369	**<0.001**	**0.014**	0.110
		Spearman	**0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**

model size and latency, with the DeepSeek-7B model, for instance, showing higher latency than its 14B counterpart. The Grok-4 model's latency, which includes network overhead due to its API-based access, was competitive on node classification tasks but increased substantially on the more complex graph classification task (it must be taken into account that Grok-4 runs on high-end proprietary hardware). Highlighting the trade-off between performance and speed, the Qwen2.5-3B model emerged as a balanced option, frequently achieving the lowest inference times (e.g., 3.0s on AIDS) while maintaining strong explanation quality. In terms of energy efficiency, power consumption for local models generally scaled inversely with model size. The Qwen2.5 family was particularly noteworthy, with the 0.5B variant establishing a lower bound for power draw (150-170W) and the 3B model offering an exceptional balance of low latency and modest power consumption (240W). Power consumption data for Grok-4 is unavailable as it was accessed via a web API. These findings underscore that a spectrum of viable options exists. For applications where real-time performance is critical, lightweight models like Qwen2.5-3B offer a compelling blend of speed and quality. For scenarios where explanation fidelity is paramount, larger models remain a practical choice, with API-based access presenting an alternative to local deployment.

Table 3.18: Inference Time (in seconds) and Power Consumption (in Watts) for each model, dataset, and explainer. The best result in each column (lower is better) is highlighted in bold.

		Cora				**Citeseer**				**AIDS**	
		CF-GNNFeatures		*CF-GNNExplainer*		*CF-GNNFeatures*		*CF-GNNExplainer*		*GNNExplainer*	
Model	**Size**	**Time ↓**	**Watts ↓**	**Time ↓**	**Watts ↓**	**Time ↓**	**Watts ↓**	**Time ↓**	**Watts ↓**	**Time ↓**	**Watts ↓**
DeepSeek	14B	17.3±0.3	321.8±0.3	17.5±0.3	315.8±0.4	18.5±0.3	326.4±0.8	14.0±1.3	312.9±2.0	16.9±0.6	324.1±0.3
	7B	25.5±0.8	331.2±0.5	20.0±0.6	319.4±0.7	24.9±0.5	331.3±0.4	25.1±0.8	319.4±0.1	28.8±0.9	329.2±0.4
	3B	11.1±0.3	319.6±0.2	12.6±0.5	313.8±0.5	9.6±0.3	326.0±0.3	7.7±0.2	313.5±1.0	14.6±0.6	322.7±0.5
	1.5B	9.2±0.4	285.5±0.3	11.9±0.5	274.3±0.0	8.7±0.4	286.6±0.1	14.2±0.0	267.0±2.4	9.5±0.4	281.9±0.8
Grok-4	1700B	13.8±0.4	n.a.	14.9±0.5	n.a.	14.9±0.5	n.a.	11.8±0.4	n.a.	39.0±1.2	n.a.
Llama-3.1	8B	12.7±0.5	331.7±0.1	13.6±0.5	318.5±0.5	14.1±0.6	331.7±0.2	11.2±0.5	319.4±0.3	17.3±0.7	324.0±6.7
	1B	6.7±0.2	291.2±0.3	2.6±0.1	276.2±0.7	6.0±0.2	290.3±0.4	9.1±0.4	276.3±0.4	8.4±0.3	291.0±0.7
Llama-3.2	3B	4.7±0.2	316.2±0.4	6.1±0.2	304.7±0.2	4.9±0.2	316.7±0.4	6.4±0.3	304.0±0.4	6.6±0.4	309.8±7.3
Mistral-v0.3	7B	11.4±0.3	327.1±0.4	10.4±0.3	314.1±2.1	13.5±0.4	327.6±0.3	12.5±0.4	313.5±3.1	11.6±0.4	324.8±4.9
Phi-3.5	3B	17.2±0.5	313.9±0.2	20.6±0.8	303.7±0.3	17.9±0.7	313.6±0.7	12.2±0.0	302.3±0.6	17.3±0.1	315.3±0.6
Qwen2.5	14B	8.7±0.1	329.1±0.3	9.6±0.2	315.7±0.1	9.3±0.2	329.8±0.2	9.6±0.1	314.8±0.1	15.1±0.3	329.3±0.6
	7B	4.8±0.2	311.1±0.4	4.7±0.0	297.2±0.5	5.4±0.1	311.7±0.0	5.1±0.4	294.4±1.1	5.5±0.1	309.5±1.0
	3B	3.5±0.1	248.1±0.6	3.8±0.1	235.1±0.5	4.4±0.1	248.0±0.2	4.5±0.1	236.0±0.4	3.0±0.1	247.0±0.3
	1.5B	6.6±0.3	214.2±0.9	5.9±0.3	204.4±1.8	6.2±0.1	217.1±0.5	8.0±1.8	197.1±7.9	8.0±1.0	206.8±2.5
	0.5B	8.6±0.2	162.8±0.9	8.1±1.0	146.2±1.2	7.8±0.2	172.0±0.3	9.4±0.3	148.1±0.8	10.1±0.2	157.5±0.7

3.4.4 Limitations

It is important to contextualize our findings by acknowledging the limitations of our research methodology and experimental setup. In this section, we discuss the constraints that influenced our work, which fall into two main categories: first, the hardware and model access limitations that shaped our experimental design; and second, the reduced scale of our human evaluation. These factors should be considered when interpreting our results, as they offer clear avenues for future work.

Hardware and Models

A primary limitation of our study stems from hardware constraints. Our experiments were conducted on a workstation equipped with an NVIDIA RTX 4090 GPU, which has 24GB of VRAM. While sufficient for models up to 14 billion parameters (using GPTQ Int-4 quantization), this hardware setup prevented us from locally running larger open-source models. The graph encoding methodology (Fatemi et al. [62]), which translates graph structures into a textual format, takes a significant number of tokens. For models exceeding the 14B scale, the memory required to process these extensive context lengths surpasses the available 24GB of VRAM.

This hardware limitation also led to a gap in our analysis, as we transitioned directly from 14B-parameter models to a 1.7T-parameter model (using it via API). The associated monetary costs for running our full suite of experiments on models ranging from 32B up to Grok-4-like sizes were prohibitive. Among the closed-source models available, we selected Grok-4 as our representative for very large models primarily because it offered the most economical API access among its peers, allowing us to include a data point from this model scale.

Furthermore, these computational constraints necessitated a reduction in the scope of our experimental validation. Our methodology was evaluated using a limited number of datasets (three in total) and GNN explainers (also three in total). While the latter is a minor point, as our framework is designed to be explainer-agnostic, a broader validation would increase confidence in its generalizability. Additionally, to maintain computational tractability, our quantitative results were averaged over only three random seeds. A comprehensive evaluation across more datasets, additional explainers, and a greater number of seeds was computationally infeasible due to the extensive time required on the available hardware.

Human Evaluation

Another limitation lies in the scope of our human evaluation. The sample size for our user studies was relatively small, consisting of 23 participants for the first test and 29 for the second. This was a consequence of relying on a pool of unpaid volunteers. While the feedback gathered provides valuable preliminary insights, the small and potentially homogeneous sample may not be representative of a broader user population. Therefore, the findings from the human evaluation should be considered indicative rather than definitive, and further studies with a larger, more diverse group of participants would be necessary to validate and generalize these results.

3.4.5 Practical Implications

The proposed framework for generating natural language narratives using LLMs holds significant promise for enhancing interpretability and transparency in graph-based machine learning models. This approach has several practical implications across a wide range of domains that utilize complex graph structures. These include not only broad areas such as cybersecurity and social network analysis, but also more specific, high-stakes applications, such as financial services (e.g., fraud detection and credit scoring), personalized recommendation systems, and biomedical research. For instance, our experiments on the AIDS dataset demonstrate a direct application in drug discovery, where explaining a model's prediction about a molecule's antiviral activity is crucial.

A core benefit of our method is its ability to translate complex, structured counterfactuals into accessible narratives, thereby bridging the gap between algorithmic outputs and human comprehension. This makes the decision-making process of a model transparent to a diverse range of stakeholders, including business managers, medical practitioners, policymakers, and end-users. By verbalizing 'what-if' scenarios, the framework can foster greater trust and user engagement with AI systems, moving beyond opaque predictions to provide clear and actionable insights.

Furthermore, the framework directly addresses the growing need for regulatory compliance. Mandates such as the European Union's General Data Protection Regulation (GDPR), which establishes a 'right to explanation', and the upcoming EU Artificial Intelligence Act, require AI systems in high-risk settings to be explainable. Our method provides a concrete solution by generating explanations that are not only technically sound and faithful to the source counterfactual but also easily interpretable by auditors and users, thus aiding compliance with legal and ethical standards.

Beyond its conceptual value, our work also establishes the operational viability of this approach. The feasibility analysis presented in Section 4.8 demonstrates a clear trade-off between explanation fidelity and computational cost, offering a spectrum of deployable options. For applications where real-time performance is critical, relatively small and efficient models like Qwen2.5-14B offer a compelling blend of speed and quality. Conversely, for scenarios where maximal explanation fidelity is paramount, larger models remain a practical choice, with API-based access presenting a viable alternative to local deployment. This flexibility ensures that our framework can be adapted to different operational constraints and business needs.

In summary, our framework offers a comprehensive solution for making complex graph-based models more transparent and trustworthy. It enhances the utility of counterfactuals, transforming them into powerful tools for model evaluation, debugging, and refinement, thereby enabling the broader and more responsible adoption of AI technologies in critical, high-stakes domains.

3.4.6 Conclusion and Future Work

In this work, we introduced a novel framework for generating natural language narratives for graph-based machine learning models by leveraging the generative capabilities of state-of-the-art LLMs. Our method addresses a key limitation in the current literature: the absence of intuitive, accessible explanations for GNNs, whose predictions often rely on intricate structural dependencies that are difficult to interpret. By translating raw counterfactual examples into coherent textual narratives, our approach enhances the transparency and interpretability of GNN-based models for end-users.

We demonstrated the generality and flexibility of our framework across multiple graph counterfactual explanation methods, tasks, and datasets, showing that it is model-agnostic and does not rely on any specific counterfactual generator. To assess the effectiveness of the generated explanations, we introduced a set of novel evaluation metrics that capture both semantic fidelity and graph-structure awareness. We further validated our results through human evaluation. Notably, the experiments were conducted using both open-source and closed-source LLMs, without any task-specific fine-tuning, which highlights the practicality and scalability of the proposed solution.

In summary, our contributions mark a meaningful step toward bridging the gap between complex graph-based counterfactuals and human-centered interpretability, thereby promoting more transparent, trustworthy, and user-aligned machine learning systems in graph domains.

Looking ahead, future work could explore several promising directions to build upon our findings. One key area is the integration of more sophisticated generation techniques beyond direct prompting, such as fine-tuning LLMs on domain-specific corpora to produce more contextually aware and accurate explanations. Another exciting avenue involves employing multi-agent systems, where different specialized LLM agents could collaborate: for instance, one agent could analyze structural graph modifications, another could interpret feature changes, and a third "narrator" agent could synthesize these distinct analyses into a single, coherent narrative. Finally, further research into advanced graph encoding methods is warranted. Developing novel graph-to-text serialization techniques that more effectively capture complex topological properties and semantic relationships could significantly reduce information loss during the translation process, thereby enhancing the LLM's underlying understanding of the graph dynamics and leading to even more faithful and insightful explanations.

Chapter 4

Explainable AI Beyond Graphs

4.1 Introduction

Modern AI systems are increasingly deployed in settings where decisions must be both accurate and understandable. Much of this thesis has examined explainability in graph-structured domains; here we step beyond graphs and ask a broader question: can explanation be made actionable for both learning and communication, so that models generalize better and people actually understand why a prediction changes?

The first section of this chapter reframes counterfactuals not only as a way to interpret a trained model, but as a signal that can shape the model during training. Intuitively, if very small perturbations are enough to flip a prediction, the decision boundary may be overly contorted and the model prone to overfitting. We turn this intuition into a training principle by encouraging a margin between each instance and its nearby counterfactuals. In practice, this takes the form of a counterfactual-aware penalty that rewards decision boundaries that remain stable under small, meaningful changes to the inputs. An appealing by-product is that counterfactuals generated during learning can later be reused at test time, amortizing the cost of explanation rather than incurring it post hoc.

The second section addresses a different, but equally practical gap: explanations must be readable. Raw feature changes rarely suffice for end users, who need short, precise narratives that say what changed, why it mattered, and how those changes interact. To meet this need, we adopt a draft-and-refine approach that produces multiple candidate narratives and then synthesizes them into a single, coherent explanation. The design favors small, efficient language models guided by larger teachers, balancing quality with computational cost. Alongside this, we employ simple, objective checks of faithfulness, namely, verifying that the narrative matches the factual and counterfactual values, together with human judgments of clarity and usefulness.

Taken together, these two lines suggest a view of explainability that is both for the model and for the human. Counterfactuals become a tool to regularize learning in data-relevant directions, while narrative refinement turns technical changes into concise, trustworthy stories. The result is an XAI toolkit that travels well beyond graphs: it helps models resist overfitting, helps people understand outcomes, and does so with attention to efficiency and real-world constraints. This chapter draws on and consolidates material from [76, 77]

4.2 Regularization via Counterfactual Explanations

One of the key challenges in machine learning is developing models that can generalize their predictions to new, unseen data beyond the scope of the training set. When the predictive accuracy of a model on the training set far exceeds that on the test set, this indicates a phenomenon known as *overfitting*. In general, the impact of overfitting is more pronounced for highly complex models, such as deep neural networks with billions of parameters. To compensate for the risk of overfitting, these models require massive amounts of training data, which may not always be feasible.

Therefore, several strategies have been proposed in the literature to mitigate the problem of model overfitting. Among these, *early stopping* interrupts the training phase before the model starts learning the noise in the data rather than the actual underlying input/output relationship. Furthermore, *data augmentation* is a technique that artificially increases the training set. For example, in the context of image data, this can include applying translation, flipping, and rotation transformations to input samples. Finally, *regularization* is a collection of training/optimization techniques that try to eliminate irrelevant factors by assessing the importance of features, preventing minor input changes from causing significant output variations.

In this work, we present a novel perspective on model overfitting, establishing a connection with the ability to generate *counterfactual examples* [195]. The notion of counterfactual examples has been successfully used, for instance, to attach post-hoc explanations for predictions of individual instances in the form: "*If A had been different, B would* ***not*** *have occurred*" [176]. Generally, finding the counterfactual example for an instance resorts to searching for the minimal perturbation of the input that crosses the decision boundary induced by a trained model. This task reduces to solving a constrained optimization problem.

In the presence of a strong degree of model overfitting, the decision boundary learned becomes a highly convoluted surface, up to the point where it perfectly separates every training input sample. Therefore, each data point, on average, is "closer" to the decision boundary, making it easier to find the best counterfactual example. The intuition behind this claim is depicted in Figure 4.1 and grounded in the theoretical foundations of margin theory [208].

Following this idea, we introduce a novel *counterfactual regularization* term in the training loss that controls overfitting by enforcing a margin between each instance and its hypothetical counterfactual.

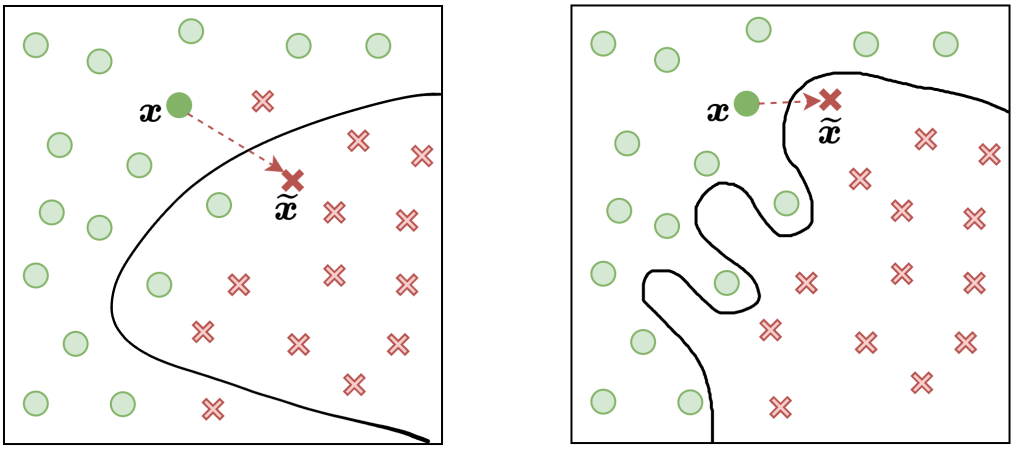
4.2.1 Related Work

Model Overfitting

Several strategies have been proposed in the literature to reduce the effects of overfitting, which can be broadly categorized as follows.

Early Stopping. This strategy aims to mitigate the phenomenon known as "learning speed slowdown." This issue occurs when the accuracy of a model ceases to improve or even worsens due to noise-learning. The concept has a long history, dating back to the 1970s in the context of the Landweber iteration [161]. It has since been widely adopted in iterative algorithms, particularly for training deep neural networks in combination with backpropagation [31].

Data Augmentation. The more complex the model, the higher the number of parameters that



(a) No overfitting **(b)** Overfitting

Figure 4.1: Distance between an input data point ($\boldsymbol{x}$) and its counterfactual example ($\widetilde{\boldsymbol{x}}$): On average, this may be higher for a well-trained model (a) than an overfitted model (b).

need to be learned. Therefore, the size of the training set must be adequate to avoid overfitting. Data augmentation techniques play a crucial role in enhancing model generalization across various domains, including pattern recognition and image processing. These techniques aim to expand existing datasets to generate additional data. Typically, four main approaches are employed [111, 178, 219]: *(i)* acquiring new data, *(ii)* introducing random noise to the existing dataset, *(iii)* reprocessing existing data to produce new instances, and *(iv)* sampling new data based on the distribution of the existing dataset.

Regularization. An overfitting model tends to incorporate all available features into its decision-making process, even those with minimal impact or that are simply noise. To address this issue, there are two main approaches: *(i) Feature selection*, where only the most relevant features are retained, discarding those deemed irrelevant; *(ii) Feature regularization*, which involves minimizing the influence of less important features by reducing their weights in the model. However, since identifying useless features can be challenging, regularization techniques work by applying a penalty term, or "regularizer", to the objective function. This penalizes complex models, encouraging simpler and more generalizable solutions. For example, L1 ("Lasso") and L2 ("Ridge") regularization, which add the L_1-norm and L_2-norm of the learned parameter as a penalties, respectively, are commonly adopted in linear regression. Furthermore, *Dropout* is a popular technique to combat overfitting in neural networks. This approach randomly drops units and relevant connections from the neural network during training to prevent co-adaptation [203].

Unlike existing regularization techniques, which either aim to reduce the magnitude of the learned model's weights (such as Lasso and Ridge) or randomly deactivate certain parameters (such as Dropout), our penalty term is inherently *data-driven*. In other words, the counterfactual regularization we introduce (CF-Reg) is computed directly from the training observations, making it highly tailored to the specific dataset under consideration.

Adversarial Training. Adversarial training is another strategy that can be adapted to mitigate overfitting. Notably, [134] [134] proposed a min-max optimization framework that enhances model robustness by training against adversarial examples generated via Projected Gradient Descent (PGD). This approach serves as an effective form of regularization, as it discourages the learned model from depending on non-robust or spurious patterns [177].

Counterfactual Examples

Counterfactual examples have gained significant attention in machine learning due to their utility in various applications, including model *explainability* and *robustness*.

Explainability. Counterfactual explanations offer valuable insights into model predictions by providing alternative scenarios under which the prediction would change [195]. Several studies have explored the use of counterfactual examples to enhance the interpretability of complex machine learning models, such as ensembles of decision trees [126, 186, 188] and deep neural networks (DNNs) [121], including graph neural networks (GNNs) [40, 129]. Counterfactual instances may elucidate the underlying decision-making process of black-box models, enhancing trust and transparency [43, 81, 109, 146].

Robustness. Counterfactual examples have also been leveraged to enhance the robustness of machine learning models against adversarial attacks [26]. Indeed, there is a strict relationship between adversarial and counterfactual examples [68], although their primary goals are divergent. While both adversarial and counterfactual examples involve perturbing input data to influence model predictions, adversarial examples are crafted to jeopardize the model, whereas counterfactual examples are generated to understand the model's behavior. By generating instances that are semantically similar to the original input but induce different model predictions, He et al. [92] enhance model robustness against adversarial perturbations.

Recent work has explored the use of generative models to produce realistic counterfactual instances for data augmentation and model refinement [69, 93], highlighting the potential of counterfactual reasoning in generative modeling tasks [125].

To the best of our knowledge, however, this is the first study to link counterfactual examples to model generalization and to employ them as the cornerstone of a new regularization strategy.

4.2.2 Background and Preliminaries

Let $f_{\boldsymbol{\theta}} : \mathcal{X} \mapsto \mathcal{Y}$ denote a predictive model parameterized by a set of (learnable) weights $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ that takes an input $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^n$ and maps it to an output $y \in \mathcal{Y}$.

In the standard supervised learning setting, the optimal weights ($\boldsymbol{\theta}^*$) are found by minimizing a specific loss function ($\mathcal{L}$) computed on a training set ($\mathcal{D}$) of m i.i.d. labeled instances, $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$.

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \Big\{ \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) \Big\} = \arg\min_{\boldsymbol{\theta}} \left\{ \frac{1}{|\mathcal{D}|} \sum_{i=1}^{m} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i) \right\}. \tag{4.1}$$

Here, ℓ represents an instance-level error between the model's prediction for a given input ($\boldsymbol{x}_i$) and its corresponding actual label (y_i), such as cross-entropy (for classification tasks) or mean squared error (for regression tasks).

Furthermore, we assume to have available a counterfactual generator model $g_{\boldsymbol{\psi}} : \mathcal{X} \mapsto \mathcal{X}$, for the predictive model $f_{\boldsymbol{\theta}}$, that takes as input a data point $\boldsymbol{x}$ and produces as output its corresponding (optimal) counterfactual $\widetilde{\boldsymbol{x}}^*$.

This typically requires solving a constrained objective as follows:

$$g_{\boldsymbol{\psi}}(\boldsymbol{x}) = \widetilde{\boldsymbol{x}}^* = \underset{\widetilde{\boldsymbol{x}} \in \mathcal{X}}{\arg\min}\ \delta(\boldsymbol{x}, \widetilde{\boldsymbol{x}}) \quad \text{s.t.: } f_{\boldsymbol{\theta}}(\boldsymbol{x}) \neq f_{\boldsymbol{\theta}}(\widetilde{\boldsymbol{x}}). \tag{4.2}$$

Here, $\delta : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}_{\geq 0}$ is a function that measures the distance between the original input data point and its counterfactual. In practice, δ often computes the L_1- or L_2-norm of the displacement vector between the original and the counterfactual example.

In general, for a given input $\boldsymbol{x}$, there could be several, possibly infinitely many *valid* counterfactuals: $\widetilde{\boldsymbol{x}}_1, \widetilde{\boldsymbol{x}}_2, \ldots$ In this work, a valid counterfactual is considered as any instance $\widetilde{\boldsymbol{x}}$ that crosses the decision boundary induced by $f_{\boldsymbol{\theta}}$, i.e., where $f_{\boldsymbol{\theta}}(\boldsymbol{x}) \neq f_{\boldsymbol{\theta}}(\widetilde{\boldsymbol{x}})$. More refined notions of validity are also conceivable, such as those entailing the *plausibility* of the generated counterfactual, which assesses whether the counterfactual example is indeed realistic. For example, suppose we have trained an image classifier to distinguish between birds and rabbits. In this context, a plausible (and valid) counterfactual for a rabbit must be an image resembling a bird. However, for the purpose of this work, we are only interested in characterizing the ability to find a valid counterfactual and relating this to the degree of model overfitting, regardless of its plausibility.

