Diagnosing AI Misbehavior: Why Do Models Fail?

by

Anqi Zhang

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

August, 2025

Dr. Jinyang Li

Dr. Aurojit Panda

Dedication

To those ordinary days that quietly shimmer with brilliance

Acknowledgements

I would like to express my deepest gratitude to my advisors, Professor Jinyang Li and Professor Aurojit Panda, for their invaluable guidance throughout these six years. At the beginning of my doctoral journey, Jinyang provided me with the intellectual freedom to explore different research directions and discover the areas and projects that truly sparked my interest. Through our collaborative exploration of various projects and countless progress discussions, I gradually absorbed the essential mindset for conducting research and learned the nuances of academic writing from both of them.

Our countless meetings were filled with brainstorming sessions about fascinating problems, discussions on how to evaluate whether a question is worth pursuing, debates over whether our solutions were rigorous, robust, and efficient, and deep dives into what our experiments - successful or not - were telling us. Every "aha!" moment brought me genuine excitement and joy. Equally memorable were those intense discussions where we passionately defended our viewpoints, particularly when facing Panda's precise and challenging questions. Initially, I often felt overwhelmed and uncertain, but over time, these intense discussions gradually taught me how to stay calm, think about whether I was expressing myself clearly, and engage in real back-and-forth academic discussions. Looking back, this was such valuable growth. Through these processes, I have gradually internalized a systematic approach to conducting research — how to frame problems, analyze their significance, and work toward solutions. This intellectual framework represents one of the most precious gifts from my doctoral experience.

I am grateful to the many collaborators and committee members who contributed to the work that culminated in this dissertation. I would like to thank Siddhartha Sen and Mathias Lécuyer (now professor at UBC) from Microsoft Research NYC, along with my teammate Jinkun Lin, for our collaboration on the AME project. The brainstorming sessions were filled with creative sparks, and everyone's hard work made the project truly rewarding. I'm especially grateful to Sid for serving on my committee and providing continuous guidance throughout my PhD journey. I also want to thank my collaborators on the reasoning model project: Professor He He, Yulin Chen, Jane Pane, and Chen Zhao (Professor from NYU Shanghai). Especially He — our conversations were incredibly enlightening and helped me discover valuable new research directions that genuinely excited me. This came at a time when I was feeling quite lost and struggling to find my path, and her insights brought much-needed clarity and renewed enthusiasm to my work. Throughout my dissertation proposal and defense, my committee memebers - Jinyang, Panda, Sid, He, and Professor Saining Xie - all provided invaluable feedback, suggestions, and guidance. I'm deeply grateful for everyone's encouragement, affirmation, and the time and effort they invested in helping me succeed.

Beyond my academic journey, I am deeply grateful to the family and friends who provided unwavering support throughout these years. I want to especially thank my boyfriend, Chaofeng Wu, for his companionship and emotional support during my doctoral studies. Whether serving as a thoughtful listener and advisor in my academic discussions, sitting side by side with me as we worked through countless ordinary days, offering comfort and encouragement when experiments weren't yielding results and stress was overwhelming, or celebrating and sharing in my joy when breakthroughs finally came — his presence has been my sanctuary and the gentle constant that made even the most challenging days feel manageable.

I am profoundly grateful to my parents for their unwavering understanding of my choices and for the love and care they've conveyed across the distance. Even from afar, their care and support reaches me like warm sunlight, perfectly attuned to what I needed. This love has been

my inexhaustible wellspring of strength.

I also want to thank my friends, whose gatherings and laughter have been like stars illuminating this phase of my life. I also want to thank all my colleagues and labmates including Jinkun Lin, Tao Wang, Lingfan Yu, Ding Ding, Zhanghan Wang, and many others. I'm grateful that we shared part of this journey together.

To everyone who has been part of this journey — thank you. Your contributions, big and small, have made this work possible and my doctoral experience truly unforgettable.