Formally, let $\boldsymbol{x} \in \mathcal{X}$ be an input sample, $f_{\boldsymbol{\theta}}$ a trained model, $\delta : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}_{\geq 0}$ a distance function, and $\varepsilon \in \mathbb{R}_{>0}$ a fixed threshold. We consider an *ε-valid counterfactual example* (ε-VCE) for $\boldsymbol{x}$ any $\widetilde{\boldsymbol{x}} \neq \boldsymbol{x}$, such that $f_{\boldsymbol{\theta}}(\boldsymbol{x}) \neq f_{\boldsymbol{\theta}}(\widetilde{\boldsymbol{x}})$ and $\delta(\boldsymbol{x}, \widetilde{\boldsymbol{x}}) \leq \varepsilon$.

4.2.3 Impact of Counterfactual Examples on Model Generalizability

Intuitive Perspective

We consider a training set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$ of m i.i.d. labeled instances, as introduced above. Then, suppose this dataset is used to train a sequence of k different models $(f_{\boldsymbol{\theta}_0}, f_{\boldsymbol{\theta}_1}, \ldots, f_{\boldsymbol{\theta}_{k-1}})$. Each model $f_{\boldsymbol{\theta}_i}$ is associated with a training accuracy α_i (calculated on $\mathcal{D}$), such that $\alpha_0 < \alpha_1 < \ldots < \alpha_{k-1}$, where $f_{\boldsymbol{\theta}_0}$ indicates the random baseline model and $f_{\boldsymbol{\theta}_{k-1}}$ is the dummy model that simply memorizes the entire training set and, therefore, achieves the perfect training accuracy ($\alpha_{k-1} = 1$).[1]

We claim that for a fixed positive threshold $\varepsilon > 0$, the expected fraction of training points for which we can find an ε-valid counterfactual example is positively correlated with the training accuracy of the model. In other words, given two trained models $f_{\boldsymbol{\theta}_i}$ and $f_{\boldsymbol{\theta}_j}$, where $i > j$, whose training accuracies are $\alpha_i > \alpha_j$, we expect to find, on average, more ε-valid counterfactual examples for $f_{\boldsymbol{\theta}_i}$.

This intuition stems from the idea that a model with a higher training accuracy tends to have a more intricate decision boundary surface, which captures all the nuances of the training data more precisely. Consequently, data points are, *on average*, closer to such a convoluted decision boundary, making it easier to cross it and thus to find valid counterfactual examples within a distance of ε.

In essence, a model for which finding counterfactual examples is "too easy" may indicate a risk of overfitting. Thus, a trade-off must exist between the model's generalizability and its counterfactual explainability, which we aim to investigate further in this study.

[1] In practice, the sequence of k models may correspond to intermediate checkpoints saved at various training epochs.

Theoretical Motivation

The intuition outlined above is supported by a well-established theoretical foundation rooted in classical margin theory. Statistical learning theory tells us that a model's generalization error can be bounded using empirical margin distributions [116]. A prominent example is the support vector machine (SVM), which *explicitly* maximizes the minimum margin on linearly separable datasets.

Interestingly, margin maximization is not exclusive to SVMs. Soudry et al. [174] show that gradient descent applied to logistic regression on linearly separable data *implicitly* maximizes the minimum margin, thereby mirroring the behavior of SVMs. This implicit bias toward margin maximization has also been observed in more complex models, including deep neural networks [83, 105].

Building on these insights, Wu et al. [208] uncover an intriguing trade-off between the minimum and *average* margins, where the latter is defined as the average distance of training examples to the decision boundary. Specifically, they show that optimizing for a larger minimum margin can unintentionally reduce the average margin, thereby increasing the model's vulnerability to adversarial examples. For the same reason, as training progresses and the average distance between data points and the decision boundary decreases, it becomes easier to identify a valid counterfactual example within a fixed perturbation radius ε.

To further validate this claim, Figure 4.2 shows the empirical distribution of margin distances computed from all training points in the `Water` dataset [107] at different training epochs of a logistic regression model. We notice that, at initialization (epoch 0), the distribution is relatively uniform, with a higher average margin distance of 0.754. As training progresses, the distribution of distances to the classifier's boundary becomes increasingly right-skewed, concentrating more of the mass near lower margin values. By epoch 8,000, the average margin distance drops to 0.202.

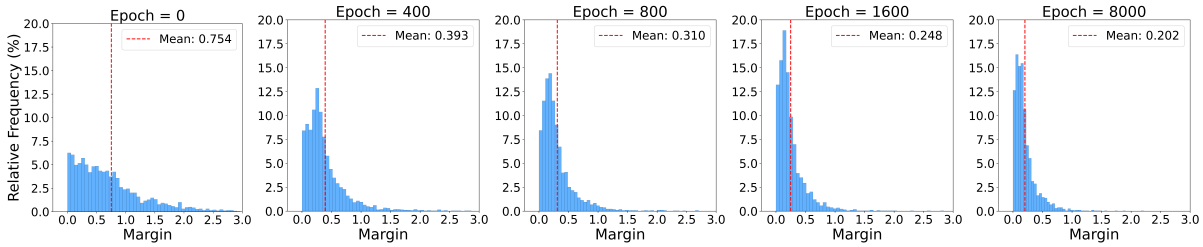


Figure 4.2: Evolution of the empirical distribution of margin distances for training data points in the `Water` dataset across different training epochs of a logistic regression model. As the training progresses, the *average* margin distance decreases.

Estimating Counterfactual Probability

In this section, we formally characterize what we mean by the ease of finding an ε-valid counterfactual example for a given data point and, therefore, for a full training set of data points.

Let us consider the generic predictive model $f_{\boldsymbol{\theta}}$ trained on $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{m}$. Suppose we associate to each training data point $\boldsymbol{x}_i$ a binary random variable $X_i \in \{0, 1\}$, which indicates whether there exists an ε-valid counterfactual example $\widetilde{\boldsymbol{x}}_i$ for $\boldsymbol{x}_i$. Therefore, each X_i follows a

Bernoulli distribution, i.e., $X_i \sim \text{Bernoulli}(p_i^\varepsilon)$, whose probability mass function is defined as below.

$$\mathbb{P}(X_i = k) = p_{X_i}(k; p_i^\varepsilon) = \begin{cases} p_i^\varepsilon & \text{, if } k = 1 \\ 1 - p_i^\varepsilon & \text{, if } k = 0. \end{cases} \tag{4.3}$$

Furthermore, let $\bar{X}$ denote the fraction of training points for which an ε-valid counterfactual example exists. Formally, $\bar{X} = \frac{1}{m}\sum_{i=1}^{m} X_i$ is the average of m Bernoulli random variables.

By the linearity of expectation, we can calculate:

$$\mathbb{E}[\bar{X}] = \mathbb{E}\Big[\frac{1}{m}\sum_{i=1}^{m} X_i\Big] = \frac{1}{m}\sum_{i=1}^{m}\mathbb{E}[X_i], \tag{4.4}$$

where $\mathbb{E}[X_i] = p_i^\varepsilon$ and, therefore, $\mathbb{E}[\bar{X}] = \bar{p}^\varepsilon$ is the average across the entire dataset. Note that the random variable $Z = \sum_{i=1}^{m} X_i$ follows a Poisson-Binomial distribution, since X_i's are independent but not necessarily identically distributed. Consequently, $\bar{X} = \frac{Z}{m}$ is also a Poisson-Binomial random variable, scaled by a factor of $1/m$.

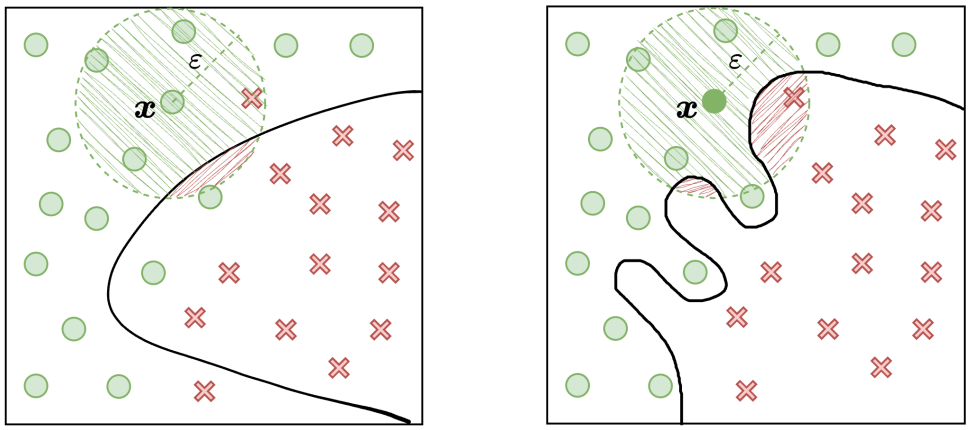
For a fixed $\varepsilon > 0$, we argue that $\mathbb{E}[\bar{X}]$ is positively correlated with the model's training accuracy. This follows from the theoretical result discussed above [208], which shows that higher training accuracy corresponds to a smaller average margin to the decision boundary. Therefore, as the training accuracy of $f_{\boldsymbol{\theta}}$ increases, so does the average fraction of training instances for which a valid counterfactual can be found within distance ε.

Firstly, we need to estimate each p_i^ε, which we refer to as the (sample-level) *ε-valid counterfactual probability* (ε-VCP). Formally, let $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^n$ be a generic training point, and $\varepsilon \in \mathbb{R}_{>0}$ a positive real number. Consider the ε n-ball as the n-dimensional hypersphere of radius ε centered around $\boldsymbol{x}$. Let $V_{\boldsymbol{x}}^\varepsilon \subseteq \mathbb{R}^n$ be the total hypervolume of this hypersphere, and let $V_{\tilde{\boldsymbol{x}}}^\varepsilon \subseteq V_{\boldsymbol{x}}^\varepsilon$ denote the portion of the total hypervolume falling within the counterfactual region delimited by the decision boundary induced by $f_{\boldsymbol{\theta}}$. Therefore, we can estimate the ε-VCP for $\boldsymbol{x}$, as $p^\varepsilon = V_{\tilde{\boldsymbol{x}}}^\varepsilon / V_{\boldsymbol{x}}^\varepsilon$.

Intuitively, given a fixed $\boldsymbol{x}$ and two models having different decision boundaries, it will generally be much easier to find an ε-valid counterfactual if the hypervolume $V_{\tilde{\boldsymbol{x}}}^\varepsilon$ is larger. This intuition is depicted in Figure 4.3, which illustrates how the probability of finding an ε-valid counterfactual for a 2-dimensional data point may increase with model overfitting.

To estimate $V_{\tilde{\boldsymbol{x}}}^\varepsilon$, we can apply standard Monte Carlo integration, a well-established technique that uses random draws to numerically compute a definite integral. Furthermore, using Monte Carlo integration will provide valuable knowledge on the shape of a model's decision boundary as a by-product. The estimate outlined above assumes that data points are uniformly distributed around $\boldsymbol{x}$ in the input space, which may be reasonable for appropriate values of ε. Indeed, it is worth noticing that $\boldsymbol{x}$ can be an embedding from an autoencoding process, mapping each raw training point into a dense, lower-dimensional latent manifold within the original, high-dimensional space. More accurate estimates could be obtained by sampling the ε-neighborhood from the manifold using advanced strategies (e.g., see [43]). However, this lies beyond the primary scope of this step and will be explored in future work.

Finally, if we extend the reasoning above to *all* training points, we should observe that $\mathbb{E}[\bar{X}] = \bar{p}^\varepsilon$ is higher for overtrained models.



(a) Low ε-valid counterfactual probability **(b)** High ε-valid counterfactual probability

Figure 4.3: The *ε-valid counterfactual probability* for a sample $\boldsymbol{x} \in \mathbb{R}^2$ can be estimated as the ratio of the area of the circle centered in $\boldsymbol{x}$ with radius ε that falls behind the decision boundary (in red).

Empirical Validation

To investigate the impact of overfitting on ε-VCP, we trained two Multi-Layer Perceptron (MLP) models on the `Water` dataset [107] for a binary classification task. Both models used the same 5-layer architecture, but only one employed dropout regularization with a rate of 0.5. The models were trained over 500 epochs using the Adam optimizer with a learning rate of $\eta = 0.001$. The ε-VCP was estimated through Monte Carlo integration, as discussed in Section 4.2.3. Specifically, it was computed as $p^{\varepsilon} = V^{\varepsilon}_{\tilde{\boldsymbol{x}}}/V^{\varepsilon}_{\boldsymbol{x}}$, where $V^{\varepsilon}_{\tilde{\boldsymbol{x}}}$ was approximated using 100 random samples per training point $\boldsymbol{x}$.

It is important to note that the choice of ε is inherently dependent on the characteristics of the dataset and is best determined empirically to ensure a meaningful evaluation of counterfactual validity while preserving consistency with the data distribution. As ε defines the maximum norm of the perturbation vector applied to each instance, it delineates the feasible search space for generating counterfactuals. An overly small value may excessively constrain the perturbations, limiting the ability to capture substantive changes in model predictions. In contrast, a value that is too large can produce counterfactuals that are implausible or inconsistent with the underlying data structure. In this experiment with the `Water` dataset, we found that setting $\varepsilon = 1.5$ provides a suitable trade-off, allowing us to capture a diverse yet interpretable range of counterfactual examples without introducing excessive modifications to the original training points.

Figure 4.4 illustrates the evolution of the average ε-VCP, calculated over all training points, as a function of the models' training accuracy. From this plot, three key insights emerge. First, the average ε-VCP increases alongside training accuracy for both models, confirming our hypothesis that the likelihood of finding valid counterfactuals rises as models tend to overfit. Second, the plain, unregularized MLP exhibits higher ε-VCP values, suggesting that its more complex decision boundary makes it easier to identify valid counterfactual examples. Third, the trend persists while the MLP trained with dropout regularization exhibits a less pronounced increase (i.e., smaller ε-VCP values). This suggests that although dropout regularization smooths the decision boundary to some extent, it may not sufficiently counteract this phenomenon, leaving room for further improvement.

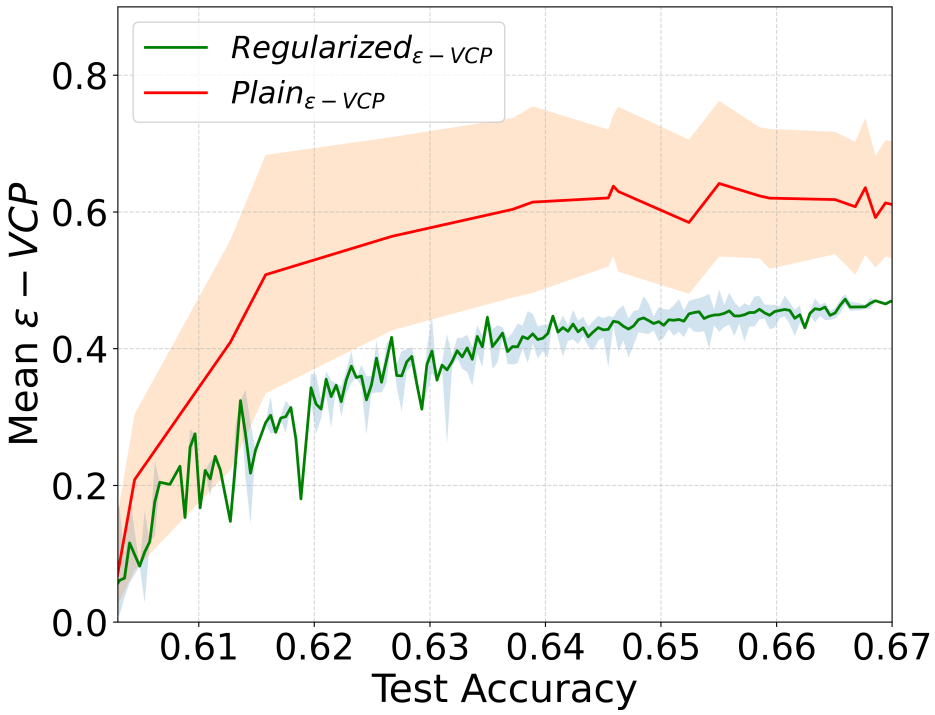


Figure 4.4: The mean ε-VCP (y-axis) vs. the model's training accuracy (x-axis). *Plain*$_{\varepsilon\text{-VCP}}$ is the "vanilla" MLP, while *Regularized*$_{\varepsilon\text{-VCP}}$ is the same MLP yet with a dropout rate of 0.5.

4.2.4 Counterfactual Regularization (CF-Reg)

A New Training Loss

We exploit the correlation between overfitting and the ability to find counterfactual examples, as highlighted in the previous section, to define a new regularized training loss as follows.

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \Big\{ \mathcal{L}_{\text{emp}}(\boldsymbol{\theta}; \mathcal{D}) - \alpha \mathcal{L}_{\text{cf}}(\boldsymbol{\theta}; \mathcal{D}, \varepsilon) \Big\}. \tag{4.5}$$

The first term ($\mathcal{L}_{\text{emp}}$) is the standard empirical risk, while the second term ($\mathcal{L}_{\text{cf}}$) is our proposed *counterfactual regularization* component, with $\alpha \in \mathbb{R}_{\geq 0}$ serving as a hyperparameter to weigh its contribution. More specifically, the counterfactual regularization component can be defined as follows:

$$\mathcal{L}_{\text{cf}}(\boldsymbol{\theta}; \mathcal{D}, \varepsilon) = \varphi(\{d(\boldsymbol{x}_i, g_{\boldsymbol{\psi}}(\boldsymbol{x}_i)),\ \boldsymbol{x}_i \in \mathcal{D}\}; \varepsilon), \tag{4.6}$$

where $\varphi(\cdot; \varepsilon)$ – parameterized by ε – acts as an aggregation function applied to the set of distances $d(\boldsymbol{x}_i, g_{\boldsymbol{\psi}}(\boldsymbol{x}_i))$ between each sample $\boldsymbol{x}_i$ and its corresponding counterfactual generated by the model, $g_{\boldsymbol{\psi}}(\boldsymbol{x}_i)$. Intuitively, the closer an instance is to its counterfactual, the greater the penalty imposed by the newly introduced loss. In other words, optimizing the objective defined in (4.5) aims to ensure a sufficient margin between each training point and its corresponding counterfactual. Moreover, we would like this penalty to be stronger for training points where finding a counterfactual is easier – i.e., those closer to the decision boundary. To achieve this goal, each distance $d(\boldsymbol{x}_i, g_{\boldsymbol{\psi}}(\boldsymbol{x}_i))$ can be assigned a weight w_i^{ε} that depends on ε during aggregation. For example, this weight may be set as the ε-VCP associated with each training instance $\boldsymbol{x}_i$ (i.e., $w_i^{\varepsilon} = p_i^{\varepsilon}$).

Note that, for each data point $\boldsymbol{x}_i$, its corresponding p_i^ε can be estimated as described in Section 4.2.3. However, since p_i^ε depends on the shape of the decision boundary, which may evolve dynamically across training epochs, its estimation should ideally be updated at *every* epoch. Therefore, this could introduce a significant computational overhead in the training process, which we can mitigate updating our estimates $\{p_i^\varepsilon\}_{i=1}^m$ periodically, for example, every t epochs. We conjecture that a trade-off exists between the effectiveness of the counterfactual regularization term – impacted by the accuracy of each estimate p_i^ε – and the computational cost of the training process.

It is worth noticing that our counterfactual regularizer – hereinafter referred to as CF-Reg – is flexible enough to support any choice of aggregation function (φ), distance (d), and counterfactual generator ($g_{\boldsymbol{\psi}}$). However, the optimization problem in (4.5) can be efficiently solved using standard gradient-based methods, provided that φ, d, and $g_{\boldsymbol{\psi}}$ are all differentiable with respect to $\boldsymbol{\theta}$.

In this work, we set φ as the mean and d as the Euclidean distance (i.e., the L_2-norm of the displacement vector resulting from the difference between the original sample and its counterfactual). For $g_{\boldsymbol{\psi}}$, we adopt the *score counterfactual explanation* method proposed by **(author?)** [195], and we discuss the rationale behind this choice below. Alternative formulations of these components are possible and will be explored in future studies.

Counterfactual Example Generator

A key component of CF-Reg is the counterfactual generator $g_{\boldsymbol{\psi}}$, as this is used to compute the optimal counterfactual example for each training data point. To incorporate this step into the regularized training process, the counterfactual generator must satisfy two essential requirements. First, it must be differentiable with respect to the predictive model weights $\boldsymbol{\theta}$. Second, it must be highly efficient to ensure the overall feasibility of our method.

Among the various counterfactual generation approaches in the literature, we selected the *score counterfactual explanation* method proposed by [195], which operates as follows. Let $\boldsymbol{x} \in \mathcal{X}$ be an input sample, $f_{\boldsymbol{\theta}}$ a trained model, $\delta : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}_{\geq 0}$ a distance function, $\beta \in \mathbb{R}_{\geq 0}$ a controlling hyperparameter, and $s \in \mathbb{R}$ a target score. Thus, the score counterfactual explanation $\widetilde{\boldsymbol{x}}$ for $\boldsymbol{x}$ is the result of the following optimization problem:

$$\widetilde{\boldsymbol{x}}^* = \arg\min_{\widetilde{\boldsymbol{x}} \in \mathcal{X}} \left(f_{\boldsymbol{\theta}}(\widetilde{\boldsymbol{x}}) - s\right)^2 + \beta\delta(\boldsymbol{x}, \widetilde{\boldsymbol{x}}). \tag{4.7}$$

With a slight abuse of notation, in (4.7), we treat the output of the predictive model $f_{\boldsymbol{\theta}}$ as a continuous value, even for classification tasks. To accommodate this requirement, we can, for instance, assume access to the logits, which are then passed through the appropriate activation function, such as sigmoid or softmax.

Note that the resulting counterfactual generator satisfies the two requirements mentioned above. In particular, by carefully selecting the distance function δ in (4.7), the resulting objective becomes differentiable with respect to the predictive model weights $\boldsymbol{\theta}$. Furthermore, when using the score-based counterfactual explanation approach, determining the optimal counterfactual perturbation vector $\boldsymbol{\delta} = \boldsymbol{x} - \widetilde{\boldsymbol{x}}^* = \boldsymbol{x} - g_{\boldsymbol{\psi}}(\boldsymbol{x})$ – i.e., the displacement vector between the original instance and its optimal counterfactual – admits a closed-form solution under specific assumptions. Specifically, if $f_{\boldsymbol{\theta}}$ is a linear function and $\widetilde{\boldsymbol{x}}^* = g_{\boldsymbol{\psi}}(\boldsymbol{x})$ is the optimal counterfactual example for the input $\boldsymbol{x}$, [156]

proved that:

$$\boldsymbol{\delta} = \frac{t}{\beta + \|\boldsymbol{\theta}\|^2} \cdot \boldsymbol{\theta}, \tag{4.8}$$

where $t = s - f_{\boldsymbol{\theta}}(\boldsymbol{x})$. This makes the method highly efficient and well-suited for incorporation into our regularized training loss.

From (4.8), we can, therefore, compute the L_2-norm of the perturbation vector $\boldsymbol{\delta}_i$, for each training instance $\boldsymbol{x}_i$. This value is then used as $d(\boldsymbol{x}_i, \widetilde{\boldsymbol{x}}_i^*)$, as required by the proposed regularized training loss in (4.6). Finally, the L_2-norms of all perturbation vectors are aggregated using the function $\varphi(\cdot; \varepsilon)$, resulting in the following (weighted) average:

$$\overline{||\boldsymbol{\delta}||} = \frac{1}{m} \sum_{i=1}^{m} w_i^{\varepsilon} * ||\boldsymbol{\delta}_i||. \tag{4.9}$$

In this work, weights are uniformly set as $w_i^{\varepsilon} = 1 \; \forall i \in \{1, \ldots, m\}$, thus reducing the aggregation to a plain average. Alternatively, more sophisticated strategies can be used, such as the one outlined in Section 4.2.4, where each training point is assigned a weight corresponding to its estimated ε-VCP, i.e., $w_i^{\varepsilon} = p_i^{\varepsilon}$.

Overall, this transforms our counterfactual regularized loss into:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \left\{ \mathcal{L}_{\text{emp}}(\boldsymbol{\theta}; \mathcal{D}) - \alpha \overline{||\boldsymbol{\delta}||} \right\}. \tag{4.10}$$

Finally, the objective in (4.10) can be solved using standard gradient-based optimization methods.

4.2.5 Experiments

In this section, we evaluate the ability of the proposed counterfactual regularized loss (CF-Reg), as defined in (4.10), to mitigate overfitting when compared to existing regularization techniques. In Appendix A.3, instead, we analyze the relationship between the test loss and the counterfactual vector norm during training.

Experimental Setup

Datasets, Models, and Tasks. The experiments are run on four datasets: `Water` [107], `Phoneme` [56], `Higgs` [206], and `CIFAR-10` [119]. Furthermore, we consider the following models: Logistic Regression (LR), two Multi-Layer Perceptrons ($\text{MLP}_{\text{small}}$ and $\text{MLP}_{\text{large}}$), and *PreactResNet-18* [90] as a representative Convolutional Neural Network (CNN) architecture. We focus on binary classification tasks and leave the extension to multiclass scenarios for future work. However, for datasets originally designed for multiclass classification, we adapt them by selecting two classes to align with our binary setting. Additional details are provided in Appendix A.2.1.

Evaluation Measures. To characterize the degree of overfitting, we use the test loss, as it serves as a reliable indicator of the model's generalization capability to unseen data. Additionally, we evaluate the predictive performance of each model using the test accuracy.

Baselines. We compare CF-Reg with the following methods: Early Stopping, Dropout, adversarial training via well-known projected gradient descent (PGD) [134], two regularizers – i.e., L1-Reg

("Lasso") and L2-Reg ("Ridge") – and no regularization (No-Reg).

Hyperparameter Settings. We analyze the sensitivity of CF-Reg to its hyperparameters in Section 4.2.5. Due to space constraints, a comprehensive discussion covering all regularization methods, including the tuning strategies adopted, is provided in Appendix A.4. The best selected values are summarized in Table A.6.

Implementation Details. To deliberately induce overfitting in logistic regression models, we apply a polynomial feature expansion that increases the input dimensionality beyond the number of training samples. This ensures that the model has sufficient capacity to memorize the training data, thereby allowing us to assess the effect of our counterfactual regularizer. The degree of the polynomial expansion is selected as the smallest value that results in a number of features exceeding the number of training instances. Note that this preprocessing step is not required for neural network models, as they are chosen to be complex enough to overfit the data.

In addition, to leverage the closed-form solution for computing the optimal perturbation vector as defined in (4.8), we use a local linear approximation of the model during the counterfactual generation process. Thus, given an instance $\boldsymbol{x}_i$, we compute the (optimal) counterfactual not with respect to the original model $f_{\boldsymbol{\theta}}$, but with respect to its first-order Taylor expansion around $\boldsymbol{x}_i$, denoted as:

$$f_{\boldsymbol{\theta}}^{lin}(\boldsymbol{x}) = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) + \nabla_{\boldsymbol{x}} f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)(\boldsymbol{x} - \boldsymbol{x}_i). \tag{4.11}$$

Note that this step is unnecessary for logistic regression, as it is inherently a linear model.

We run all experiments on a machine equipped with an AMD Ryzen 9 7900 12-Core Processor and an NVIDIA GeForce RTX 4090 GPU. Our implementation is based on the PyTorch Lightning framework, and the source code for our experiments is available at the following GitHub repository: `https://anonymous.4open.science/r/CF-Reg-5AB6/README.md`

Further details are described in Appendix A.2.2.

Effectiveness of CF-Reg

We compare the performance of CF-Reg against the baselines indicated in Section 4.2.5. For each model and dataset combination, Table 4.1 shows the mean value and standard deviation of test accuracy achieved by each method across 5 random initializations. From the results in Table 4.1, the following key insights can be drawn.

General Validity of CF-Reg. Across the board, CF-Reg demonstrates strong performance, particularly on tabular datasets such as `Water`, `Phoneme`, and `Higgs`. In many configurations, it either matches or outperforms all other baselines, and in several cases, the improvements are statistically significant. This supports the core hypothesis behind CF-Reg: that training with informative counterfactual examples can improve generalization by encouraging decision boundary robustness in data-relevant directions.

Comparison with other Regularization Techniques. CF-Reg consistently outperforms traditional penalty-based regularizers like L1-Reg and L2-Reg. These methods penalize model complexity indirectly through weight constraints, whereas CF-Reg directly encourages decision-level stability, which appears more effective in preserving model generalization. Moreover, CF-Reg performs on par with or better than implicit regularization methods such as early stopping and dropout. Notably, in cases where CF-Reg underperforms slightly, it still retains a critical advantage: it leverages

the full expressive capacity of the architecture, rather than curtailing it through early termination of training or by randomly omitting connections during forward passes. This is particularly relevant for modern architectures where expressiveness is key to capturing subtle data patterns. When compared to adversarial training (PGD), CF-Reg also shows almost always superior or comparable accuracy.

Handling Complex Input Domains. On the `CIFAR-10` dataset, CF-Reg does not outperform traditional regularizers. In fact, in this context, the No-Reg baseline performs the best, suggesting that overfitting is less of a concern, perhaps due to the indirect regularizing effect of convolutional architectures. Additionally, the current counterfactual generation strategy appears less effective for image data, where generating meaningful perturbations is substantially harder. This points to a promising direction for future work: using more expressive and domain-aware counterfactual generators tailored to high-dimensional, unstructured inputs, in order to fully extend the applicability of CF-Reg to complex data.

Table 4.1: Mean and standard deviation of test accuracy over 5 random initializations for each combination of model, dataset, and regularization method. Best results are shown in bold; results that are statistically significantly better than the second-best (at the $\alpha = 0.05$ significance level) are marked with a *.

Model	Dataset	No-Reg	L1-Reg	L2-Reg	Dropout	Early Stopping	PGD	CF-Reg (ours)
LR	`Water`	0.6030 ± 0.0053	0.6652 ± 0.0053	0.6677 ± 0.0114	N/A	0.6098 ± 0.0000	0.6384 ± 0.0183	$\mathbf{0.6915 \pm 0.0017}^\star$
$\text{MLP}_{\text{small}}$	`Water`	0.6128 ± 0.0103	0.6098 ± 0.0000	0.6759 ± 0.0065	0.6515 ± 0.0100	0.6765 ± 0.0059	0.6524 ± 0.0011	$\mathbf{0.6796 \pm 0.0045}$
$\text{MLP}_{\text{large}}$	`Water`	0.6168 ± 0.0063	0.6098 ± 0.0000	0.6320 ± 0.0158	0.6564 ± 0.0104	0.6573 ± 0.0112	0.6464 ± 0.0053	$\mathbf{0.6787 \pm 0.0092}^\star$
LR	`Phoneme`	0.8729 ± 0.0052	0.8157 ± 0.0036	0.8427 ± 0.0078	N/A	0.8622 ± 0.0109	0.7353 ± 0.0006	$\mathbf{0.8764 \pm 0.0084}$
$\text{MLP}_{\text{small}}$	`Phoneme`	$\mathbf{0.9016 \pm 0.0088}$	0.8511 ± 0.0092	0.7963 ± 0.0041	$\mathbf{0.9016 \pm 0.0059}$	0.8957 ± 0.0105	0.8915 ± 0.0029	0.9005 ± 0.0065
$\text{MLP}_{\text{large}}$	`Phoneme`	0.9101 ± 0.0026	0.7068 ± 0.0000	0.8735 ± 0.0042	0.9099 ± 0.0078	0.9066 ± 0.0083	0.9021 ± 0.0021	$\mathbf{0.9149 \pm 0.0024}^\star$
$\text{MLP}_{\text{small}}$	`Higgs`	0.7204 ± 0.0043	0.4706 ± 0.0000	0.7231 ± 0.0033	0.7210 ± 0.0016	0.7271 ± 0.0049	$\mathbf{0.7334 \pm 0.0025}^\star$	0.7245 ± 0.0057
$\text{MLP}_{\text{large}}$	`Higgs`	0.6904 ± 0.0020	0.4706 ± 0.0000	0.7223 ± 0.0056	0.7323 ± 0.0039	0.7149 ± 0.0043	0.6995 ± 0.0018	$\mathbf{0.7278 \pm 0.0027}$
CNN	`CIFAR-10`	$\mathbf{0.8415 \pm 0.0059}$	0.8355 ± 0.0160	0.8408 ± 0.0035	N/A	0.8413 ± 0.0008	0.7071 ± 0.0316	0.8388 ± 0.0106

Table 4.2: Mean and standard deviation of training time (in seconds) over 5 independent runs. Each value refers to the training process for the corresponding entry in Table 4.1.

Model	Dataset	No-Reg	L1-Reg	L2-Reg	Dropout	Early Stopping	PGD	CF-Reg (ours)
LR	`Water`	19.86 ± 0.51	20.87 ± 0.33	21.06 ± 0.29	N/A	0.46 ± 0.11	119.23 ± 0.83	21.78 ± 0.54
$\text{MLP}_{\text{small}}$	`Water`	19.78 ± 0.21	22.47 ± 0.47	22.60 ± 0.41	21.20 ± 0.26	1.37 ± 0.49	120.31 ± 0.05	23.07 ± 0.28
$\text{MLP}_{\text{large}}$	`Water`	22.94 ± 0.37	26.25 ± 0.18	27.21 ± 0.42	25.88 ± 0.45	0.76 ± 0.10	157.83 ± 0.05	27.48 ± 0.29
LR	`Phoneme`	29.54 ± 0.40	32.25 ± 0.68	32.22 ± 0.23	N/A	20.40 ± 3.27	128.67 ± 0.05	33.84 ± 0.37
$\text{MLP}_{\text{small}}$	`Phoneme`	31.02 ± 0.56	34.90 ± 0.35	35.55 ± 0.72	33.12 ± 0.18	31.33 ± 0.07	565.60 ± 1.67	36.60 ± 0.72
$\text{MLP}_{\text{large}}$	`Phoneme`	35.66 ± 0.60	41.27 ± 0.99	42.60 ± 0.59	40.25 ± 0.70	20.95 ± 0.70	253.80 ± 0.83	43.95 ± 0.56
$\text{MLP}_{\text{small}}$	`Higgs`	415.16 ± 4.70	470.94 ± 8.46	485.45 ± 8.22	448.73 ± 5.04	156.89 ± 7.03	459.40 ± 1.67	506.11 ± 5.73
$\text{MLP}_{\text{large}}$	`Higgs`	486.08 ± 5.35	571.48 ± 10.70	593.64 ± 8.38	545.84 ± 23.19	29.38 ± 2.73	584.10 ± 1.41	613.39 ± 10.71
CNN	`CIFAR-10`	377.47 ± 1.66	401.37 ± 1.94	404.53 ± 3.54	N/A	318.19 ± 2.20	1304.40 ± 5.62	1294.23 ± 3.88

Hyperparameter Sensitivity Analysis

CF-Reg relies on two key hyperparameters: α and β. The former is intrinsic to the loss formulation defined in (4.2), while the latter is closely tied to the choice of the score-based counterfactual explanation method used.

Figure 4.5 illustrates how the test accuracy of an MLP trained on the `Water` dataset changes for different combinations of α and β.

We observe that, for a fixed β, increasing the weight of our counterfactual regularizer (α) can slightly improve test accuracy until a sharp drop occurs for $\alpha > 0.1$. This behavior is expected, as the impact of CF-Reg, like any regularization term, can be disruptive if not properly calibrated.

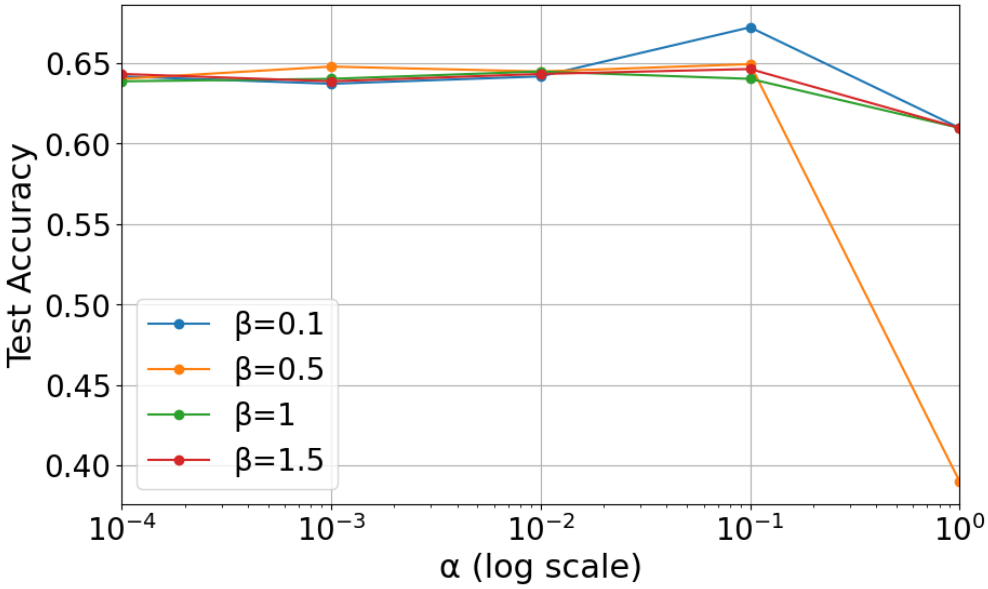


Figure 4.5: The test accuracy of an MLP trained on the `Water` dataset, evaluated while varying the weight of our counterfactual regularizer (α) for different values of β.

Moreover, this finding confirms that CF-Reg is inherently data-driven and, thus, requires specific fine-tuning based on the combination of the model and dataset at hand.

The best values for α and β are shown in Table A.6 (Appendix A.4).

4.2.6 Discussion

Feasibility of CF-Reg

As with any regularizer, CF-Reg introduces some computational overhead in exchange for improved model generalization. Specifically, the extra cost stems from the need to compute a counterfactual example *for each* training instance during learning. Since CF-Reg is compatible with any counterfactual generator, the actual overhead depends on the complexity of the generator used.

Indeed, more advanced counterfactual generators may yield more effective regularization but require longer computation times. Conversely, simpler generators, such as the score-based model employed in this study, offer efficient training at the potential cost of regularization quality. We analyze this trade-off in greater depth in Section 4.2.6.

We quantify the cost of CF-Reg by evaluating its impact on model training time. Specifically, Table 4.2 reports the mean and standard deviation of training time for each model-dataset pair presented in Table 4.1. We observe that CF-Reg introduces only a modest computational overhead when employing the score-based generator. However, on larger datasets, the need to compute counterfactual distances for every training point leads to increased training time, as expected. Interestingly, the number of model parameters does not significantly affect the training time. Instead, input dimensionality plays a more substantial role, especially for nonlinear models. For example, in `CIFAR-10`, gradients for local linearization must be computed in a high-dimensional space, $\mathbb{R}^{3\times32\times32}$, which contributes considerably to the increased training cost.

Counterfactual Explanations as a By-Product

While CF-Reg introduces some computational overhead and does not always significantly outperform all baselines, it offers a unique advantage not provided by traditional regularizers: the generation of counterfactual explanations as a natural by-product of training. These explanations, typically computed through separate and often costly post hoc procedures [43, 146], are instead amortized over training with CF-Reg, yielding improved generalization, possibly without additional inference cost.

At test time, given a new input $\boldsymbol{x}_{\text{new}}$, approximate counterfactual explanations for the prediction $f_{\boldsymbol{\theta}}(\boldsymbol{x}_{\text{new}})$ can be efficiently retrieved by: (i) identifying the k nearest training instances to $\boldsymbol{x}_{\text{new}}$, and (ii) returning their precomputed counterfactual examples via a simple lookup, which takes constant time ($O(1)$) per instance.

Main Limitations and Future Improvements

In this section, we discuss the main limitations of CF-Reg and outline directions for future improvements. Our method was designed to be general and efficient; however, some design choices, while practical, may limit performance in certain settings. Below, we sketch the key areas that are worth investigating.

Efficiency vs. Regularization Trade-off. As introduced in Section 4.2.4, our framework supports *any* counterfactual generator $g_{\boldsymbol{\psi}}$, but this flexibility introduces variability in computational cost. When using lightweight generators (e.g., score-based), training remains fast, but the regularization may be less effective. In contrast, more expressive generators could improve generalization at the expense of longer training times. Exploring this trade-off, and the use of more sophisticated counterfactual generators, is a promising avenue for future research.

To improve the feasibility of CF-Reg, we plan to investigate the following strategies: (i) Cache and update counterfactuals every t epochs instead of at each iteration; (ii) Cluster training points and generate one counterfactual per cluster rather than per sample; (iii) Perform counterfactual search only for a selected subset of influential training points, identified via random sampling or heuristic criteria.

Linear Approximation. Even when using the score-based generator, we do not fully leverage the closed-form solution for computing the optimal perturbation vector in (4.8), which assumes a linear model $f_{\boldsymbol{\theta}}$. For deep neural networks, this assumption generally does not hold. We instead use the first-order Taylor approximation in (4.11), which may yield suboptimal counterfactuals due to its inability to fully capture the model's nonlinear behavior.

Uniform Weighting Scheme. Our current implementation assigns equal importance to each training instance ($w_i^{\varepsilon} = 1$). While simple, this uniform weighting may limit the potential of CF-Reg. A more fine-grained weighting strategy could better balance the regularization strength across samples, though it would likely increase training complexity. Investigating adaptive weighting mechanisms remains an open and worthwhile direction.

Counterfactual Generator Learning. In this work, we treat the counterfactual generator $g_{\boldsymbol{\psi}}$ as a fixed component, queried during the regularized training of the primary model $f_{\boldsymbol{\theta}}$. A promising direction for future research is to integrate the learning of $g_{\boldsymbol{\psi}}$ directly into the training process, thereby formulating the task as a bilevel optimization problem. Solving this problem would allow

for the joint learning of both the predictive model $f_{\boldsymbol{\theta}}$ and its associated counterfactual generator $g_{\boldsymbol{\psi}}$.

4.2.7 Conclusion

We introduced and analyzed the trade-off between the generalizability and explainability of predictive models, highlighting their often conflicting nature. Specifically, based on well-known theoretical results in margin theory, we demonstrated that the degree of model overfitting is positively correlated with its ability to generate counterfactual examples. Building on this insight, we proposed CF-Reg, a novel regularization technique that integrates a counterfactual regularization term into the training objective, to balance between predictive performance and interpretability.

Through extensive experiments across multiple datasets and model architectures, we showed that CF-Reg generally outperforms existing regularization techniques, improving generalization while simultaneously generating meaningful counterfactual explanations. Our results suggest that counterfactual-based regularization can serve as a principled approach to improving model robustness without sacrificing interpretability.

Finally, we outlined promising directions for future work, primarily aimed at improving the efficiency of our method.

4.3 Narrative refinement

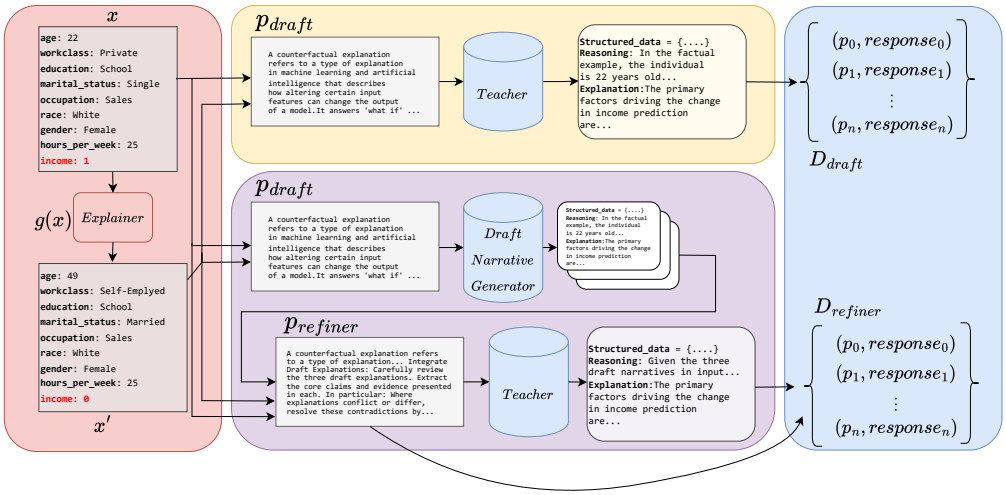


Figure 4.6: Overview of the dataset generation process for the draft narrative generation step and the refiner step using knowledge distillation. Given a factual instance $\boldsymbol{x}$ with its features (e.g., age, work class, education, etc.), a counterfactual generator $g(\boldsymbol{x})$ modifies the instance to create a counterfactual example $\boldsymbol{x}'$, altering specific attributes (e.g., age, work class) to yield a different predicted outcome (e.g., income = 0) (red zone). The factual and counterfactual instances are then integrated into the prompt that is fed into the teacher model, which produces the narrative for the draft narrative generation step (yellow zone) and for the refiner step (purple zone). For the draft narrative generation step, the response is collected and put along with the prompt p_{draft} in the new dataset D_{draft}. Concerning the dataset for the refiner step, we generate $N = 3$ draft responses and integrate all three into the prompt $p_{refiner}$. Finally, we feed the prompt into the teacher, collect the response, and put the newly formed pair $p_{refiner}, response$ into the dataset $D_{refiner}$.

Counterfactual Explanations (CE) have become a cornerstone of Explainable AI (XAI), providing intuitive insights into model decisions by demonstrating how minimal changes to input features can alter outcomes. This approach is increasingly critical in domains like healthcare, finance, and policy-making, where transparency is essential for trust, accountability, and compliance with emerging AI policies. For instance, the European Union's AI Act [61] mandates explainability for high-risk AI systems, while U.S. guidelines on AI governance [152] emphasize interpretable decision-making to mitigate bias and ensure fairness. While early XAI efforts targeted specific data types—such as graph-based models in drug discovery [91], recent work has expanded to diverse modalities, including tabular data and natural language. Large Language Models (LLMs) have played a crucial role in this shift, converting technical counterfactuals into human-readable narratives. Fredes and Vitrià [67] showcase LLM-driven explanations for tabular data counterfactuals, prioritizing user accessibility, while Cheng et al. [44] explore LLMs' role in generating and evaluating counterfactuals for natural language tasks. Similarly, He et al. [91] and Wang et al. [202] demonstrate LLMs' versatility in graph-based contexts, suggesting their potential as a unifying tool across XAI applications.

Building counterfactual narratives, particularly for tabular data, which underpins applications like credit scoring and medical diagnostics, is extremely important to improve models' explainability. Current methods often remain domain-specific or computationally expensive, limiting their scalability and alignment with policy-driven transparency requirements [61, 152]. This paper introduces a novel pipeline for counterfactual narrative generation, centering on tabular data. We propose a methodology leveraging Language Models to produce semantically rich, user-centric counterfactual narratives. By aligning with AI policy mandates for interpretability, this work advances the prac-

tical adoption of counterfactual reasoning in XAI, enhancing model transparency and usability in real-world, regulated environments.