Abstract

As artificial intelligence systems become increasingly pervasive across critical domains, understanding and diagnosing their failures has become paramount for ensuring safety, reliability, and trust. This dissertation addresses the fundamental challenge of diagnosing AI misbehavior across shifting deep learning paradigms – from classifiers to Graph Neural Networks (GNNs) to Large Reasoning Models (LRMs) – each exhibiting distinct failure problems that demand specialized diagnostic approaches.

This thesis presents three complementary contributions that develop targeted diagnostic frameworks for each major model paradigm. First, for classifiers, we introduce the Average Marginal Effect (AME), a scalable data attribution method that traces prediction errors back to problematic training data, achieving efficient attribution with only $O(klogN)$ evaluations under sparsity assumptions. Second, for GNNs, we develop MimicLDT, a novel long-distance targeted poisoning attack that reveals critical blind spots in GNN-explainer tools, and demonstrate how AME can be adapted to graph structures for diagnosing these stealthy attacks. Third, for large reasoning models, we design self-verification probes that uncover latent correctness signals in intermediate reasoning steps, achieving ROC-AUC scores above 0.7 with remarkably low expected calibration error below 0.1, and enable confidence-based early-exit strategies that reduce inference tokens by up to 24% without compromising accuracy. This work establishes a comprehensive framework for model-aware diagnostics that advances both our understanding of AI misbehavior and practical tools for building more trustworthy, efficient, and interpretable AI systems.

Contents

List of Figures

List of Tables

List of Appendices

1 Introduction

Artificial Intelligence (AI) and Deep Learning (DL) models have become pervasive across every domain of human activity, from healthcare and finance to autonomous systems and scientific discovery. During my PhD, the landscape of AI research and deployment has shifted dramatically: the focus has moved from classifiers tackling specific recognition tasks, to Graph Neural Networks (GNNs) handling complex relational data, and now to Large Language Models (LLMs) and Large Reasoning Models (LRMs) powering sophisticated generation and reasoning tasks. Each paradigm brought unique capabilities, evolving needs of AI applications, and correspondingly distinct failure modes that demanded targeted diagnostic approaches.

Despite their remarkable performance across diverse tasks, DL models do not always work reliably, and their failures can have significant real-world consequences. Model misbehavior manifests in various forms, for examples: classifiers may misclassify critical medical images due to biased training data, leading to misdiagnosis; GNNs can be manipulated through carefully crafted graph structures, potentially compromising fraud detection systems; and reasoning models may generate plausible but incorrect solutions to mathematical problems, undermining educational or scientific applications. Such failures may lead to financial losses, safety risks, privacy violations, and diminished public confidence in AI systems. Therefore, as AI models grow more powerful and pervasive, *understanding why they fail – diagnosing misbehavior – becomes critical* for safety, trust, and improvement. The challenge of diagnosing model misbehavior is further complicated by the fact that different model paradigms exhibit fundamentally different failure problems and

vulnerabilities, requiring specialized diagnostic attention and approaches tailored to their unique characteristics.

This thesis explores DL models misbehavior and addresses the challenges of diagnosing across the shifting landscape of AI paradigms, developing targeted and scalable diagnostic methods. The work encompasses three complementary projects that correspond to the major model paradigms encountered during my PhD: (1) For classifiers, we introduce the estimation of Average Marginal Effect (AME), a scalable data attribution method that traces prediction errors back to problematic training data, overcoming the computational limitations of traditional Shapley value approaches; (2) For GNNs, we develop MimicLDT, a stealthy long-distance targeted attack that reveals diagnostic blind spots in current GNN-explainer tools, and demonstrate how AME can be adapted to locate poisoned subgraphs in graph structures; (3) For large reasoning models, we design self-verification probes that uncover latent correctness signals in intermediate reasoning steps and leverage them to save tokens usage through the confidence-based early exit strategy. This thesis establishes a comprehensive framework for model-aware diagnostics that advances our understanding of DL models misbehavior across paradigms and contributes to building more trustworthy and efficient AI systems. The work demonstrates that diagnostic attention and approaches must evolve with the shifting landscape of AI models, while principled tools can transcend specific domains when their core assumptions are properly reconsidered for new contexts.

The remainder of this dissertation is organized as follows: Section 1.1 introduces the shifting landscape of deep learning models, Section 1.3 details the challenges of diagnosing model misbehavior, and Section 1.4 summarizes the contributions of this thesis. Chapters 2, 3, and 4 present the three main contributions corresponding to classifiers, GNNs, and reasoning models respectively, each containing detailed methodology, experimental validation, and analysis of results. Chapter 5 concludes with discussions on broader implications, limitations, future directions, and reflections on the evolving relationship between model paradigms and diagnostic needs.

1.1 The Shifting of Deep Learning Models

The landscape of artificial intelligence has undergone rapid and transformative changes over the past decade, fundamentally altering how we approach machine learning problems and deploy AI systems in real-world applications. During my PhD (2019–2025), the focus of AI research and deployment pivoted decisively. This chapter describes the shifted model paradigm, each bringing revolutionary capabilities while simultaneously introducing new complexities and failure modes that demand sophisticated diagnostic approaches.

From Classical Machine Learning to Deep Learning. The transition from traditional Machine Learning (ML) algorithms to Deep Neural Networks (DNNs) marked the first major paradigm shift in modern AI. Classical machine learning algorithms generally rely heavily on feature engineering approaches. Traditional algorithms like Support Vector Machines [79], Random Forests [17], and linear models required careful manual feature selection, transformation and domain expertise, while deep learning models promised to learn meaningful representations directly from raw data. These early DNNs, though revolutionary in their automatic feature learning capabilities, were primarily focused on supervised learning tasks with relatively straightforward input-output mappings.

The Deep Learning Classification Era. The period from 2012 to 2020 witnessed an explosion in deep learning applications across multiple domains, fundamentally reshaping how we approach classification problems. In computer vision, Convolutional Neural Networks (CNNs) achieved breakthrough performance on image recognition tasks, with architectures like AlexNet [107], VGGNet [160], and ResNet [77] progressively pushing the boundaries of accuracy while introducing concepts like skip connections and very deep architectures. These models demonstrated unprecedented ability to learn hierarchical visual features, from low-level edges and textures to high-level semantic concepts.

Simultaneously, the natural language processing domain experienced its own deep learning

revolution through Recurrent Neural Networks (RNNs) and their sophisticated variants. Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) addressed the vanishing gradient problem in sequential modeling, enabling effective processing of long text sequences. The introduction of word embeddings like Word2Vec and GloVe, followed by contextualized representations from models like BERT (Bidirectional Encoder Representations from Transformers) [45], marked a progression toward more nuanced understanding of language semantics.

Beyond vision and language, deep learning classifiers found applications in diverse domains including speech recognition with deep neural networks [1, 140], recommendation systems with deep collaborative filtering [134, 184], and structured data analysis with deep feedforward networks [173]. However, this proliferation of deep learning models also introduced new challenges in model interpretability and reliability [117, 170, 217]. Unlike their classical predecessors, these models operated as "*black boxes*" with millions or billions of parameters [121], making it difficult to understand why they made specific predictions or how they might fail.

The Rise of Graph Neural Networks. As AI applications expanded beyond the domains of images and text, the need to process relational and structured data became increasingly apparent. People pay more attention to Graph Neural Networks (GNNs), a powerful paradigm for handling data where relationships between entities are as important as the entities themselves. This architectural innovation was driven by the recognition that many real-world problems involve graph-structured data: social networks, molecular structures, knowledge graphs, citation networks, and transportation systems.

The foundational GNN architectures, including Graph Convolutional Networks (GCNs) [101], GraphSAGE [71] and Graph Attention Networks (GATs) [179], introduced the concept of *message passing*, where nodes iteratively aggregate information from their neighbors to update their representations. This approach enabled models to capture both local neighborhood structures and global graph properties. More sophisticated variants like Graph Isomorphism Networks (GINs) [201] addressed theoretical limitations around the expressive power of GNNs, while models like

Graph Transformer Networks [209] incorporate attention mechanisms into graph processing.