4.3.1 Related Works

Recent advancements in narrative-driven explainable AI leveraging LLMs significantly enhance the interpretability of complex machine learning models across diverse data types. This trend addresses a critical gap between the outputs of XAI algorithms and the comprehension of end-users, in particular, non-experts. A significant portion of this research targets tabular data. For instance, Zeng et al. [221] use a locally deployed LLM to translate the numerical outputs of SHAP [130] into plain-language summaries, enhancing their accessibility. Similarly, Fredes and Vitrià [67] demonstrate that LLMs can effectively explain sets of counterfactual examples by mimicking human reasoning, achieving high validity on benchmark datasets. Going beyond static explanations, Slack et al. [173] develop TalkToModel, an interactive dialogue system that allows users to converse with a model to understand its predictions.

Researchers also develop broader frameworks for generating and evaluating narrative explanations. Cedro and Martens [33] propose a comprehensive survey of narrative-driven XAI, introducing "XAIstories" that translate SHAP values into compelling narratives. Recognizing the need for systematic quality control, Zytek et al. [223] introduce Explingo, a dual-LLM system composed of a narrator to generate narratives from SHAP explanations and an automated grader to assess them on metrics like accuracy, completeness, and fluency. On the evaluation front, Ichmoukhamedov et al. [102] propose quantitative metrics such as faithfulness and human similarity to standardize the assessment of LLM-generated narratives.

The application of narrative XAI extends beyond tabular data to more complex structures like graphs, images, and 3D point clouds. For Graph Neural Networks (GNNs), several approaches have emerged. Giorgi et al. [73], for example, generate counterfactual explanations for graphs and use open-source LLMs to produce human-readable outputs, while Cedro and Martens's [34] GraphXAIN translates explanatory subgraphs into natural language, showing high user satisfaction. Pan et al. [154] develop TAGExplainer to generate faithful and concise explanations for text-attributed graphs, which are common in social networks and recommendation systems. In the visual domain, Castellano et al. [32] combine Grad-CAM heatmaps from image classifiers with LLMs to create textual descriptions that explain why a model focused on certain areas of an artwork. Kočić et al. [115] instead present a framework where an LLM is an active participant in the counterfactual generation process itself, selecting segment perturbations in 3D point clouds to ensure they are semantically meaningful. Finally, these techniques are also being adapted for highly specialized fields. He et al. [91], for example, apply LLMs to explain GNNs used for molecular property prediction, demonstrating the broad applicability of this paradigm.

Together, these works illustrate a clear and growing trend toward using LLMs to make Machine Learning models more transparent and trustworthy.

4.3.2 Background

Before introducing our pipeline, we formalize two important definitions, namely, the Counterfactual Explanation Problem (CEP) and the Counterfactual Narrative Generation Problem (CNGP). We

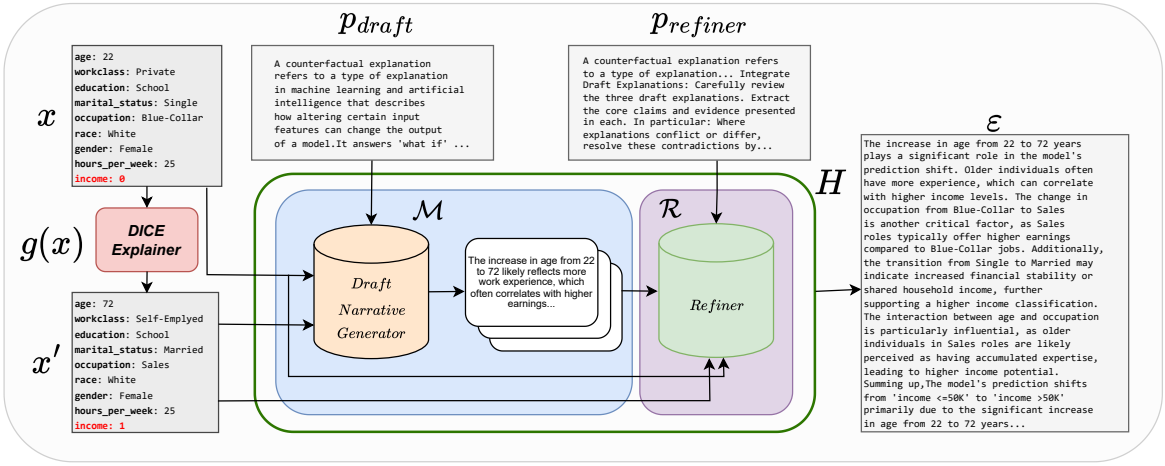


Figure 4.7: Multi-Narrative Refinement pipeline schema

take the definition for the CEP from [73].

Definition 4.3.1 (The Counterfactual Explanation Problem)**.** *Given a sample $\boldsymbol{x}$ and a predictive model $f_{\boldsymbol{\theta}}$ parametrized by $\boldsymbol{\theta}$, hereinafter referred to as oracle, a distance function $d(\cdot,\cdot)$, and a generic counterfactual generator $g(\cdot)$ the goal is to find a sample $\boldsymbol{x}'$ using $g(\cdot)$ where $\boldsymbol{x}' \neq \boldsymbol{x}$ and $f_{\boldsymbol{\theta}}[\boldsymbol{x}'] \neq f_{\boldsymbol{\theta}}[\boldsymbol{x}]$ such that the distance $d(\boldsymbol{x}', \boldsymbol{x})$ is minimized, or $\perp$ if no valid counterfactual example exists. The sample $\boldsymbol{x}'$ is called a counterfactual example for $\boldsymbol{x}$.*

The distance function $d(\cdot,\cdot)$ ensures that the counterfactual sample $\boldsymbol{x}'$ remains as close as possible to the original factual sample $\boldsymbol{x}$. We can formalize the role of the counterfactual generator g by casting it as the problem of solving the following objective:

$$\boldsymbol{x}' = \arg\min_{\boldsymbol{x}^*} \; d(\boldsymbol{x}, \boldsymbol{x}^*) \quad \text{s.t.: } \boldsymbol{x} \neq \boldsymbol{x}^* \wedge f(\boldsymbol{x}) \neq f(\boldsymbol{x}^*).$$

Hereinafter, we assume to have an oracle $f_{\boldsymbol{\theta}}$ for a binary classification task; the arguments generalize with minor modifications to multiclass classification and regression as well. Below, we formally define the Counterfactual Narrative Generation Problem.

Definition 4.3.2 (The Counterfactual Narrative Generation Problem)**.** *Given a generic pair of factual-counterfactual samples $(\boldsymbol{x}, \boldsymbol{x}')$ where the counterfactual sample $\boldsymbol{x}'$ is a solution of the Counterfactual Explanation Problem, a generic function H, and a prompt p we want to generate a natural language narrative ε associated with the generic factual-counterfactual pair $(\boldsymbol{x}, \boldsymbol{x}')$, namely, $\varepsilon = H(p, \boldsymbol{x}, \boldsymbol{x}')$ also known as counterfactual narrative.*

The function H is defined generically to accommodate different instantiations, including deterministic mappings, probabilistic generative models, or ensembles thereof.

4.3.3 Proposed Pipeline

In order to solve the problem in Definition 4.3.2, we propose a new pipeline (see Figure 4.7, Algorithm 12) to generate coherent and useful narratives for factual-counterfactual pairs. Our approach enhances the capabilities of Small Language Models (SLMs) through a two-stage pipeline called

Multi-Narrative Refinement (MNR), which encourages structured self-correction. The entire framework is fine-tuned using knowledge distillation, allowing us to leverage the expertise of a much larger teacher models.

In the first stage, a Draft Narrative Generator (DNG) $\mathcal{M}$ is tasked with generating multiple candidate draft explanations. Given a factual instance $\boldsymbol{x}$, its counterfactual $\boldsymbol{x}'$, and a structured prompt p encoding task-specific instructions, the DNG is queried independently $N = 3$ times to produce diverse draft explanations for the prediction shift from $\boldsymbol{x}$ to $\boldsymbol{x}'$. These drafts reflect different plausible reasoning paths the model might follow.

In the second stage, a different model called the Refiner $\mathcal{R}$ receives as input the same factual instance $\boldsymbol{x}$, counterfactual $\boldsymbol{x}'$, prompt p, and the three draft explanations. Its goal is to synthesize a coherent and accurate explanation by comparing, contrasting, and integrating the drafts. This additional refinement step promotes consistency and correctness, encouraging the model to resolve contradictions and highlight salient differences between $\boldsymbol{x}$ and $\boldsymbol{x}'$ in a principled manner. We introduce the refinement step because integrating it into the generation pipeline has been shown to enhance the performance and reliability of language models, particularly by fostering self-correction through iterative reasoning [159, 193, 222].

Algorithm 12 MNR pipeline pseudocode

Require: $\boldsymbol{x}$: a factual example, $\boldsymbol{x}'$: a counterfactual example, p_{draft}: the draft prompt, $p_{refiner}$: the refiner prompt
Ensure: ε: a narrative ε for (x, x')

```
p_draft ← format_prompt(p_draft, x, x')
i ← 0
E ← ∅ while i < 3 do
    ε^i_draft ← M(p_draft)
E ← E ∪ ε^i_draft
i ← i + 1

p_refiner ← format_prompt(p_refiner, x, x', E)
ε ← R(p_refiner)
```

Finetuning Using Knowledge Distillation In order to be more efficient and accurate, our pipeline uses Knowledge Distillation (KD), a training paradigm where a compact student model (an SLM) is trained to replicate the behavior of a powerful teacher model (T). This allows us to transfer the powerful reasoning and generation abilities of a large, state-of-the-art model to our much smaller, more efficient SLMs. Since the generation process is split into two different stages, each one addressing a different task, we use KD differently for each one of the stages.

Concerning the draft generation (stage 1), the KD process begins with creating a high-quality training dataset (See Figure 4.6). Given a counterfactual explainer $g(\cdot)$, consider a set of n instances $\{\boldsymbol{x}_i\}_{i=1}^n$. For each of these instances, we generate its corresponding optimal counterfactual, i.e., $\boldsymbol{x}'_i = g(\boldsymbol{x}_i)$, thereby obtaining n factual-counterfactual pairs $\{(\boldsymbol{x}_i, \boldsymbol{x}'_i)\}_{i=1}^n$. These pairs are embedded into n different structured prompts $p_1, p_2 \ldots, p_n$ and then fed to the teacher model T (for further details on the prompts see Figure 4.9). The high-quality outputs from T are then used to build a dataset $D_{\mathcal{M}}$ to finetune the DNG.

The KD process is also applied to the refiner $\mathcal{R}$. Here, we leverage the teacher model T to construct a training dataset $D_{\mathcal{R}}$ including not only the factual and counterfactual inputs and prompt, but also a corresponding set of $N = 3$ draft explanations. This enriched supervision allows the student model to learn from the implicit variation, redundancy, and conflict present across multiple generated rationales. Further details on the practical implementation of the KD process can be found in Section 4.3.4.

Quantitative Evaluation Metrics for Factual-Counterfactual Narratives

To quantitatively assess the narratives produced by our MNR pipeline, we developed a structured evaluation framework that allows us to measure the accuracy of each narrative. Our evaluation approach, inspired by the methodology in [73], requires the model to structure its output. Specifically, the prompt p instructs the SLM to embed its explanation within a structured dictionary format (see Figure 4.9). After the refiner model generates the final narrative, we parse this dictionary to extract key information about the feature changes between the factual instance $\boldsymbol{x}$ and its counterfactual $\boldsymbol{x}'$. The core assumption is that a model's ability to populate this structured format correctly reflects its underlying understanding of the narrative. Therefore, errors in the dictionary signal potential inaccuracies in the subsequent natural language narrative. This technique offers a reliable and automated method for evaluating the factual correctness of the generated text. The following metrics quantify how accurately the refined narratives align with the ground truth factual and counterfactual feature changes.

Given four feature vectors $\mathbf{F}_{fact}$, $\mathbf{F}_{cf}$, $\widetilde{\mathbf{F}}_{fact}$, $\widetilde{\mathbf{F}}_{cf}$ containing respectively, the factual ground truth, the counterfactual ground truth, the factual values, and the counterfactual values generated by the MNR pipeline, we can define the following metrics:

Average Feature Faithfulness (AvgFF) Given the set of all the narratives generated E, and all the vectors above for each narrative ε, we can compute the fraction of correctly interpreted features as:

$$\text{AvgFF} = \frac{\sum_{\varepsilon \in E} \sum_{i=0}^{k} \mathbb{I}\left[\left(\mathbf{F}^{\varepsilon}[i]_{fact} = \widetilde{\mathbf{F}}^{\varepsilon}[i]_{fact}\right) \wedge \left(\mathbf{F}^{\varepsilon}[i]_{cf} = \widetilde{\mathbf{F}}^{\varepsilon}[i]_{cf}\right)\right]}{|E| \cdot k}$$

where $k = |\mathbf{F}_{fact}|$ is the vector size, and $\mathbb{I}$ is the indicator function.

Perfect Feature Faithfulness (PFF) This metric measures the fraction of generated explanations that correctly identify all feature changes. Formally, given the set of all the narratives generated E, the *Perfect Feature Faithfulness* is calculated as:

$$\text{PFF} = \frac{\sum_{\varepsilon \in E} \mathbb{I}\left[\left(\mathbf{F}^{\varepsilon}_{fact} = \widetilde{\mathbf{F}}^{\varepsilon}_{fact}\right) \wedge \left(\mathbf{F}^{\varepsilon}_{cf} = \widetilde{\mathbf{F}}^{\varepsilon}_{cf}\right)\right]}{|E|}$$

This binary indicator is averaged over all narratives $\varepsilon \in E$, resulting in the proportion of narratives that precisely match all factual and counterfactual feature values.

Target Faithfulness (TF) This metric measures the ability of the function H to "understand" the factual and counterfactual dependent variables and compare them against the provided ground

truth. Let E be the set of all the narratives generated by our pipeline, y_{fact}, y_{cf} the ground truth target outcomes, and let $\widetilde{y}_{fact}, \widetilde{y}_{cf}$ be the corresponding predicted outcomes by the pipeline. The Target Faithfulness computed over all the narratives $\varepsilon \in E$ is defined as:

$$\text{TF} = \frac{\sum_{\varepsilon \in E} \mathbb{I}\left[\left(y^{\varepsilon}_{fact} = \widetilde{y}^{\varepsilon}_{fact}\right) \wedge \left(y^{\varepsilon}_{cf} = \widetilde{y}^{\varepsilon}_{cf}\right)\right]}{|E|}$$

These metrics provide a clear, quantitative assessment of how reliably the explanations reflect the ground truth provided by the factual and counterfactual scenarios.

Narrative Quality

The Narrative Quality dimension focuses on assessing explanations through human evaluation, emphasizing the readability, clarity, and coherence of the provided narrative. The questionnaire comprises 5 specific questions, each rated on a scale from 1 (poor) to 5 (excellent):

- Q1: Is the terminology and language used in the narrative appropriate and easy to understand?
- Q2: How clear and understandable is the provided narrative regarding feature changes?
- Q3: How clearly does the narrative describe the specific feature changes that led to the counterfactual outcome?
- Q4: Are the described feature changes easy to interpret, and do they make sense within the context of the original data?
- Q5: What is your overall assessment of the clarity and coherence of the narrative?

4.3.4 Experiments

Our experimental framework is detailed below, covering the hardware, datasets, models, and language models used in our evaluation. Experiments are conducted on a system equipped with an AMD Ryzen 9 7900 12-Core Processor, an NVIDIA GeForce RTX 4090 GPU, and 64 GB of RAM. We evaluate our approach on two benchmark datasets: the Adult dataset [21], for predicting whether an individual's income exceeds \$50K/year, and the Titanic dataset [65], for predicting passenger survival. The oracle model is a Decision Tree classifier from the `scikit-learn` library, configured with a `max_depth` of 4, a `criterion` of `'gini'`, and `min_samples_split` set to 2. This model achieves an accuracy of 0.84 on the Adult dataset and 0.82 on the Titanic dataset. Counterfactual explanations are generated using the DiCE library [143]. We utilize models from two distinct families for narrative generation: Qwen2.5 [184] and DeepSeek-r1 [52]. The specific hyperparameters used for response generation are detailed in Table 4.3.

Knowledge Distillation and Fine-tuning

Since our pipeline uses Knowledge Distillation to improve model performance, we built four new datasets (two for each dataset we tested) tailored explicitly for our tasks (draft narrative generation and narrative refinement). The draft narrative generation dataset consists of a JSON file containing prompt–response pairs, where each prompt is designed for the explanation task and the

Table 4.3: Hyperparameter Settings for all the Large Language Models used, namely, Temperature, Top-k, Top-p, Max Tokens, Repetition Penalty

Model Name	T.	Top-k	Top-p	M. T.	R. P.
Qwen2.5-0.5B-Instruct	0.6	10	0.8	8192	1.05
Qwen2.5-3B-Instruct	0.6	10	0.8	8192	1.05
DeepSeek-R1-D.-Qwen-1.5B	0.6	10	0.7	8192	1.05
DeepSeek-R1-D.-Qwen-7B	0.6	10	0.7	8192	1.05
DeepSeek-R1-D.-Qwen-32B	0.7	10	0.7	8192	1.05

corresponding response is generated by the `DeepSeek-R1-Distill-Qwen-32B` teacher model. We choose the `DeepSeek-R1-Distill-Qwen-32B` model due to its strong initial performance in generating coherent and contextually accurate text, especially suited for complex reasoning tasks. For the narrative refinement dataset, we employ fine-tuned `Qwen2.5-0.5B-Instruct` models as the Draft Narrative Generators, which are the smallest model considered in our evaluation. Our rationale is to construct a dataset using what can be seen as the hypothetically weakest drafter, thereby making the refinement task particularly challenging. In this way, the Refiner is forced to handle noisy, incomplete, or inconsistent drafts, which provides a more rigorous test of its ability to synthesize coherent explanations through self-correction and additional reasoning. Also in this case, the refiner model that merges the drafts is `DeepSeek-R1-Distill-Qwen-32B`.

Leveraging this structured and targeted dataset, we expect our fine-tuned models to achieve significantly better accuracy and clarity in producing comprehensive and natural-language counterfactual explanations. Figure 4.6 illustrates the dataset generation process for both the draft narrative generation and the narrative refinement tasks. To perform the fine-tuning, we used the Unsloth [49] framework; all the parameters can be seen in Table 4.4.

Table 4.4: Finetuning parameters: Checkpoint: checkpoint used during evaluation, B.S.: Batch Size, T.E.: Training Epochs, M.S.: Max Steps, C.S.S.: Checkpoint Saving Steps

Model	Checkpoint	B.S.	T.E.	M.S.	C.S.S.
Qwen2.5-0.5B-Instruct	500	1	1	1562	50
Qwen2.5-3B-Instruct	500	1	1	1562	50
Deepseek-R1-D.-Qwen-1.5B	400	1	1	1562	50
Deepseek-R1-D.-Qwen-7B	1000	1	1	1562	50

4.3.5 Results

In this section, we present the results of our study along two different dimensions. First, we quantitatively and qualitatively analyze the performance of MNR pipeline, assessing both accuracy and narrative quality. Next, we examine the feasibility of employing SLMs for narrative explanation tasks, focusing on efficiency and resource requirements. Finally, we provide an illustrative example to highlight the effectiveness and interpretability of the generated narratives.

Multi-Narrative Refinement Results.

The results presented in Table 4.6 provide compelling evidence for the efficacy of our pipeline, highlighting two primary conclusions: the necessity of fine-tuning and the significant performance boost provided by the refinement stage.

Across both datasets, the base models are incapable of performing the narrative generation task, with metrics for Feature Fidelity (FF) and Target Faithfulness (TF) consistently at or near zero. The impact of fine-tuning is dramatic and immediate. For instance, in the "No Refiner" baseline for the Adult dataset, the `DeepSeek-R1-Distill-Qwen-1.5B` Draft Explainer sees its Perfect FF score jump from 0.0 to 0.946 after fine-tuning. This pattern confirms that distillation is essential to impart the required capabilities to the models.

The value of the two-stage refinement process is most evident when examining the interplay between models. A fine-tuned Draft Explainer alone sets a strong baseline, but adding a capable refiner elevates performance. This is particularly striking when a weaker drafter is used. For example, the `Qwen2.5-0.5B-Instruct` fine-tuned drafter only achieves a Perfect FF of 0.485, but when its outputs are processed by the `Qwen2.5-3B-Instruct` refiner, the score more than doubles (0.980).

The `Qwen2.5-3B-Instruct` model emerges as the best refiner. Across nearly all configurations on the Adult dataset, it delivers the highest scores, pushing both Perfect FF and TF metrics above 0.97 and achieving near-zero standard deviation, indicating highly consistent and accurate outputs.

On the Titanic dataset, the best configurations reach a Perfect FF of 0.790, and the standard deviation is relatively high (often >0.40). This suggests that while our pipeline is robust, its performance limit could be influenced by the inherent difficulty of the dataset or by the model's hyperparameters.

In Table 4.6, we also introduced the performance obtained using the teacher model to directly generate the narratives. Results show that our MNR pipeline can perform better than the teacher model used to generate the dataset for the model fine-tuning.

Since the combination draft generator $\mathcal{M}$ Qwen2.5-0.5B-Instruct and refiner $\mathcal{R}$ Qwen2.5-3B-Instruct have the best performance, from now on, we will use these models in our pipeline.

Table 4.5: Narrative Quality Evaluation Scores. Our solution (MNR pipeline) has as a draft generator $\mathcal{M}$ Qwen2.5-0.5B-I. and as refiner $\mathcal{R}$ Qwen2.5-3B-I. The model DeepSeek-R1-D.-Qwen-32B is the teacher model we used to build the fine-tuning dataset, Qwen2.5-0.5B-I., instead, is the fine-tuned model without refiner.

Model	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**
DeepSeek-R1-D.-Qwen-32B	4.5 ± 0.6	4.4 ± 0.6	4.3 ± 0.6	4.2 ± 1.0	4.4 ± 0.6
Qwen2.5-0.5B-I.	3.2 ± 0.9	3.1 ± 1.2	3.1 ± 1.1	2.9 ± 1.0	3.0 ± 0.7
MNR pipeline	4.4 ± 0.7	4.6 ± 0.5	4.0 ± 0.9	4.3 ± 0.6	4.4 ± 0.5

Qualitative Results

The survey has been conducted on a sample of 15 people with different backgrounds, ages, and genders. The results of the human evaluation in Table 4.5 highlight the significant impact of our pipeline on the narrative quality. The teacher model (DeepSeek-R1-D.-Qwen-32B) gets high scores across all five of the evaluation criteria. Notably, our approach (MNR pipeline) achieves comparably high scores in all categories, with Q4 (interpretability and contextual coherence) receiving the highest rating of 4.3 (slightly higher than the teacher). This suggests that the narratives generated by the MNR pipeline are as clear as the one generated by the 32B model used as teacher. Similarly, Q2 and Q3, which assess the clarity and specificity of feature changes, exhibit strong performance (4.6 and 4.0, respectively), indicating that our solution effectively conveys how and why changes lead to a counterfactual outcome.

Table 4.6: Performance comparison of base versus fine-tuned models using the Multi-Narrative Refinement (MNR) pipeline. The first row for each dataset shows the performance of the teacher model solving the counterfactual narrative generation problem by itself. The rows with No Refiner serve as a baseline to get the model performance without a refiner attached. The results shown here evaluate the impact of fine-tuning the Draft Explainer model directly. If the Refiner is indicated, the rows evaluate the performance of the Refiner model. In these cases, the comparison is between a plain and a fine-tuned Refiner, both of which use drafts generated by an already fine-tuned Draft Explainer.

Draft Model ($\mathcal{M}$)	**Refiner Model ($\mathcal{R}$)**	**AvgFF ± std ↑**		**PFF ↑**		**TF ↑**	
		Base	**Tuned**	**Base**	**Tuned**	**Base**	**Tuned**
Adult Dataset							
DeepSeek-Q-32B (*Teacher*)		0.945±0.1		0.865		0.980	
Qwen2.5-0.5B-I	*No Refiner*	0.0 ± n.d.	0.485 ± 0.24	0.0	0.485	0.0	0.515
	Qwen2.5-0.5B-I	0.012 ± 0.19	0.796 ± 0.12	0.005	0.790	0.0	0.810
	DeepSeek-Q-1.5B	0.0 ± n.d.	0.868 ± 0.1	0.0	0.855	0.150	0.750
	Qwen2.5-3B-I	0.122 ± 0.1	**0.980 ± 0.0**	0.120	**0.980**	0.125	**0.980**
	DeepSeek-Q-7B	0.048 ± 0.44	0.854 ± 0.24	0.035	0.755	0.120	0.965
DeepSeek-Q-1.5B	*No Refiner*	0.0 ± n.d.	0.946 ± 0.03	0.0	0.945	0.135	0.410
	Qwen2.5-0.5B-I	0.005 ± n.d.	0.892 ± 0.08	0.005	0.890	0.0	0.900
	DeepSeek-Q-1.5B	0.0 ± n.d.	0.902 ± 0.1	0.0	0.885	0.120	0.785
	Qwen2.5-3B-I	0.085 ± 0.0	**0.976 ± 0.04**	0.085	**0.970**	0.085	**0.980**
	DeepSeek-Q-7B	0.030 ± 0.51	0.823 ± 0.26	0.030	0.700	0.070	0.955
Qwen2.5-3B-I	*No Refiner*	0.053 ± 0.1	0.988 ± 0.0	0.050	0.988	0.055	0.988
	Qwen2.5-0.5B-I	0.007 ± 0.42	0.908 ± 0.02	0.005	0.905	0.005	0.910
	DeepSeek-Q-1.5B	0.005 ± 0.13	0.922 ± 0.08	0.005	0.905	0.090	0.795
	Qwen2.5-3B-I	0.085 ± 0.0	**0.995 ± 0.0**	0.085	**0.995**	0.085	**0.995**
	DeepSeek-Q-7B	0.038 ± 0.48	0.840 ± 0.24	0.035	0.750	0.105	0.945
DeepSeek-Q-7B	*No Refiner*	0.251 ± 0.47	0.855 ± 0.11	0.230	0.855	0.405	0.865
	Qwen2.5-0.5B-I	0.005 ± n.d.	0.873 ± 0.1	0.005	0.865	0.0	0.885
	DeepSeek-Q-1.5B	0.0 ± n.d.	0.904 ± 0.05	0.0	0.895	0.106	0.770
	Qwen2.5-3B-I	0.080 ± 0.0	**0.973 ± 0.04**	0.080	**0.970**	0.080	0.975
	DeepSeek-Q-7B	0.035 ± 0.47	0.843 ± 0.26	0.030	0.715	0.090	**0.980**
Titanic Dataset							
DeepSeek-Q-32B (*Teacher*)		0.608 ± 0.47		0.590		0.945	
Qwen2.5-0.5B-I	*No Refiner*	0.0 ± n.d.	**0.790 ± 0.16**	0.0	0.740	0.0	0.840
	Qwen2.5-0.5B-I	0.052 ± 0.4	0.251 ± 0.49	0.040	0.243	0.015	0.592
	DeepSeek-Q-1.5B	0.0 ± n.d.	0.522 ± 0.47	0.0	0.510	0.050	0.805
	Qwen2.5-3B-I	0.040 ± 0.0	0.782 ± 0.4	0.040	**0.780**	0.040	**0.985**
	DeepSeek-Q-7B	0.080 ± 0.48	0.562 ± 0.49	0.075	0.550	0.210	0.960
DeepSeek-Q-1.5B	*No Refiner*	0.003 ± 0.15	0.475 ± 0.49	0.0	0.470	0.035	0.095
	Qwen2.5-0.5B-I	0.052 ± 0.38	0.350 ± 0.49	0.045	0.340	0.0	0.660
	DeepSeek-Q-1.5B	0.0 ± n.d.	0.507 ± 0.48	0.0	0.500	0.030	0.790
	Qwen2.5-3B-I	0.050 ± 0.0	**0.790 ± 0.4**	0.050	**0.785**	0.050	**0.995**
	DeepSeek-Q-7B	0.117 ± 0.49	0.568 ± 0.49	0.110	0.555	0.240	0.955
Qwen2.5-3B-I	*No Refiner*	0.0 ± n.d.	0.635 ± 0.46	0.0	0.625	0.005	0.920
	Qwen2.5-0.5B-I	0.040 ± 0.34	0.383 ± 0.49	0.030	0.365	0.010	0.715
	DeepSeek-Q-1.5B	0.005 ± 0.22	0.507 ± 0.48	0.005	0.495	0.055	0.840
	Qwen2.5-3B-I	0.065 ± 0.0	**0.738 ± 0.44**	0.065	**0.737**	0.065	**0.987**
	DeepSeek-Q-7B	0.102 ± 0.49	0.578 ± 0.49	0.095	0.565	0.210	0.975
DeepSeek-Q-7B	*No Refiner*	0.287 ± 0.36	0.385 ± 0.49	0.280	0.280	0.345	0.490
	Qwen2.5-0.5B-I	0.033 ± 0.37	0.417 ± 0.49	0.030	0.410	0.005	0.720
	DeepSeek-Q-1.5B	0.010 ± 0.39	0.522 ± 0.47	0.010	0.515	0.035	0.805
	Qwen2.5-3B-I	0.050 ± 0.0	**0.777 ± 0.41**	0.050	**0.775**	0.050	**0.990**
	DeepSeek-Q-7B	0.133 ± 0.48	0.583 ± 0.49	0.120	0.570	0.245	**0.990**

In contrast, the model without refiner scores significantly lower, particularly in Q4 and Q5, with ratings of 2.9 and 3.0, respectively. These results suggest that without the refinement step, the model struggles to produce clear and coherent narratives, making them harder to interpret and understand.

Feasibility

Beyond accuracy and quality, the feasibility of deploying counterfactual narrative generators critically depends on their computational requirements. In our study, we compare three settings: a teacher LLM, a worker SLM without refinement, and a Multi-Narrative Refinement pipeline composed of two SLMs, both finetuned. To assess their practical applicability, we analyze three key dimensions: (i) model size, (ii) inference time, and (iii) energy consumption during explanation generation.

Model size directly affects deployability. Large LLMs demand tens of gigabytes of memory, whereas distilled SLMs are an order of magnitude smaller, enabling deployment on commodity hardware. In the case of the MNR pipeline, model size reflects the combined size of both SLMs. Regarding inference time, the MNR pipeline's reported latency corresponds to the full generation process of a final explanation, which includes producing three independent draft narratives followed by a refinement step. By contrast, the SLM without refinement and the teacher model T each generate only a single draft. For energy consumption, we measure the total GPU energy required to generate a single explanation. GPU power draw was sampled every 200 milliseconds, yielding a fine-grained trace of instantaneous consumption. The total energy was then computed by integrating power measurements over time. Specifically, given the set of samples $S = \{(t_0, P_0), (t_1, P_1), \ldots, (t_n, P_n)\}$, where t_i denotes the timestamp of sample i and P_i the corresponding power (in Watts), the energy consumption is approximated as

$$E_{total} \approx \sum_{i=1}^{n} \frac{P_{i-1} + P_i}{2} \cdot (t_i - t_{i-1}).$$

This calculation yields the total energy expenditure (in Joules) for generating a single explanation under each configuration. Table 4.7 reports the results for the Adult dataset, comparing the LLM, the worker SLM without refiner, and the MNR pipeline. Our findings show that the MNR approach reduces memory usage by a factor of 2.7 compared to the large model (`DeepSeek-R1-D.-Qwen-32B`), while also lowering inference time by nearly 50% and energy consumption by 62% (see Fig. 4.8 for the power consumption trends over time). Although the standalone SLM achieves even lower memory usage, faster inference, and reduced energy consumption, its performance is consistently halved across all evaluation metrics.

Table 4.7: Feasibility results for explanation generation across different pipelines for the Adult dataset. The first row reports the performance of the teacher T, the second corresponds to the worker SLM without refinement, and the third to the MNR pipeline composed of two fine-tuned SLMs, a draft generator $\mathcal{M}$ Qwen2.5-0.5B-I. and a refiner $\mathcal{R}$ Qwen2.5-3B-I.

Model Name	Size (GB)	Time (s)	Energy (J)
DeepSeek-R1-D.-Qwen-32B	18.16	20.5 ± 4.1	7613.6 ± 1536.5
Qwen2.5-0.5B-I.	0.92	1.23 ± 0.29	215.5 ± 53.3
MNR pipeline	6.71	12.33 ± 2.0	2867.2 ± 548.4

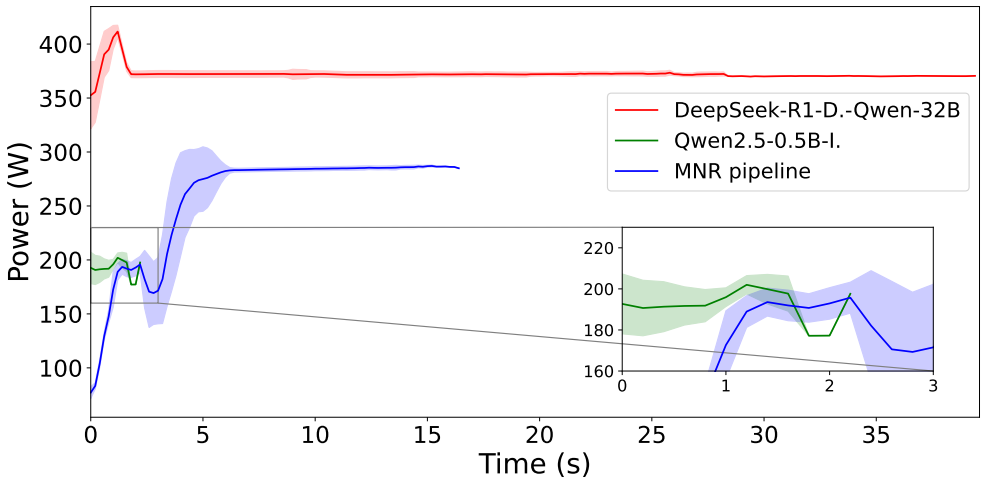


Figure 4.8: GPU power comparison of three techniques for narrative generation. Our approach (MNR pipeline) uses Qwen2.5-0.5B-I. as draft generator $\mathcal{M}$ and Qwen2.5-3B-I. as refiner $\mathcal{R}$. DeepSeek-R1-D.-Qwen-32B is the teacher model, while Qwen2.5-0.5B-I is the fine-tuned model without refiner. Solid lines show average power; shaded areas denote standard deviation.

Narrative Example

Table 4.8 illustrates the effectiveness of the MNR pipeline by comparing the raw drafts generated by the Draft Narrative Generator $\mathcal{M}$ with the merging phase produced by the Refiner $\mathcal{R}$.

The three drafts provide diverse but imperfect accounts: for instance, Draft 1 correctly identifies occupation and marital status as influential, yet introduces minor ambiguities by overstating the role of race; Draft 2 emphasizes the importance of the first two feature changes, and also speculates that race change does not have a direct impact on prediction; Draft 3, instead, misattributes relevance to unchanged factors such as age, thereby introducing noise into the explanation. Left on their own, each draft offers only a partial and potentially misleading picture of the prediction shift.

The refinement stage addresses these limitations by systematically comparing the drafts, resolving contradictions, and merging complementary insights. In the merging phase shown in the table, the Refiner explicitly discards Draft 3's error ("Draft 3 seems incorrect because age didn't change") and aligns the narrative with the factual–counterfactual pair. At the same time, it strengthens the explanation by integrating overlapping claims from Drafts 1 and 2: namely, that occupation is the primary driver of the income change, while marital status contributes additional financial stability. The Refiner also goes beyond simple aggregation, providing deeper reasoning on feature interactions — for example, noting that being married and working in a white-collar job together exert a stronger effect than either factor alone, or that the impact of the change in race from 'White'to 'Other' is uncertain or secondary.

This process exemplifies the self-correcting nature of MNR: by leveraging multiple imperfect narratives, the pipeline constructs a coherent, accurate, and more nuanced explanation than any single draft could provide. As a result, the final narrative not only corrects factual inconsistencies but also enriches the reasoning, highlighting feature interactions and causal plausibility in a way that enhances both fidelity and user interpretability. For other examples, please refer to the code `https://github.com/flaat/llm_kd`.

p_{draft}

```
A counterfactual explanation refers to a type of explanation in machine learning
and artificial intelligence that describes how altering certain input features can change the
output of a model. It answers 'what if' scenarios by identifying minimal changes necessary
to achieve a different desired outcome. Counterfactual explanations provide
insights into the decision-making process of complex models, enhancing
transparency and interpretability. For example, consider a credit scoring model that
denies a loan application. A counterfactual explanation might be: 'If your annual income
had been $50,000 instead of $45,000, your loan would have been approved.' This helps
the
applicant understand what specific change could lead to a different decision. Your task is
to generate a comprehensive, natural language counterfactual explanation of the
classification change when transitioning from a factual example. Given the following
inputs:
- Dataset Description: Background knowledge about the dataset, including
feature definitions, their significance, and statistical distributions.
- Factual Example: A specific instance from the dataset that was classified under
the original conditions.
- Counterfactual Example: A modified version of the factual example where
certain features have been altered, resulting in a different classification.
The explanation should:
1. Identify Feature Changes: List and describe the features that differ between the factual
and counterfactual examples. You should follow the structure outlined below.
2. Reasoning: Carry out a reasoning step that is functional to generating the
final summary, in particular:
- Analyze Contribution of Features: Assess the influence of each changed feature on the
classification outcome, leveraging dataset knowledge to justify its impact.
- Highlight Interactions: Discuss any interactions between features that may have played
a role in shifting the classification outcome.
- Summarize Key Factors: Conclude with a concise summary of the most
influential features and their role in altering the prediction.
Provide your responses in complete sentences and paragraphs, explaining concepts
clearly and concisely in a continuous flow. The summary should be clear, coherent, and
provide an intuitive understanding of how the model's decision was influenced by the
observed
feature modifications. Your output should follow the following JSON structure:
{{ "feature_changes": [
        {{"<FEATURE_1>": {{"factual": "<FACTUAL_VALUE_1>",
        "counterfactual": "<COUNTERFACTUAL_VALUE_1>"}}}}, ...
        {{"<FEATURE_N>": {{"factual": "<FACTUAL_VALUE_N>",
        "counterfactual": "<COUNTERFACTUAL_VALUE_N>"}}}}, ],
"target_variable_change": {{"factual": "<FACTUAL_TARGET>",
"counterfactual": "<COUNTERFACTUAL_TARGET>"}},
"reasoning": "<YOUR_REASONING>",
"explanation": "<YOUR_SUMMARY> }}
Please remeber to include also the target variable in the feature_changes list.
Please avoid any further explanation, clarification or other unnecessary outputs,
just provide the JSON. Here is your input: ### Dataset Description ###
{dataset_description} ### Factual Example ### {factual_example} ###
Counterfactual Example ### {counterfactual_example} Avoid using bullet points, lists, or
numerical outlines.
```

$p_{refiner}$

```
A counterfactual explanation refers to a type of explanation in machine learning and artificial
intelligence that describes how altering certain input features can change the output of a
model. It answers 'what if' scenarios by identifying minimal changes necessary to achieve a
different desired outcome. Counterfactual explanations provide insights into the decision-
making process of complex models, enhancing transparency and interpretability.
For example, consider a credit scoring model that denies a loan application. A counterfactual
explanation might be: 'If your annual income had been $50,000 instead of $45,000, your loan
would have been approved.' This helps the applicant understand what specific change could
lead to a different decision. Your task is to generate a comprehensive, natural language
counterfactual explanation of the classification change when transitioning from a factual
example to its counterfactual counterpart. Given the following inputs:
- Dataset Description: Background knowledge about the dataset, including feature
definitions, their significance, and statistical distributions.
- Factual Example: A specific instance from the dataset that was classified under the original
conditions.
- Counterfactual Example: A modified version of the factual example where certain features
have been altered, resulting in a different classification.
- Draft Explanations: Three independent draft explanations of the same
factual/counterfactual pair. These explanations may overlap, complement, or partially
contradict each other. The explanation should:
1. Identify Feature Changes: List and describe the features that differ between the factual
and counterfactual examples. You should follow the structure outlined below.
2. Reasoning: Carry out a reasoning step that is functional to generating the final summary,
in particular:
   - Analyze Contribution of Features: Assess the influence of each changed feature on the
classification outcome, leveraging dataset knowledge to justify its impact.
   - Highlight Interactions: Discuss any interactions between features that may have played a
role in shifting the classification outcome.
3. Integrate Draft Explanations: Carefully review the three draft explanations. Extract the
core claims and evidence presented in each. In particular:
  - Where explanations conflict or differ, resolve these contradictions by prioritizing
statements best supported by the dataset description and the identified feature changes.
  - Merge complementary insights to create a unified, logically consistent explanation,
avoiding redundancy and ensuring the final narrative flows coherently.
4. Summarize Key Factors: Conclude with a concise summary of the most influential
features and their role in altering the prediction. The summary should be approximately 250
words. Avoid using bullet points, lists, or numerical outlines. Provide your responses in
complete sentences and paragraphs, explaining concepts clearly and concisely in a
continuous flow. The summary should be clear, coherent, and provide an intuitive
understanding of how the model's decision was influenced by the observed feature
modifications. Your output should follow the following JSON structure:
{{ "feature_changes": [
      {{"<FEATURE_1>": {{"factual": "<FACTUAL_VALUE_1>",
                "counterfactual": "<COUNTERFACTUAL_VALUE_1>"}}}}, ...
      {{"<FEATURE_N>": {{"factual": "<FACTUAL_VALUE_N>", "counterfactual": "
<COUNTERFACTUAL_VALUE_N>"}}}},
   ],
   "target_variable_change": {{"factual": "<FACTUAL_TARGET>", "counterfactual": "
<COUNTERFACTUAL_TARGET>"}},
   "reasoning": "<YOUR_REASONING>",
   "explanation": "<YOUR_SUMMARY>"
}}
Please remeber to include also the target variable in the feature_changes list.
Please avoid any further explanation, clarification or other unnecessary outputs, just provide
the JSON. Here is your input: ### Dataset Description ### {dataset_description} ### Factual
Example ###
{factual_example} ### Counterfactual Example ### {counterfactual_example} ### Draft
Explanation 1 ### {draft_explanation_1} ### Draft Explanation 2
### {draft_explanation_2} ### Draft Explanation 3 ### {draft_explanation_3}
```

Figure 4.9: Right side: prompt used to refine multile narratives. **Left side:** prompt to generate the draft explanations.

4.3.6 Conclusion

In this work, we introduced the Multi-Narrative Refinement pipeline, a novel approach to generating human-readable counterfactual narratives from tabular data. By combining knowledge distillation with a structured two-stage draft-and-refine process, we demonstrated that Small Language Models can be empowered to deliver coherent and contextually accurate narratives that demystify the decisions for complex AI models.

Our experiments yield several key insights. First, fine-tuning via knowledge distillation is not merely advantageous but indispensable: base SLMs without distillation fail to perform the task, whereas distilled versions consistently achieve high feature fidelity and narrative coherence. Second, the refinement stage consistently amplifies performance by resolving contradictions between drafts, correcting errors, and synthesizing more robust explanations.

Taken together, these results highlight the dual importance of distillation and multi-narrative refinement. The MNR pipeline not only advances the state of the art in counterfactual narrative generation but also demonstrates a practical path toward deploying smaller, resource-efficient models that preserve both reasoning depth and explanatory quality.

4.3.7 Social Impact

The research presented in this paper carries relevant social implications by addressing the need for transparency and interpretability in AI systems. The positive impact of our work is the democrati-

zation of AI explainability. By converting technical counterfactuals into clear and accessible natural language narratives, our framework empowers individuals who are not AI experts to understand the reasoning behind automated decisions that directly affect their lives. This fosters greater trust in AI systems and provides a basis for users to contest unfair or erroneous outcomes. Furthermore, in high-stakes domains like finance and healthcare, our method can help organizations meet regulatory requirements for transparency, such as those outlined in the EU AI Act (see Section 3.4). Furthermore, by enabling the use of smaller and more efficient models, our work also contributes to a more sustainable and economically viable approach to deploying explainable AI (see Section 4.3.5), making these powerful tools available to a wider range of organizations. Despite its benefits, this technology is not without risks. For example, the generated narratives could be highly persuasive but subtly inaccurate and could mask underlying model flaws or biases [22, 37, 86]. The process of simplification inherent in creating a narrative might also omit crucial details, creating a gap between the explanation's perceived meaning and the model's true internal logic.

Table 4.8: Comparison between the three draft explanations and the Refiner's reasoning phase for the Adult dataset.

Example	
Factual: age: 33; workclass: Private; education: School; marital status: Divorced; occupation: Blue-Collar; race: White; gender: Male; hours per week: 35 **Counterfactual**: age: 33; workclass: Private; education: School; marital status: Married; occupation: White-Collar; race: Other; gender: Male; hours per week: 35	
DNG ($\mathcal{M}$): **Qwen 2.5 0.5B**	Refiner ($\mathcal{R}$): **Qwen 2.5 3B**
Draft 1: The most influential features contributing to the change in income prediction are marital status, occupation, and race. Being married, which often signifies stability, likely enhances earning potential. Changing occupation to white-collar suggests a higher-paying job, which directly impacts income. Lastly, race may play a role in the model's perception of economic status, possibly reflecting broader societal factors. Together, these changes shift the model's assessment. **Draft 2**: The most influential factor in altering the prediction is the occupation change from 'Blue-Collar' to 'White-Collar'. This shift suggests a move to a higher-paying, professional role, which is strongly correlated with higher income. Additionally, the marital status change to 'Married' may imply financial stability or dual income, contributing to the higher income prediction. The race change might reflect broader economic disparities, though this effect is less direct. **Draft 3**: The increase in income classification from 'Income <=50K' to 'Income >50K' is primarily driven by three factors: older age, married status, and a white-collar job. These changes suggest that individuals in these categories are more likely to earn above 50K	**Merging Phase:** Okay, so I need to help the user by generating a comprehensive counterfactual explanation based on the given inputs. Let me start by understanding the problem. First, the dataset is the Adult dataset, which predicts income above or below 50K. The factual example has an income of 0 (<=50K), and the counterfactual has 1 (>50K). So, the goal is to explain why changing certain features caused this shift. Looking at the factual and counterfactual examples: Factual: - age: 33 - workclass: Private - education: School - marital_status: Divorced - occupation: Blue-Collar - race: White - gender: Male - hours-per-week: 35 Counterfactual: - age: 33 - workclass: Private - education: School - marital_status: Married - occupation: White-Collar - race: Other - gender: Male - hours-per-week: 35. So, the features that changed are: - marital_status: Divorced -> Married - occupation: Blue-Collar -> White-Collar - race: White -> Other. Wait, but looking back, the factual example's education is 'School', while the counterfactual's is still 'School'? Or did I misread? Oh no, wait, the factual example's education is 'School', and the counterfactual's is 'School' too. So those didn't change. Hmm, maybe I missed something. Let me check again. No, both have 'School' as education. So the only changes are marital_status, occupation, and race. Wait, in the factual example, the occupation is 'Blue-Collar', and in the counterfactual, it's 'White-Collar'. Also, race changes from 'White' to 'Other'. Now, looking at the draft explanations: Draft 1 says marital status, occupation, and race are the main factors. It mentions being married enhances earning potential, white-collar jobs indicate higher pay, and race reflects economic status. Draft 2 focuses mainly on occupation change to white-collar, suggesting higher income correlation. It also mentions marital status implying financial stability and possible dual income. Race might reflect economic disparities. Draft 3 talks about older age, married status, and white-collar job driving the income increase. But in our case, age remains 33, so that doesn't apply here. Hmm, so Draft 3 seems incorrect because age didn't change. So we can disregard that part. Now, analyzing the contributions: Occupation is a big one. Blue-Collar typically pays less than White-Collar. So changing to White-Collar likely increases income. Marital status: Married might correlate with higher income due to potential dual income or stability. Race: Changing from White to Other this is tricky. The dataset's race categories might have different income implications. Maybe 'Other' includes groups with higher average incomes, but it's not straightforward. But in the dataset description, race is a categorical variable, and the target variable is income. So perhaps 'Other' isn't necessarily worse; maybe it's neutral or positive in this context. Interactions: Maybe being married and having a White-Collar job together have a stronger effect than individually. Integrating the drafts: Draft 1 and 2 agree on occupation and marital status. Draft 2 adds marital status implying financial stability. Draft 3 incorrectly mentions age and doesn't address all changes. So, the key factors are occupation and marital status. Race's impact is uncertain but might contribute positively. Putting it all together, the explanation should highlight occupation as the primary driver, followed by marital status, and note the race change without overemphasizing it since age remained the same. Narrative: Occupation change from Blue-Collar to White-Collar likely increased income. Marital status from Divorced to Married might imply stability or dual income. Race change to Other might have a slight positive effect. The main factors are occupation and marital status. The race change is less impactful but still contributes. Wait, but in the factual example, the person is male, divorced, working blue-collar, white, and making <=50K. In the counterfactual, they're married, white-collar, male, and making >50K. So the key changes are occupation and marital status. So, the summary should focus on those two features, noting that occupation is a strong indicator of income level, and marital status can add to that effect. Race's role is secondary or unclear without more data. I think I've got it.

Chapter 5

Conclusions

In this thesis, we propose graphs as a unifying substrate for building intelligent networked systems; furthermore, we explore graph explainability methods, with a focus on practical deployments and transparent decision-making.

In Chapter 2 we study graphs in real-world systems, with an emphasis on constrained communication control. We formulate trajectory and task-planning problems that couple sensing, connectivity, and data offloading, and we develop optimization and heuristic solutions that respect platform limitations such as limited battery lifetime and link capacity. We further design planning and routing mechanisms that maintain periodic connectivity to a base station during exploration, enabling offloading of observations while sustaining coverage quality. Extensive simulation campaigns and testbed experiments show consistent gains over networking-only baselines in terms of task completion, freshness of the information delivered to the depot, and overall mission efficiency.

In Chapter 3, we turn to explainable AI for graphs. We introduce counterfactual explanation methods tailored to graph-structured data, encompassing structural edits, edge attribute changes, and node feature perturbations. We propose a unified generation pipeline with scheduling policies that strike a balance between fidelity and sparsity. We study runtime-quality trade-offs to make counterfactuals practical on standard benchmark datasets. To make explanations actionable for users, we complement the edits with natural-language narratives derived from the underlying counterfactual operations. We assess faithfulness and perceived usefulness through quantitative metrics and human evaluation.

In Chapter 4, we broadened the scope beyond graphs, investigating how counterfactual reasoning can be integrated into training to improve robustness and generalization. We present a counterfactual regularization strategy that encourages decision-level stability under realistic perturbations, demonstrating competitive or superior performance to common regularizers across a range of models and tabular datasets, while also enabling fast test-time counterfactual retrieval. We discussed limits on highly structured modalities and pointed to domain-aware generators as a promising avenue. We also present a new pipeline to generate counterfactual narratives in the domain of tabular data using pairs of small language models. Results show that our approach achieves better performance and is more energy-efficient than using a single large model.

Bibliography

[1] Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (general data protection regulation). Official Journal of the European Union L 119, 2016.

[2] Regulation (eu) 2024/1689 of the european parliament and of the council of 13 june 2024 laying down harmonised rules on artificial intelligence and amending certain union legislative acts (ai act). Official Journal of the European Union, 2024.

[3] M. Abdin, J. Aneja, H. Awadalla, A. Awadallah, A. A. Awan, N. Bach, A. Bahree, A. Bakhtiari, J. Bao, H. Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. arXiv preprint arXiv:2404.14219, 2024.

[4] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.

[5] F. Afghah, A. Razi, J. Chakareski, and J. Ashdown. Wildfire monitoring in remote areas using autonomous unmanned aerial vehicles. In IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pages 835–840. IEEE, 2019.

[6] R. Aggarwal, V. Sounderajah, G. Martin, D. S. Ting, A. Karthikesalingam, D. King, H. Ashrafian, and A. Darzi. Diagnostic accuracy of deep learning in medical imaging: a systematic review and meta-analysis. NPJ digital medicine, 4(1):65, 2021.

[7] F. Amigoni, J. Banfi, and N. Basilico. Multirobot exploration of communication-restricted environments: A survey. IEEE Intelligent Systems, 32(6):48–57, 2017.

[8] P. P. Angelov, E. A. Soares, R. Jiang, N. I. Arnold, and P. M. Atkinson. Explainable artificial intelligence: an analytical review. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 11(5):e1424, 2021.

[9] A. Bacciu, C. Campagnano, G. Trappolini, and F. Silvestri. DanteLLM: Let‘s push Italian LLM research forward! In N. Calzolari, M.-Y. Kan, V. Hoste, A. Lenci, S. Sakti, and N. Xue, editors, Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024), pages 4343–4355, Torino, Italia, May 2024. ELRA and ICCL.

[10] M. Bajaj, L. Chu, Z. Y. Xue, J. Pei, L. Wang, P. C.-H. Lam, and Y. Zhang. Robust counterfactual explanations on graph neural networks. Advances in Neural Information Processing Systems, 34:5644–5655, 2021.

[11] J. Banfi, A. Q. Li, N. Basilico, I. Rekleitis, and F. Amigoni. Asynchronous multirobot exploration under recurrent connectivity constraints. In 2016 IEEE International Conference on Robotics and Automation (ICRA), pages 5491–5498, 2016.

[12] J. Banfi, A. Q. Li, N. Basilico, I. Rekleitis, and F. Amigoni. Multirobot online construction of communication maps. In 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 2577–2583. IEEE, 2017.

[13] J. Banfi, A. Q. Li, I. Rekleitis, F. Amigoni, and N. Basilico. Strategies for coordinated multirobot exploration with recurrent connectivity constraints. Autonomous Robots, 42(4):875–894, 2018.

[14] N. Bartolini, A. Coletta, A. Gennaro, G. Maselli, and M. Prata. Mad for fanets: Movement assisted delivery for flying ad-hoc networks. In 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS), pages 315–325. IEEE, 2021.

[15] N. Bartolini, A. Coletta, F. Giorgi, G. Maselli, M. Prata, and D. Silvestri. Stop & offload: Periodic data offloading in uav networks. Computer Communications, 212:239–250, 2023.

[16] N. Bartolini, A. Coletta, F. Giorgi, G. Maselli, M. Prata, and D. Silvestri. Stop & route: Periodic data offloading in uav networks. In 2023 18th Wireless On-Demand Network Systems and Services Conference (WONS), pages 92–99. IEEE, 2023.

[17] N. Bartolini, A. Coletta, and G. Maselli. Side: Self driving drones embrace uncertainty. IEEE Transactions on Mobile Computing, 2021.

[18] N. Bartolini, A. Coletta, G. Maselli, et al. A multi-trip task assignment for early target inspection in squads of aerial drones. IEEE Transactions on Mobile Computing, 20(11):3099–3116, 2021.

[19] N. Bartolini, A. Coletta, G. Maselli, and M. Piva. Druber: a trustable decentralized drone-based delivery system. In Proceedings of the 6th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications, pages 1–6, 2020.

[20] N. Bartolini, A. Coletta, M. Prata, and C. Serino. On connected deployment of delay-critical fanets. In 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 9720–9727. IEEE, 2021.

[21] B. Becker and R. Kohavi. Adult. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

[22] L. Berti, F. Giorgi, and G. Kasneci. Emergent abilities in large language models: A survey. arXiv preprint arXiv:2503.05788, 2025.

[23] L. Bertizzolo, S. D'oro, L. Ferranti, L. Bonati, E. Demirors, Z. Guan, T. Melodia, and S. Pudlewski. Swarmcontrol: An automated distributed control framework for self-optimizing drone networks. In IEEE International Conference on Computer Communications (INFOCOM), pages 1768–1777. IEEE, 2020.

[24] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, others, and P. S. Liang. On the opportunities and risks of foundation models. arXiv preprint arXiv:2108.07258, 2021.

[25] A. C. Bovik. Handbook of image and video processing. Academic press, 2010.

[26] T. B. Brown, N. Carlini, C. Zhang, C. Olsson, P. Christiano, and I. Goodfellow. Unrestricted Adversarial Examples. arXiv preprint arXiv:1809.08352, 2018.

[27] R. Cai, Y. Zhu, X. Chen, Y. Fang, M. Wu, J. Qiao, and Z. Hao. On the probability of necessity and sufficiency of explaining graph neural networks: A lower bound optimization approach. Neural Networks, 184:107065, 2025.

[28] D. Callegaro, M. Levorato, and F. Restuccia. SeReMAS: Self-Resilient Mobile AutonomousSystems Through Predictive Edge Computing. arXiv preprint arXiv:2105.15105, 2021.

[29] C. Campagnano, S. Conia, and R. Navigli. SRL4E – Semantic Role Labeling for Emotions: A unified evaluation framework. In S. Muresan, P. Nakov, and A. Villavicencio, editors, Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 4586–4601, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[30] D. Cao, Y. Wang, J. Duan, C. Zhang, X. Zhu, C. Huang, Y. Tong, B. Xu, J. Bai, J. Tong, et al. Spectral temporal graph neural network for multivariate time-series forecasting. Advances in neural information processing systems, 33:17766–17778, 2020.

[31] R. Caruana, S. Lawrence, and C. Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In T. Leen, T. Dietterich, and V. Tresp, editors, Advances in Neural Information Processing Systems, volume 13, pages 381–387. MIT Press, 2000.

[32] G. Castellano, M. G. Miccoli, R. Scaringi, G. Vessio, G. Zaza, et al. Using llms to explain ai-generated art classification via grad-cam heatmaps. In Proceedings of 5th Italian Workshop on Explainable Artificial Intelligence, co-located with the 23rd International Conference of the Italian Association for Artificial Intelligence, Bolzano, Italy, 2024.

[33] M. Cedro and D. Martens. Tell me a story! narrative-driven xai with large language models, 2023.

[34] M. Cedro and D. Martens. Graphxain: Narratives to explain graph neural networks, 2024.

[35] H. Chefer, S. Gur, and L. Wolf. Transformer interpretability beyond attention visualization. arxiv, 2020.

[36] J. Chen, S. Wu, A. Gupta, and R. Ying. D4explainer: In-distribution gnn explanations via discrete denoising diffusion. arXiv preprint arXiv:2310.19321, 2023.

[37] W. Chen, Y. Su, J. Zuo, C. Yang, C. Yuan, C. Qian, C.-M. Chan, Y. Qin, Y. Lu, R. Xie, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. arXiv preprint arXiv:2308.10848, 2(4):6, 2023.

[38] W. Chen, Z. Su, Q. Xu, T. H. Luan, and R. Li. VFC-based cooperative UAV computation task offloading for post-disaster rescue. In IEEE International Conference on Computer Communications (INFOCOM), pages 228–236. IEEE, 2020.

[39] Y. Chen, N. Zhao, Z. Ding, and M.-S. Alouini. Multiple UAVs as relays: Multi-hop single link versus multiple dual-hop links. IEEE Transactions on Wireless Communications, 17(9):6348–6359, 2018.

[40] Z. Chen, J. Huang, F. Silvestri, Y. Zhang, H. Ahn, and G. Tolomei. Joint Factual and Counterfactual Explanations for Top-k GNN-based Recommendations. ACM Trans. Recomm. Syst., May 2025. Just Accepted.

[41] Z. Chen, F. Silvestri, G. Tolomei, J. Wang, H. Zhu, and H. Ahn. Explain the explainer: Interpreting model-agnostic counterfactual explanations of a deep reinforcement learning agent. IEEE Transactions on Artificial Intelligence, 5(4):1443–1457, 2022.

[42] Z. Chen, F. Silvestri, J. Wang, H. Zhu, H. Ahn, and G. Tolomei. Relax: Reinforcement learning agent explainer for arbitrary predictive models. In Proceedings of the 31st ACM international conference on information & knowledge management, pages 252–261, 2022.

[43] Z. Chen, F. Silvestri, J. Wang, H. Zhu, H. Ahn, and G. Tolomei. ReLAX: Reinforcement Learning Agent Explainer for Arbitrary Predictive Models. In Proceedings of the 31st ACM International Conference on Information & Knowledge Management, CIKM '22, page 252–261, New York, NY, USA, 2022. Association for Computing Machinery.

[44] F. Cheng, Y. Xu, Y. Li, Y. Yao, and Y. Song. Llms for generating and evaluating counterfactuals: A comprehensive study. arXiv preprint arXiv:2405.00722, May 2024.

[45] C.-H. Chiang and H.-y. Lee. Can large language models be an alternative to human evaluations? In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 15607–15631, Toronto, Canada, July 2023. Association for Computational Linguistics.

[46] S. Chuprov, L. Reznik, A. Obeid, and S. Shetty. How degrading network conditions influence machine learning end systems performance? In IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pages 1–6. IEEE, 2022.

[47] A. Coletta, F. Giorgi, G. Maselli, M. Prata, D. Silvestri, J. Ashdown, and F. Restuccia. A 2-uav: Application-aware content and network optimization of edge-assisted uav systems. In IEEE INFOCOM 2023-IEEE Conference on Computer Communications, pages 1–10. IEEE, 2023.

[48] A. Coletta, F. Giorgi, G. Maselli, M. Prata, D. Silvestri, J. Ashdown, and F. Restuccia. A 2-uav: Application-aware resilient edge-assisted uav networks. Computer Networks, 256:110887, 2025.

[49] M. H. Daniel Han and U. team. Unsloth, 2023.

[50] G. Danoy, M. R. Brust, and P. Bouvry. Connectivity stability in autonomous multi-level uav swarms for wide area monitoring. In Proceedings of the 5th ACM Symposium on Development and Analysis of Intelligent Vehicular Networks and Applications, pages 1–8, 2015.

[51] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. Computational Geometry: Algorithms and Applications. Springer, Berlin, Heidelberg, 3 edition, 2008.

[52] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[53] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. Advances in neural information processing systems, 29, 2016.

[54] B. Delaunay. Sur la sphère vide. Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk, 7:793–800, 1934.

[55] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. IEEE, 2009.

[56] T.-S. Dominique Van Cappel. Phoneme, 1993. `https://www.openml.org/search?type=data&sort=version&status=any&order=asc&exact_name=phoneme&id=1489`.

[57] F. Doshi-Velez and B. Kim. Towards a rigorous science of interpretable machine learning. arXiv preprint arXiv:1702.08608, 2017.

[58] F. K. Došilović, M. Brčić, and N. Hlupić. Explainable artificial intelligence: A survey. In 2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO), pages 0210–0215. IEEE, 2018.

[59] A. Dutta, A. Ghosh, and O. P. Kreidl. Multi-robot informative path planning with continuous connectivity constraints. In 2019 International Conference on Robotics and Automation (ICRA), pages 3245–3251. IEEE, 2019.

[60] European Commission. Proposal for a regulation of the european parliament and of the council laying down harmonised rules on artificial intelligence (Artificial Intelligence Act), 2021. COM/2021/206 final.

[61] European Parliament and Council. Regulation (eu) 2024/1689 of the european parliament and of the council on artificial intelligence (ai act), June 2024. Accessed: February 21, 2025.

[62] B. Fatemi, J. Halcrow, and B. Perozzi. Talk like a graph: Encoding graphs for large language models. In The Twelfth International Conference on Learning Representations, 2024.

[63] S. Fortune. A sweepline algorithm for voronoi diagrams. Algorithmica, 2:153–174, 1987.

[64] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur. Protein interface prediction using graph convolutional networks. Advances in neural information processing systems, 30, 2017.

[65] T. C. Frank E. Harrell Jr. Titanic dataset, oct 2017.

[66] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023.

[67] A. Fredes and J. Vitrià. Using llms for explaining sets of counterfactual examples to final users, 2024.

[68] T. Freiesleben. The Intriguing Relation Between Counterfactual Explanations and Adversarial Examples. Minds and Machines, 32(1):77–109, mar 2022.

[69] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-Adversarial Training of Neural Networks. Journal of Machine Learning Research, 17(1):2096–2030, jan 2016.

[70] Y. Gao, J. Fang, Y. Sui, Y. Li, X. Wang, H. Feng, and Y. Zhang. Graph anomaly detection with bi-level optimization. In Proceedings of the ACM Web Conference 2024, WWW '24, page 4383–4394, New York, NY, USA, 2024. Association for Computing Machinery.

[71] M. Garouani, J. Mothe, A. Barhrhouj, and J. Aligon. Investigating the duality of interpretability and explainability in machine learning. In 2024 IEEE 36th International Conference on Tools with Artificial Intelligence (ICTAI), pages 861–867. IEEE, 2024.

[72] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In International Conference on Machine Learning (ICML), pages 1263–1272, 2017.

[73] F. Giorgi, C. Campagnano, F. Silvestri, and G. Tolomei. Natural language counterfactual explanations for graphs using large language models. In International Conference on Artificial Intelligence and Statistics, pages 3565–3573. PMLR, 2025.

[74] F. Giorgi, F. Silvestri, and G. Tolomei. Generate counterfactual explanations for graph neural networks from node feature perturbations. TechRxiv preprint, 2024.

[75] F. Giorgi, F. Silvestri, and G. Tolomei. Combinex: A unified counterfactual explainer for graph neural networks via node feature and structural perturbations. arXiv preprint arXiv:2502.10111, 2025.

[76] F. Giorgi, M. Silvestri, C. Campagnano, F. Silvestri, and G. Tolomei. Enhancing xai narratives through multi-narrative refinement and knowledge distillation, 2025.

[77] F. Giorgi, F. Veglianti, F. Silvestri, and G. Tolomei. Generalizability through explainability: Countering overfitting with counterfactual examples. arXiv preprint arXiv:2502.09193, 2025.

[78] N. Goddemeier, K. Daniel, and C. Wietfeld. Role-based connectivity management with realistic air-to-ground channels for cooperative uavs. IEEE Journal on Selected Areas in Communications, 30(5):951–963, 2012.

[79] R. Guidotti. Counterfactual explanations and how to find them: literature review and benchmarking. Data Mining and Knowledge Discovery, pages 1–55, 2022.

[80] R. Guidotti, A. Monreale, F. Giannotti, D. Pedreschi, S. Ruggieri, and F. Turini. Factual and counterfactual explanations for black box decision making. IEEE Intelligent Systems, 34(6):14–23, 2019.

[81] R. Guidotti, A. Monreale, S. Ruggieri, D. Pedreschi, F. Turini, and F. Giannotti. Local Rule-Based Explanations of Black Box Decision Systems. arXiv preprint arXiv:1805.10820, 2018.

[82] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, D. Pedreschi, and F. Giannotti. A survey of methods for explaining black box models. ACM Computing Surveys, 51(5):93:1–93:42, 2018.

[83] S. Gunasekar, J. D. Lee, D. Soudry, and N. Srebro. Implicit Bias of Gradient Descent on Linear Convolutional Networks. In Proceedings of the 32nd International Conference on Neural Information Processing Systems, NeurIPS '18, pages 9482–9491, Red Hook, NY, USA, 2018. Curran Associates Inc.

[84] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948, 2025.

[85] L. Gurobi Optimization. Gurobi optimizer reference manual, 2019.

[86] T. Hagendorff. Deception abilities emerged in large language models. Proceedings of the National Academy of Sciences, 121(24):e2317967121, 2024.

[87] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In Advances in Neural Information Processing Systems 30 (NeurIPS), 2017.

[88] S. Hayat, E. Yanmaz, C. Bettstetter, and T. X. Brown. Multi-objective drone path planning for search and rescue with quality-of-service requirements. Autonomous Robots, 44(7):1183–1198, 2020.

[89] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[90] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, Computer Vision – ECCV 2016, pages 630–645, Cham, 2016. Springer International Publishing.

[91] Y. He, Z. Zheng, P. Soga, Y. Zhu, J. Li, et al. Explaining graph neural networks with large language models: A counterfactual perspective for molecular property prediction. arXiv preprint arXiv:2410.15165, 2024.

[92] Z. He, T. Zhang, and R. B. Lee. Model inversion attacks against collaborative inference. In Proceedings of the 35th Annual Computer Security Applications Conference, ACSAC '19, pages 148–162, New York, NY, USA, 2019. Association for Computing Machinery.

[93] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio. Learning Deep Representations by Mutual Information Estimation and Maximization. In 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net, 2019.

[94] G. A. Hollinger and S. Singh. Multirobot coordination with periodic connectivity: Theory and experiments. IEEE Transactions on Robotics, 28(4):967–973, 2012.

[95] S. Hosseinalipour, A. Rahmati, and H. Dai. Interference avoidance position planning in dual-hop and multi-hop UAV relay networks. IEEE Transactions on Wireless Communications, 19(11):7033–7048, 2020.

[96] A. Howard, M. J. Matarić, and G. S. Sukhatme. An incremental self-deployment algorithm for mobile sensor networks. Autonomous Robots, 13(2):113–126, 2002.

[97] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec. Strategies for pre-training graph neural networks. arXiv preprint arXiv:1905.12265, 2019.

[98] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4700–4708, 2017.

[99] Q. Huang, M. Yamada, Y. Tian, D. Singh, and Y. Chang. Graphlime: Local interpretable model explanations for graph neural networks. IEEE Transactions on Knowledge and Data Engineering, 2022. Early Access.

[100] Z. Huang, M. Kosan, S. Medya, S. Ranu, and A. Singh. Global counterfactual explainer for graph neural networks. In Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining, pages 141–149, 2023.

[101] F. K. Hwang and D. S. Richards. Steiner tree problems. Networks, 22(1):55–89, 1992.

[102] T. Ichmoukhamedov, J. Hinns, and D. Martens. How good is my story? towards quantitative metrics for evaluating llm-generated xai narratives. arXiv preprint arXiv:2412.10220, 2024.

[103] N. C. Institute. Aids antiviral screen data, 2004. Accessed: 2025-01-10.

[104] A. Jacovi and Y. Goldberg. Towards faithfully interpretable NLP systems: How should we define and evaluate faithfulness? In D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, editors, Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 4198–4205, Online, July 2020. Association for Computational Linguistics.

[105] Z. Ji and M. Telgarsky. Gradient Descent Aligns the Layers of Deep Linear Networks. In 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net, 2019.

[106] A. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, et al. Mistral 7b (2023). arXiv preprint arXiv:2310.06825, 2023.

[107] A. Kadiwal. Water potability dataset, 2020. Accessed: 2025-01-27.

[108] H. Kang, G. Han, and H. Park. Unr-explainer: Counterfactual explanations for unsupervised node representation learning models. In The Twelfth International Conference on Learning Representations.

[109] A.-H. Karimi, G. Barthe, B. Balle, and I. Valera. Model-Agnostic Counterfactual Explanations for Consequential Decisions. In International Conference on Artificial Intelligence and Statistics, pages 895–905. PMLR, 2020.

[110] A.-H. Karimi, B. Schölkopf, and I. Valera. Algorithmic recourse: From counterfactual explanations to interventions. In Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency (FAccT). ACM, 2020.

[111] G. Karystinos and D. Pados. On Overfitting, Generalization, and Randomly Expanded Training Sets. IEEE Transactions on Neural Networks, 11(5):1050–1057, 2000.

[112] T. Kimura and M. Ogura. Distributed collaborative 3d-deployment of UAV base stations for on-demand coverage. In IEEE International Conference on Computer Communications (INFOCOM), pages 1748–1757. IEEE, 2020.

[113] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.

[114] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In International Conference on Learning Representations (ICLR), 2017.

[115] V. Kočić, N. Lukač, D. Rožajac, S. Schweng, C. Gollob, A. Nothdurft, K. Stampfer, J. Del Ser, and A. Holzinger. Llm in the loop: A framework for contextualizing counterfactual segment perturbations in point clouds. IEEE access, 2025.

[116] V. Koltchinskii and D. Panchenko. Empirical Margin Distributions and Bounding the Generalization Error of Combined Classifiers. The Annals of Statistics, 30(1):1–50, 2002.

[117] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for steiner trees. Acta Informatica, 15:141–145, 1981.

[118] N. M. Kriege, M. Fey, D. Fisseler, P. Mutzel, and F. Weichert. Recognizing cuneiform signs using graph based methods. In International Workshop on Cost-Sensitive Learning, pages 31–44. PMLR, 2018.

[119] A. Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009.

[120] M. Kulbacki, J. Segen, W. Knieć, R. Klempous, K. Kluwak, J. Nikodem, J. Kulbacka, and A. Serester. Survey of drones for agriculture automation from planting to harvest. In 2018 IEEE 22nd International Conference on Intelligent Engineering Systems (INES), pages 000353–000358. IEEE, 2018.

[121] T. Le, S. Wang, and D. Lee. GRACE: Generating Concise and Informative Contrastive Sample to Explain Neural Network Model's Prediction. KDD '20, pages 238–248, New York, NY, USA, 2020. Association for Computing Machinery.

[122] G.-H. Lin and G. Xue. Steiner tree problem with minimum number of steiner points and bounded edge-length. Information Processing Letters, 69(2):53–57, 1999.

[123] X. Liu, T. Xi, E. Ngai, and W. Wang. Path planning for aerial sensor networks with connectivity constraints. In 2017 IEEE International Conference on Communications (ICC), pages 1–6. IEEE, 2017.

[124] Y. Liu, C. Chen, Y. Liu, X. Zhang, and S. Xie. Multi-objective explanations of gnn predictions. In 2021 IEEE International Conference on Data Mining (ICDM), pages 409–418. IEEE, 2021.

[125] R. Lopez, A. Gayoso, and N. Yosef. Enhancing Scientific Discoveries in Molecular Biology with Deep Generative Models. Molecular Systems Biology, 16(9):e9198, 2020.

[126] A. Lucic, H. Oosterhuis, H. Haned, and M. de Rijke. FOCUS: Flexible Optimizable Counterfactual Explanations for Tree Ensembles. In Proceedings of the AAAI conference on artificial intelligence, volume 36, pages 5313–5322, 2022.

[127] A. Lucic, M. ter Hoeve, G. Tolomei, M. de Rijke, and F. Silvestri. Cf-gnnexplainer: Counterfactual explanations for graph neural networks. In AISTATS, volume 151 of Proceedings of Machine Learning Research, pages 4499–4511. PMLR, 2022.

[128] A. Lucic, M. A. Ter Hoeve, G. Tolomei, M. De Rijke, and F. Silvestri. Cf-gnnexplainer: Counterfactual explanations for graph neural networks. In International Conference on Artificial Intelligence and Statistics, pages 4499–4511. PMLR, 2022.

[129] A. Lucic, M. A. Ter Hoeve, G. Tolomei, M. De Rijke, and F. Silvestri. CF-GNNExplainer: Counterfactual Explanations for Graph Neural Networks. In G. Camps-Valls, F. J. R. Ruiz, and I. Valera, editors, Proceedings of The 25th International Conference on Artificial Intelligence and Statistics, volume 151 of Proceedings of Machine Learning Research, pages 4499–4511. PMLR, 28–30 Mar 2022.

[130] S. Lundberg. A unified approach to interpreting model predictions. arXiv preprint arXiv:1705.07874, 2017.

[131] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang. Parameterized explainer for graph neural network. In NeurIPS, 2020.

[132] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang. Parameterized explainer for graph neural network. Advances in neural information processing systems, 33:19620–19631, 2020.

[133] J. Ma, R. Guo, S. Mishra, A. Zhang, and J. Li. Clear: Generative counterfactual explanations on graphs. Advances in neural information processing systems, 35:25895–25907, 2022.

[134] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. In International Conference on Learning Representations, 2018.

[135] Y. Marchukov and L. Montano. Fast and scalable multi-robot deployment planning under connectivity constraints. In 2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), pages 1–7, 2019.

[136] d. B. Mark, C. Otfried, v. K. Marc, and O. Mark. Computational geometry algorithms and applications. Spinger, 2008.

[137] D. Martens, J. Hinns, C. Dams, M. Vergouwen, and T. Evgeniou. Tell me a story! narrative-driven xai with large language models. Decision Support Systems, 191:114402, 2025.

[138] M.-A. Messous, S.-M. Senouci, H. Sedjelmaci, and S. Cherkaoui. A game theory based efficient computation offloading in an UAV network. IEEE Transactions on Vehicular Technology, 68(5):4964–4974, 2019.

[139] B. Min, H. Ross, E. Sulem, A. Veyseh, T. Nguyen, O. Sainz, E. Agirre, I. Heinz, and D. Roth. Recent advances in natural language processing via large pre-trained language models: a survey. arxiv. arXiv preprint arXiv:2111.01243, 10, 2021.

[140] G. Montavon, W. Samek, and K.-R. Müller. Methods for interpreting and understanding deep neural networks. Digital signal processing, 73:1–15, 2018.

[141] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In Proceedings of the AAAI conference on artificial intelligence, volume 33, pages 4602–4609, 2019.

[142] R. K. Mothilal, A. Sharma, and C. Tan. Explaining machine learning classifiers through diverse counterfactual explanations. In Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency (FAccT). ACM, 2020.

[143] R. K. Mothilal, A. Sharma, and C. Tan. Explaining machine learning classifiers through diverse counterfactual explanations. In Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency, pages 607–617, 2020.

[144] A. Moussa. Delaunay triangulation and voronoi diagrams, 2022.

[145] M. Movin, F. Siciliano, R. Ferreira, F. Silvestri, and G. Tolomei. Consistent counterfactual explanations via anomaly control and data coherence. IEEE Transactions on Artificial Intelligence, 2024.

[146] M. Movin, F. Siciliano, R. Ferreira, F. Silvestri, and G. Tolomei. Consistent Counterfactual Explanations via Anomaly Control and Data Coherence. IEEE Transactions on Artificial Intelligence, 6(4):794–804, 2025.

[147] E. Natalizio, N. R. Zema, E. Yanmaz, L. D. P. Pugliese, and F. Guerriero. Take the field from your smartphone: Leveraging UAVs for event filming. IEEE Transactions on Mobile Computing, 19(8):1971–1983, 2019.

[148] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini, et al. Deep learning recommendation model for personalization and recommendation systems. arXiv preprint arXiv:1906.00091, 2019.

[149] T. N. Nguyen, B.-H. Liu, and S.-Y. Wang. On new approaches of maximum weighted target coverage and sensor connectivity: Hardness and approximation. IEEE Transactions on Network Science and Engineering, 7(3):1736–1751, 2019.

[150] nsnam. Network Simulator-3 (NS-3), 2021.

[151] Nvidia. Jetson nano: Deep learning inference benchmarks, 2021.

[152] Office of Science and Technology Policy. Blueprint for an ai bill of rights: Making automated systems work for the american people, October 2023. Accessed: February 21, 2025.

[153] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback. In Advances in Neural Information Processing Systems 35 (NeurIPS), 2022.

[154] B. Pan, Z. Xiong, G. Wu, Z. Zhang, Y. Zhang, and L. Zhao. Tagexplainer: Narrating graph explanations for text-attributed graph learning models. arXiv preprint arXiv:2410.15268, 2024.

[155] E. Parliament. Artificial intelligence act: Deal on comprehensive rules for trustworthy ai, 2023.

[156] M. Pawelczyk, C. Agarwal, S. Joshi, S. Upadhyay, and H. Lakkaraju. Exploring Counterfactual Explanations Through the Lens of Adversarial Examples: A Theoretical and Empirical Analysis . In G. Camps-Valls, F. J. R. Ruiz, and I. Valera, editors, Proceedings of The 25th International Conference on Artificial Intelligence and Statistics, volume 151 of Proceedings of Machine Learning Research, pages 4574–4594. PMLR, 28–30 Mar 2022.

[157] M. A. Prado-Romero, B. Prenkaj, G. Stilo, and F. Giannotti. A survey on graph counterfactual explanations: Definitions, methods, evaluation, and research challenges. ACM Computing Surveys, Sept. 2023.

[158] J. Qiu, J. Tang, H. Ma, Y. Dong, K. Wang, and J. Tang. Deepinf: Social influence prediction with deep learning. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD), pages 2110–2119. ACM, 2018.

[159] L. Ranaldi and A. Freitas. Self-refine instruction-tuning for aligning reasoning in language models. arXiv preprint arXiv:2405.00402, 2024.

[160] M. T. Rashid, D. Y. Zhang, and D. Wang. Socialdrone: An integrated social media and drone sensing system for reliable disaster response. In IEEE International Conference on Computer Communications (INFOCOM), pages 218–227. IEEE, 2020.

[161] G. Raskutti, M. J. Wainwright, and B. Yu. Early Stopping for Non-Parametric Regression: An Optimal Data-Dependent Stopping Rule. In 2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton), pages 1318–1325, 2011.

[162] J. Reich, V. Misra, D. Rubenstein, and G. Zussman. Connectivity maintenance in mobile wireless networks via constrained mobility. IEEE Journal on Selected Areas in Communications, 30(5):935–950, 2012.

[163] M. T. Ribeiro, S. Singh, and C. Guestrin. " why should i trust you?" explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pages 1135–1144, 2016.

[164] K. Riesen and H. Bunke. Iam graph database repository for graph based pattern recognition and machine learning. In Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshop, SSPR & SPR 2008, volume 5342 of Lecture Notes in Computer Science, pages 287–297. Springer, 2008.

[165] G. Robins and A. Zelikovsky. Improved steiner tree approximation in graphs. In Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 770–779. SIAM, 2000.

[166] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. International journal of computer vision, 115(3):211–252, 2015.

[167] M. Samir, C. Assi, S. Sharafeddine, D. Ebrahimi, and A. Ghrayeb. Age of information aware trajectory planning of UAVs in intelligent transportation systems: A deep learning approach. IEEE Transactions on Vehicular Technology, 69(11):12382–12395, 2020.

[168] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. CoRR, abs/1801.04381, 2018.

[169] J. Scherer and B. Rinner. Persistent multi-uav surveillance with energy and communication constraints. In 2016 IEEE International Conference on Automation Science and Engineering (CASE), pages 1225–1230. IEEE, 2016.

[170] J. Scherer and B. Rinner. Multi-robot persistent surveillance with connectivity constraints. IEEE Access, 8:15093–15109, 2020.

[171] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In European Semantic Web Conference (ESWC), pages 593–607. Springer, 2018.

[172] F. Senel and M. Younis. Relay node placement in structurally damaged wireless sensor networks via triangular steiner tree approximation. Computer Communications, 34(16):1932–1941, 2011.

[173] D. Slack, S. Krishna, H. Lakkaraju, and S. Singh. Talktomodel: Explaining machine learning models with interactive natural language conversations. arXiv preprint arXiv:2207.04154, 2022.

[174] D. Soudry, E. Hoffer, M. S. Nacson, S. Gunasekar, and N. Srebro. The Implicit Bias of Gradient Descent on Separable Data. J. Mach. Learn. Res., 19(1):2822–2878, Jan. 2018.

[175] S. Srinivas, A. Subramanya, and R. Venkatesh Babu. Training sparse neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition workshops, pages 138–145, 2017.

[176] I. Stepin, J. M. Alonso, A. Catala, and M. Pereira-Fariña. A Survey of Contrastive and Counterfactual Explanation Generation Methods for Explainable Artificial Intelligence. IEEE Access, 9:11974–12001, 2021.

[177] D. Stutz, M. Hein, and B. Schiele. Disentangling adversarial robustness and generalization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 6976–6987. IEEE, 2019.

[178] Y. Sun, X. Wang, and X. Tang. Deep Learning Face Representation from Predicting 10,000 Classes. In 2014 IEEE Conference on Computer Vision and Pattern Recognition, pages 1891–1898, 2014.

[179] G. Tan, S. A. Jarvis, and A.-M. Kermarrec. Connectivity-guaranteed and obstacle-adaptive deployment schemes for mobile sensor networks. IEEE Transactions on Mobile Computing, 8(6):836–848, 2009.

[180] J. Tan, S. Geng, Z. Fu, Y. Ge, S. Xu, Y. Li, and Y. Zhang. Learning and evaluating graph neural network explanations based on counterfactual and factual reasoning. In Proceedings of the ACM Web Conference 2022, pages 1018–1027, 2022.

[181] D. Tateo, J. Banfi, A. Riva, F. Amigoni, and A. Bonarini. Multiagent connected path planning: PSPACE-Completeness and how to deal with it. In Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[182] A. Team. Grok-4 vs. OpenAI Models: A Deep Comparison for Startup Builders. `https://axion.pm/blogs/grok-4-vs-openai-models-a-deep-comparison-for-startup-builders/`, July 2025.

[183] C. Team. Grok-4 vs OpenAI O3: A Detailed Comparison. `https://composio.dev/blog/grok-4-vs-openai-o3-a-detailed-comparison`, July 2025.

[184] Q. Team. Qwen2.5: A party of foundation models, September 2024.

[185] G. Tolomei and F. Silvestri. Generating actionable interpretations from ensembles of decision trees. IEEE Transactions on Knowledge and Data Engineering, 33(4):1540–1553, 2019.

[186] G. Tolomei and F. Silvestri. Generating Actionable Interpretations from Ensembles of Decision Trees. IEEE Transactions on Knowledge and Data Engineering, 33(4):1540–1553, 2021.

[187] G. Tolomei, F. Silvestri, A. Haines, and M. Lalmas. Interpretable predictions of tree-based ensembles via actionable feature tweaking. In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, pages 465–474, 2017.

[188] G. Tolomei, F. Silvestri, A. Haines, and M. Lalmas. Interpretable Predictions of Tree-based Ensembles via Actionable Feature Tweaking. KDD '17, pages 465–474, New York, NY, USA, 2017. Association for Computing Machinery.

[189] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971, 2023.

[190] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In Advances in Neural Information Processing Systems 30 (NeurIPS), 2017.

[191] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In International Conference on Learning Representations (ICLR), 2018.

[192] S. Verma, B. Armgaan, S. Medya, and S. Ranu. Induce: Inductive counterfactual explanations for graph neural networks. Transactions on Machine Learning Research, 2024.

[193] G. Vernikos, A. Bražinskas, J. Adamek, J. Mallinson, A. Severyn, and E. Malmi. Small language models improve giants by rewriting their outputs. arXiv preprint arXiv:2305.13514, 2023.

[194] P. Voigt and A. Von dem Bussche. The eu general data protection regulation (gdpr). A Practical Guide, 1st Ed., Cham: Springer International Publishing, 10(3152676):10–5555, 2017.

[195] S. Wachter, B. Mittelstadt, and C. Russell. Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR. Harvard Journal of Law & Technology., 31:841–887, 2017.

[196] S. Wachter, B. Mittelstadt, and C. Russell. Counterfactual explanations without opening the black box: Automated decisions and the gdpr. Harvard Journal of Law & Technology, 31(2):841–887, 2018. Also available as SSRN Electronic Journal (2017).

[197] J. Wang, S. Zhang, Y. Xiao, and R. Song. A review on graph neural network methods in financial applications. arXiv preprint arXiv:2111.15367, 2021.

[198] X. Wang and L. Duan. Dynamic pricing and capacity allocation of UAV-provided mobile services. In IEEE International Conference on Computer Communications (INFOCOM), pages 1855–1863. IEEE, 2019.

[199] X. Wang, X. He, Y. Cao, M. Liu, and T.-S. Chua. Kgat: Knowledge graph attention network for recommendation. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD), pages 353–361. ACM, 2019.

[200] X. Wang, Y. Wu, A. Zhang, F. Feng, X. He, and T.-S. Chua. Reinforced causal explainer for graph neural networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 45(2):2297–2309, 2022.

[201] X. Wang, F. Zhang, M. Zhao, W. Li, X. Xie, and M. Guo. Neural graph collaborative filtering. In Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 165–174. ACM, 2019.

[202] Y. Wang, Y. Chen, Y. Li, Y. Yang, Y. Yan, and J. Tang. Graph machine learning in the era of large language models (llms). arXiv preprint arXiv:2404.14928, April 2024.

[203] D. Warde-Farley, I. J. Goodfellow, A. C. Courville, and Y. Bengio. An Empirical Analysis of Dropout in Piecewise Linear Networks. In Y. Bengio and Y. LeCun, editors, 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, 2014.

[204] H. Wasserbacher and M. Spindler. Machine learning for financial forecasting, planning and analysis: recent developments and pitfalls. Digital Finance, 4(1):63–88, 2022.

[205] G. P. Wellawatte, A. Seshadri, and A. D. White. Model agnostic generation of counterfactual explanations for molecules. Chemical science, 13(13):3697–3705, 2022.

[206] D. Whiteson. HIGGS. UCI Machine Learning Repository, 2014. DOI: https://doi.org/10.24432/C5V312.

[207] F. Wong, E. J. Zheng, J. A. Valeri, N. M. Donghia, M. N. Anahtar, S. Omori, A. Li, A. Cubillos-Ruiz, A. Krishnan, W. Jin, A. L. Manson, J. Friedrichs, R. Helbig, B. Hajian, D. K. Fiejtek, F. F. Wagner, H. H. Soutter, A. M. Earl, J. M. Stokes, L. Renner, and J. J. Collins. Discovery of a structural class of antibiotics with explainable deep learning. Nature, 2023.

[208] K. Wu and Y. Yu. Understanding Adversarial Robustness: The Trade-off between Minimum and Average Margin, 2019.

[209] Q. Wu, Y. Zeng, and R. Zhang. Joint trajectory and communication design for multi-uav enabled wireless networks. IEEE Transactions on Wireless Communications, 17(3):2109–2121, 2018.

[210] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems, 32(1):4–24, 2021.

[211] xAI. Grok-4 large language model. `https://x.ai`, 2025.

[212] G. Xia. The stretch factor of the delaunay triangulation is less than 1.998. SIAM Journal on Computing, 42(4):1620–1659, 2013.

[213] Y. Xie, S. Zhou, Y. Xiao, S. Kulturel-Konak, and A. Konak. A β-accurate linearization method of euclidean distance for the facility layout problem with heterogeneous distance metrics. European Journal of Operational Research, 265(1):26–38, 2018.

[214] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In International Conference on Learning Representations (ICLR), 2019.

[215] B. Xua and G. Yang. Interpretability research of deep learning: A literature survey. Information Fusion, page 102721, 2024.

[216] B. Yang, X. Cao, C. Yuen, and L. Qian. Offloading Optimization in Edge Computing for Deep Learning Enabled Target Tracking by Internet-of-UAVs. IEEE Internet of Things Journal, 8(12):9878–9893, 2020.

[217] Z. Yang, W. Cohen, and R. Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In International conference on machine learning, pages 40–48. PMLR, 2016.

[218] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec. Gnnexplainer: Generating explanations for graph neural networks. In Advances in Neural Information Processing Systems 32 (NeurIPS), pages 9244–9255. Curran Associates, Inc., 2019.

[219] K. Y. Yip and M. Gerstein. Training Set Expansion: An Approach to Improving the Reconstruction of Biological Networks from Limited and Uneven Reliable Interactions. Bioinformatics, 25(2):243–250, 11 2008.

[220] H. Yuan, H. Yu, J. Wang, K. Li, and S. Ji. On explainability of graph neural networks via subgraph explorations. In ICML, volume 139 of Proceedings of Machine Learning Research, pages 12241–12252. PMLR, 2021.

[221] X. Zeng. Enhancing the interpretability of shap values using large language models. arXiv preprint arXiv:2409.00079, 2024.

[222] Y. Zhang, M. Khalifa, L. Logeswaran, J. Kim, M. Lee, H. Lee, and L. Wang. Small language models need strong verifiers to self-correct reasoning. arXiv preprint arXiv:2404.17140, 2024.

[223] A. Zytek, S. Pido, S. Alnegheimish, L. Berti-Equille, and K. Veeramachaneni. Explingo: Explaining ai predictions using large language models. In 2024 IEEE International Conference on Big Data (BigData), pages 1197–1208. IEEE, 2024.

[224] A. Zytek, S. Pidò, and K. Veeramachaneni. Llms for xai: Future directions for explaining explanations. arXiv preprint arXiv:2405.06064, 2024.

Appendix A

Appendix

A.1 Chapter 3 - Appendix

A.1.1 Parameters

The experimental configurations for our models were set as follows. Both the GCNConv and the GINEConv model was configured with three hidden layers, each containing 128 units, and a dropout of 0.5. The model training epochs are 500 with early stopping and a learning rate scheduling policy. Figure A.1 shows the values for train and test loss for the models used.

Model	Dataset	Test Accuracy
GINEConv	Cuneiform	0.82 ± 0.05
GCNConv	Citeseer	0.74 ± 0.005
GCNConv	PubMed	0.87 ± 0.001
GCNConv	Cora	0.86 ± 0.01
GCNConv	AIDS	0.98 ± 0.001

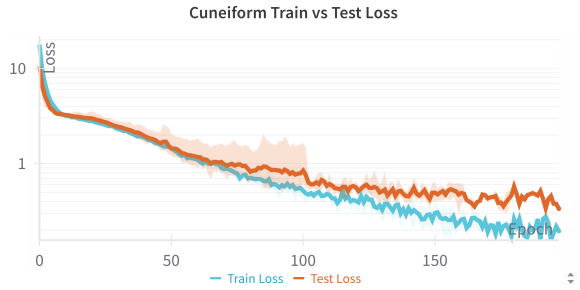
Table A.1: Accuracy values for each pair model-dataset used during the experiments

Table A.1 instead shows the different accuracy values reached by the models.

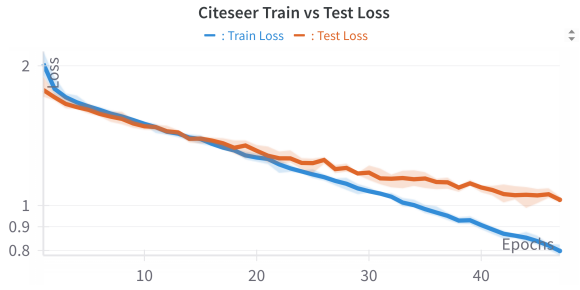
A.1.2 Node Classification Results

Table A.2 shows the performance of various explainers on the PubMed and Cora datasets. A careful examination of these results reveals important insights into the behavior of our UCExplainer approach compared to other methods. On PubMed, UCExplainer variants strike a favorable balance, with validity and fidelity scores around 0.66 and 0.75 respectively and very low distribution distances. On Cora, the UCExplainer methods continue to outperform all the baselines, reaching with all the scheduling policies, perfect validity with negligible node sparsity, and edge sparsity, even though the distribution distances remain a bit higher if compared to other baselines with much lower validity (i.e. EGO, CFF, UNR, Random Edges)

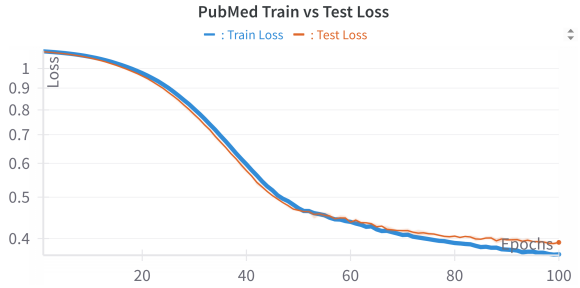
In summary, our approach consistently provides balanced, interpretable, and faithful explanations. The approach is robust across different datasets, achieving high validity and fidelity while minimizing both distribution distance and sparsity—qualities that are essential for effective explanation in graph neural networks.



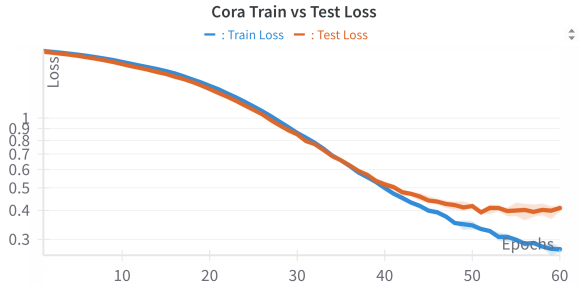
(a) Loss 1



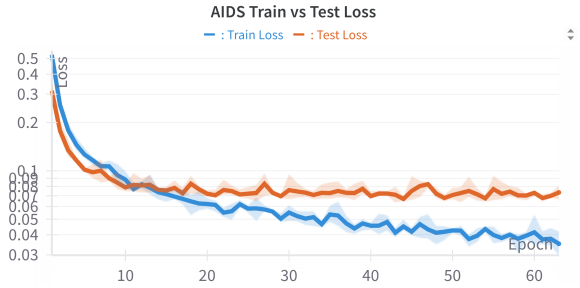
(b) Loss 2



(c) Loss 3



(d) Loss 4



(e) Loss 5

Figure A.1: This is the main caption for the 3×2 grid (five images shown).

A.1.3 Examples

In Figure A.2 there are 8 factual-counterfactual pairs obtained using the Cuneiform dataset and UCExplainer. In order to condense all the information available for the graph (node features and edge attributes) we encoded them as follow: The node features are visualized using:

- **Node Color**: the primary node feature ($\mathbf{X}_{i,0}$) determines color using the `viridis` colormap. Colors are normalized globally across each factual-counterfactual pair using min-max normalization.
- **Node Size**: Node sizes are computed as follows: $\text{node\_size}_i = |\sum_{j=0}^{d-1} \mathbf{X}_{i,j}| \cdot 30 + 50$ where d is the number of node features.

Edge features are represented through:

- **Edge Width**: the first edge feature determines line thickness, $\text{width}_{(i,j)} = |\mathbf{E}_{(i,j),0}| \cdot 2 + 0.5$
- **Edge Color**: the second edge feature ($\mathbf{E}_{(i,j),1}$) determines color using the `plasma` colormap. Edge colors are normalized globally across factual-counterfactual pairs using min-max normalization.

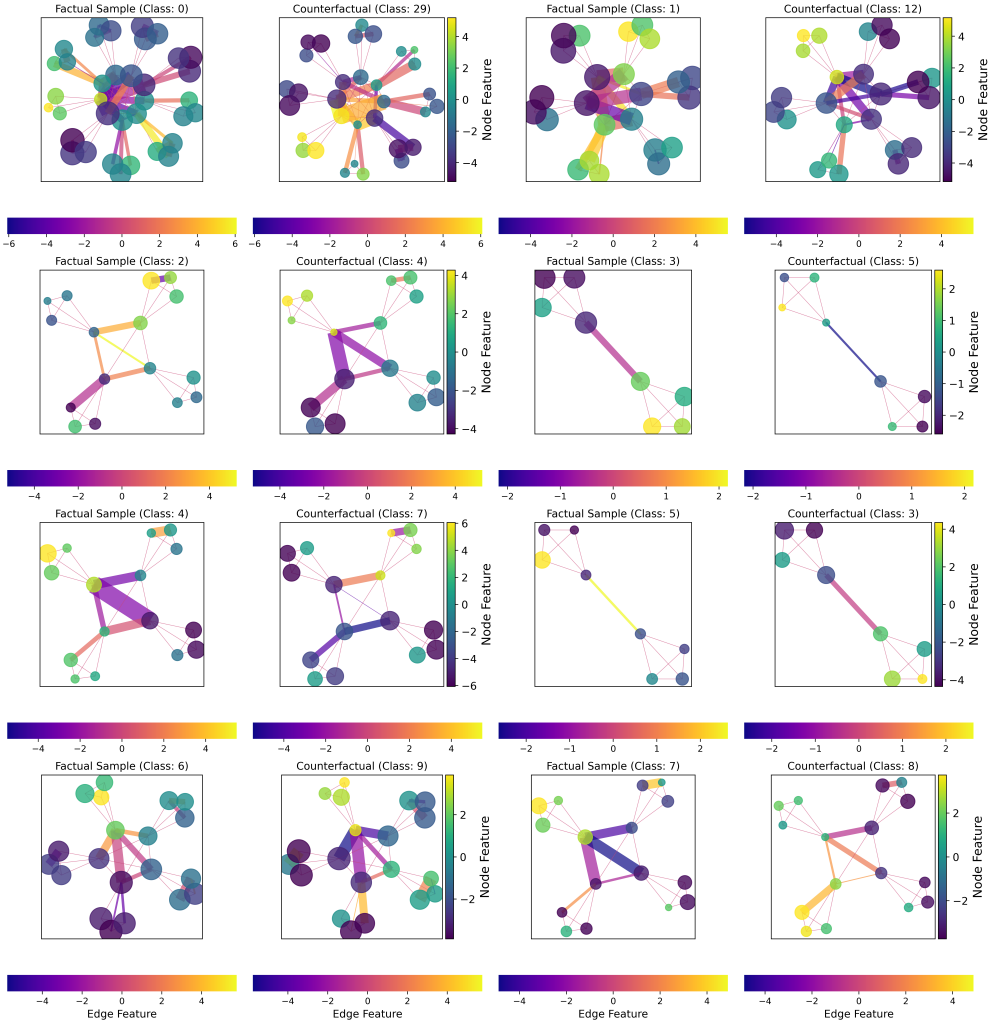


Figure A.2: Samples from the Cuneiform dataset and their counterfactual samples obtained using UCExplainer

Explainers	Validity ↑	Fidelity ↑	Distribution Distance ↓	Node Sparsity ↓	Edge Sparsity ↓
			Dataset: PubMed		
UCExplainer$_{def}$	0.650(±0.028)	0.754(±0.022)	0.108(±0.002)	0.107(±0.000)	0.12(±0.02)
UCExplainer$_{dyn}$	**0.663**(±**0.043**)	0.756(±0.020)	0.180(±0.011)	0.107(±0.000)	**0.000**(±**0.000**)
UCExplainer$_{exp}$	0.658(±0.041)	0.749(±0.014)	0.166(±0.020)	0.107(±0.000)	**0.000**(±**0.000**)
UCExplainer$_{lin}$	0.662(±0.045)	0.751(±0.016)	0.126(±0.004)	0.107(±0.000)	**0.000**(±**0.000**)
UCExplainer$_{sin}$	0.648(±0.029)	0.753(±0.019)	0.117(±0.003)	0.107(±0.000)	**0.000**(±**0.000**)
EGO	0.043(±0.012)	0.650(±0.238)	0.103(±0.010)	*n.a.*	0.984(±0.004)
Random Edges	0.293(±0.022)	0.607(±0.037)	0.068(±0.002)	*n.a.*	0.478(±0.001)
Random Features	0.513(±0.025)	0.734(±0.016)	1.661(±0.002)	**0.032**(±**0.000**)	*n.a.*
CFF	0.030(±0.014)	−0.900(±0.200)	**0.056**(±**0.007**)	*n.a.*	0.660(±0.084)
CF-GNNExplainer	0.123(±0.009)	0.511(±0.125)	0.080(±0.001)	*n.a.*	0.004(±0.001)
UNR	0.008(±0.006)	**1.000**(±**0.000**)	0.065(±0.008)	*n.a.*	0.001(±0.001)
GNNExplainer	0.007(±0.003)	0.544(±0.075)	0.079(±0.007)	*n.a.*	0.053(±0.012)
InduCE [1]	0.138(±0.018)	0.623(±0.070)	0.061(±0.004)	*n.a.*	0.003(±0.001)
PGExplainer [2]	0.603(±0.016)	0.781(±0.021)	0.654(±0.023)	*n.a.*	0.211(±0.071)
			Dataset: Cora		
UCExplainer$_{def}$	**1.000**(±**0.000**)	0.855(±0.003)	1.628(±0.104)	0.018(±0.001)	**0.000**(±**0.000**)
UCExplainer$_{dyn}$	**1.000**(±**0.000**)	0.855(±0.003)	7.558(±0.044)	0.133(±0.003)	**0.000**(±**0.000**)
UCExplainer$_{exp}$	**1.000**(±**0.000**)	0.855(±0.003)	9.573(±0.176)	0.175(±0.006)	**0.000**(±**0.000**)
UCExplainer$_{lin}$	**1.000**(±**0.000**)	0.855(±0.003)	2.727(±0.113)	0.035(±0.001)	**0.000**(±**0.000**)
UCExplainer$_{sin}$	**1.000**(±**0.000**)	0.855(±0.003)	2.800(±0.111)	0.037(±0.001)	**0.000**(±**0.000**)
EGO	0.022(±0.003)	**1.000**(±**0.000**)	0.535(±0.038)	*n.a.*	0.980(±0.001)
Random Edges	0.240(±0.016)	0.729(±0.014)	0.564(±0.014)	*n.a.*	0.472(±0.003)
Random Features	0.127(±0.013)	0.947(±0.005)	18.795(±0.313)	0.492(±0.000)	*n.a.*
CFF	0.010(±0.009)	−0.556(±0.770)	**0.482**(±**0.146**)	*n.a.*	0.715(±0.246)
CF-GNNExplainer	0.165(±0.022)	0.782(±0.049)	0.598(±0.039)	*n.a.*	0.08(±0.001)
UNR	0.017(±0.004)	**1.000**(±**0.000**)	0.747(±0.432)	*n.a.*	0.012(±0.013)
GNNExplainer	0.151(±0.015)	0.004(±0.002)	5.531(±0.602)	*n.a.*	0.572(±0.031)
induCE [1]	0.413(±0.023)	0.187(±0.014)	3.237(±0.315)	*n.a.*	0.751(±0.024)
PGExplainer [2]	0.845(±0.13)	0.722(±0.036)	1.677(±0.011)	*n.a.*	0.253(±0.003)

Table A.2: Results for PubMed and Cora. The oracles g use GCNConv layers. In **bold** the best result, the second best result is underlined

Table A.3: Main properties and description of the datasets.

Dataset	#Samples	#Features	#Min. Class (%)	#Maj. Class (%)	Description
`Water`	3,276	9	1,278 (39%)	1,998 (61%)	This tabular dataset includes various water quality metrics and is used to predict water potability, i.e., whether it is safe for human consumption.
`Phoneme`	5,404	5	1,586 (29%)	3,818 (71%)	This tabular dataset contains five attributes selected to characterize each vowel, with the goal of distinguishing between nasal and oral sounds.
`Higgs`	200,000	28	94,131 (47%)	105,869 (53%)	This dataset is a random sample without replacement of the original `Higgs` tabular dataset, comprising 18% of the total 1.1 million instances, and is used to distinguish between a signal process that produces Higgs bosons and a background process that does not.
`CIFAR-10`	12,000	$3 \times 32 \times 32$	6,000 (50%)	6,000 (50%)	This dataset is a sample of the original `CIFAR-10` image dataset, adapted to a binary classification setting, containing only instances from class 3 ("cats") and class 5 ("dogs").

A.2 Chapter 4 - Additional Experimental Details

A.2.1 Datasets, Models, and Tasks

Three of the four datasets used in this study, namely `Water`, `Phoneme`, and `Higgs`, are tabular, while the fourth, `CIFAR-10`, is a popular image dataset. Table A.3 summarizes key characteristics of each dataset, including the number of samples, number of features, class distribution (minority and majority class sizes), and a brief description.

All datasets except `CIFAR-10` are naturally associated with a binary classification task. To adapt `CIFAR-10` to a binary classification setting, we retain only instances from class 3 ("cats") and class 5 ("dogs").

For the `Water`, `Phoneme`, and `Higgs` datasets, we performed an 80/20 train-test split. For `CIFAR-10`, we used the standard train-test split provided by the `torchvision` library.

Table A.4 summarizes the key characteristics of the models evaluated in our experiments. The selection covers a broad spectrum of architectures, from simple logistic regression models with a few thousand parameters to deep neural networks comprising tens of millions of parameters.

Table A.4: Comparison of various models across datasets, layer configurations, and parameter counts.

Model	**Dataset**	**Layers Width**	**#Parameters**
LR	`Water`	N/A	5,005
$\text{MLP}_{\text{small}}$	`Water`	[100, 30]	3,930
$\text{MLP}_{\text{large}}$	`Water`	[150, 1000, 150, 30]	305,880
LR	`Phoneme`	N/A	4,368
$\text{MLP}_{\text{small}}$	`Phoneme`	[100, 40]	4,540
$\text{MLP}_{\text{large}}$	`Phoneme`	[150, 1000, 150, 30]	305,280
$\text{MLP}_{\text{small}}$	`Higgs`	[100, 30]	5,830
$\text{MLP}_{\text{large}}$	`Higgs`	[150, 1000, 150, 30]	308,730
CNN	`CIFAR-10`	ResNet-18	11,200,000

A.2.2 Training Settings

Table A.5 illustrates the training settings used for each dataset. We adopt Adam as optimizer due to its faster convergence properties. This choice enables us to reduce the number of training epochs while still performing an exhaustive hyperparameter search within the same computational budget.

To illustrate Adam's faster convergence in practice, Figure A.3 compares its performance to standard stochastic gradient descent (SGD) when training an MLP on the `Higgs` dataset.

Table A.5: Training settings for each dataset.

Dataset	**Optimizer**	**#Epochs**	**Batch Size**	**Learning Rate** (η)
`Water`	Adam	2,000	128	0.001
`Phoneme`	Adam	2,000	128	0.001
`Higgs`	Adam	200	128	0.001
`CIFAR-10`	Adam	200	128	0.001

Training and testing are performed over 2,000 epochs for the `Water` and `Phoneme` datasets, and over 200 epochs for both `CIFAR-10` and `Higgs`. In all cases, we use a batch size of 128, and we set the learning rate to $\eta = 0.001$ with no weight decay.

Table A.6: Hyperparameter settings used to generate the results in Table 4.1. L1-Reg and L2-Reg report the regularization coefficients, i.e., the weights of the L_1 and L_2 norms of the model parameters, respectively. Dropout indicates the dropout probability. Early Stopping shows the patience parameter. PGD reports: the step size of the adversarial attack (α); the maximum perturbation budget (ϵ); and the number of attack iterations (k). Lastly, CF-Reg represents the regularization coefficients α and β.

Model	Dataset	L1-Reg	L2-Reg	Dropout	Early Stopping	PGD	CF-Reg (ours)
LR	`Water`	4.933 e-02	7.413 e-01	N/A	5	1.076 e-02/1.128 e-02/15	3.353 e-01/9.816 e-01
$\text{MLP}_{\text{small}}$	`Water`	1.280 e-02	2.920 e-03	2.004 e-01	15	1.364 e-01/4.714 e-02/05	8.325 e-01/1.886 e+00
$\text{MLP}_{\text{large}}$	`Water`	1.605 e-02	2.613 e-03	8.192 e-01	10	9.430 e-01/1.614 e-02/05	1.121 e+00/2.118 e+00
LR	`Phoneme`	1.796 e-02	3.659 e-02	N/A	260	1.543 e-02/6.009 e-03/05	1.303 e-05/3.461 e-02
$\text{MLP}_{\text{small}}$	`Phoneme`	1.097 e-03	6.972 e-03	1.606 e-01	280	1.100 e-02/9.069 e-02/20	1.485 e-02/1.883 e-01
$\text{MLP}_{\text{large}}$	`Phoneme`	5.885 e-03	3.003 e-03	1.001 e-02	260	1.044 e-02/7.132 e-03/05	5.539 e-03/1.797 e-01
$\text{MLP}_{\text{small}}$	`Higgs`	6.667 e-03	3.214 e-04	9.820 e-02	50	3.099 e-02/2.153 e-02/05	1.097 e-01/4.724 e-01
$\text{MLP}_{\text{large}}$	`Higgs`	1.391 e-02	1.675 e-04	3.918 e-01	5	1.422 e-02/8.403 e-02/05	4.380 e-01/2.289 e+00
CNN	`CIFAR-10`	8.700 e-06	4.868 e-06	N/A	160	3.755 e-01/5.043 e-03/07	7.246 e-04/1.852 e-01

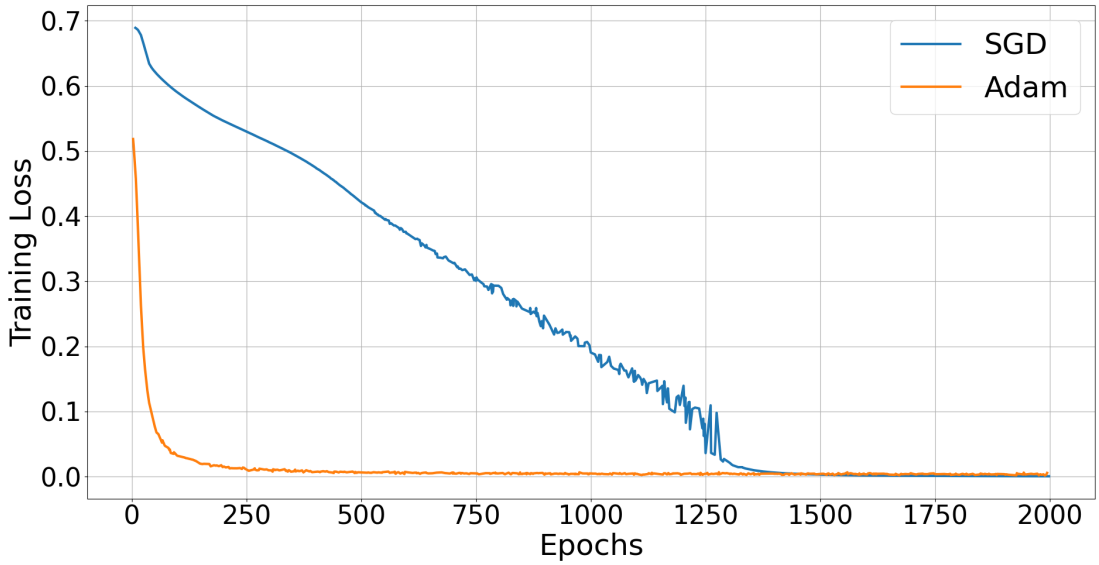


Figure A.3: The training loss evolution over training epochs for an MLP trained on the `Higgs` dataset without regularization with SGD (blue) or Adam (orange) as optimizer. Adam shows a faster fit to training data in terms of training epochs.

A.3 Counterfactual Perturbation vs. Overfitting

In Section 4.2.3, we empirically demonstrated that the average ε-VCP is correlated with the risk of overfitting: specifically, higher training accuracy tends to correspond to a larger average ε-VCP. That analysis provided a general observation, with the average ε-VCP estimated using Monte Carlo integration.

In this section, instead, we conduct a more targeted analysis by examining the relationship between the test loss (i.e., the complement of test accuracy) and the average L_2-norm of the counterfactual perturbation vectors ($\overline{||\boldsymbol{\delta}||}$) over the course of training. The latter serves as a practical proxy for the average ε-VCP.

Figure A.4 illustrates the evolution of $\overline{||\boldsymbol{\delta}||}$ and the test loss over training epochs for an MLP trained without regularization on the `Water` dataset. The plot shows a clear trend as the model starts to overfit the data (evidenced by an increase in test loss). Notably, $\overline{||\boldsymbol{\delta}||}$ begins to decrease, which aligns with the hypothesis that the average distance to the optimal counterfactual example gets smaller as the model's decision boundary becomes increasingly adherent to the training data.

It is worth noticing that this trend is heavily influenced by the choice of the counterfactual generator model. In particular, the relationship between $\overline{||\boldsymbol{\delta}||}$ and the degree of overfitting may

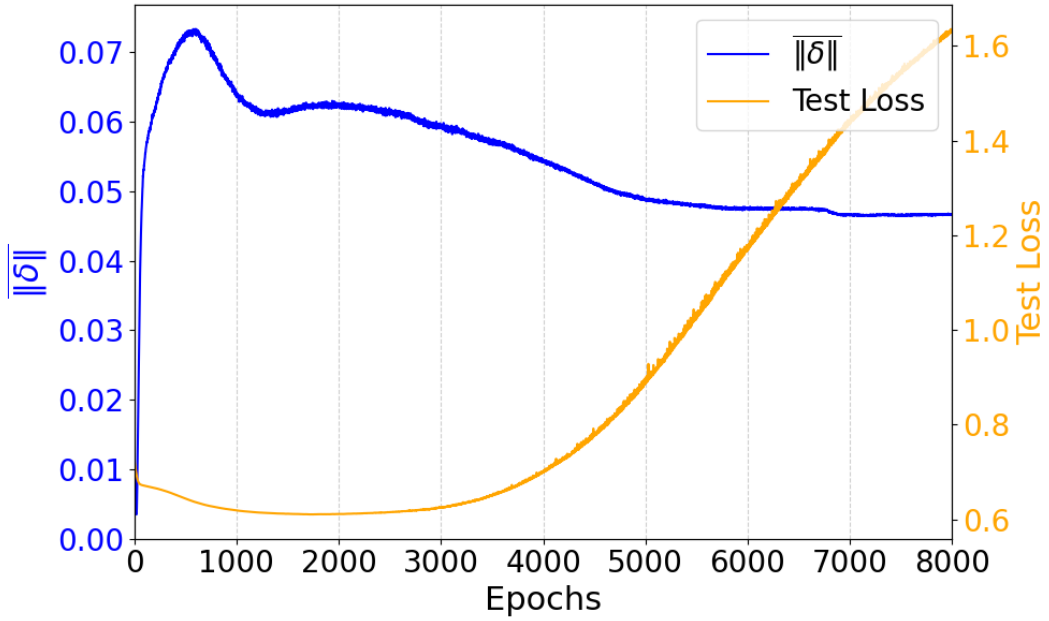


Figure A.4: The average counterfactual perturbation vector $\overline{||\boldsymbol{\delta}||}$ (left y-axis) and the cross-entropy test loss (right y-axis) over training epochs (x-axis) for an MLP trained on the `Water` dataset *without* regularization.

become even more pronounced when leveraging more accurate counterfactual generators. However, these models often come at the cost of higher computational complexity, and their exploration is left to future work.

Nonetheless, we expect that $\overline{||\boldsymbol{\delta}||}$ will eventually stabilize at a plateau, as the average L_2-norm of the optimal counterfactual perturbations cannot vanish to zero.

A.4 Hyperparameter Tuning

Most regularization techniques are highly sensitive to hyperparameter choices. Table A.6 summarizes all hyperparameters used in this study. Each value was selected via random Bayesian optimization over approximately 80 trials, aiming to maximize accuracy on a held-out validation set.

In the table, L1-Reg and L2-Reg refer to the coefficients of the respective regularization terms, i.e., the weights of the L_1 and L_2 norms of the model parameters. The Dropout entry indicates the dropout probability applied within the model architecture. For Early Stopping, the reported value corresponds to the patience parameter, namely the number of epochs without improvement before training is stopped. The three entries listed under PGD correspond to: the step size of the adversarial attack (α); the maximum perturbation budget (ϵ); and the number of attack iterations (k). Finally, the two CF-Reg values represent the regularization coefficients α and β, as defined in (4.5) and (4.7), respectively.