The applications of GNNs proved to be remarkably diverse: drug discovery through molecular property prediction, social media analysis through user interaction modeling, fraud detection in financial networks, and knowledge graph completion for semantic reasoning. However, this architectural sophistication came with new vulnerabilities that were not present in traditional deep learning models. We refer in detail to § 1.2.

The Large Language Model and Reasoning Revolution. The most recent and perhaps most breathtaking shift has been the emergence of Large Language Models (LLMs), which demonstrate unprecedented capabilities in natural language understanding and generation through transformer-based architectures [177] trained on vast corpora of text data. Models such as GPT-3 [19], PaLM [69], and LLaMA [172] showcased remarkable few-shot learning [19] abilities and cross-domain knowledge transfer, establishing the foundation for modern AI systems. However, despite their impressive linguistic fluency and broad knowledge base, LLMs exhibited fundamental limitations in systematic reasoning, logical consistency, and complex problem-solving tasks that require multi-step analytical processes.

The recognition of these limitations catalyzed the development of Large Reasoning Models (LRMs), such as OpenAI o1 [143] and DeepSeek-R1[44] models series, which represent the current frontier in deep learning evolution. LRMs extend beyond the pattern matching and statistical associations that characterize traditional language models, they can articulate their reasoning steps, search multiple solution approaches via backtrackings, and even verify their own reasoning solutions. Models like DeepSeek-R1 and specialized reasoning architectures have shown significantly improved performance on challenging reasoning benchmarks in mathematics, science, and logical reasoning that require multi-step analytical inference and sophisticated problem-solving strategies. These models integrate techniques such as chain-of-thought prompting [191], tool usage, retrieval-augmented generation [115], and multi-agent reasoning frameworks[56, 187] to enable more robust logical inference and complex problem-solving capabilities. This transition

from language modeling to reasoning represents a crucial step toward artificial general intelligence (AGI) [66], where models can not only understand and generate text but also engage in the kind of systematic, logical thinking that characterizes human cognitive processes.

However, this reasoning capability introduces new forms of complexity and failure modes (§ 1.2). The opacity of their internal reasoning processes, despite their ability to articulate reasoning steps, creates new challenges for understanding *when* and *why* they fail.

1.2 Important Questions among Three Model Paradigms

Despite the remarkable capabilities of modern AI systems, understanding *why they fail* and diagnosing why they misbehave – producing wrong or biased predictions, succumbing to adversarial attacks, generating hallucinations and reasoning errors, or wasting computational resources – remains a critical unsolved challenge. Crucially, the very nature of "misbehavior" as well as how to diagnose it evolve with notable shifts of AI's paradigm during my PhD journey, leading us to explore interesting questions that target different models throughout this progression.

Classifiers: Deep learning classification models, such as Convolutional Neural Networks (CNNs), Multilayer Perceptrons (MLPs), and Recurrent Neural Networks (RNNs), demonstrated remarkable capabilities in pattern recognition but were fundamentally opaque. When predictions failed, people were often left without effective tools to trace errors back to problematic training inputs. Central and pressing questions emerged: ***how do training data influence model predictions?*** In terms of the failed prediction, ***which training data points caused this specific error?*** The theoretical foundation for addressing this challenge existed in the form of Shapley value-based [157] attribution methods, which promised to quantify the contribution of individual training examples to model predictions. However, the practical application of these methods to deep learning

faced severe scalability problem. Computing exact Shapley values requires evaluating the model's performance on all possible subsets of the training data, resulting in $O(2^N)$ complexity that was computationally prohibitive for datasets containing millions of examples. This revealed a critical diagnostic gap: the need for *scalable*, *causal frameworks* to trace and audit training data influence on model (mis)behavior. The method could operate efficiently on large-scale deep learning systems while providing stable, actionable insights.

GNNs: The emergence of Graph Neural Networks (GNNs) introduced a fundamentally new class of diagnostic challenges, stemming from the intricate interplay between graph structure and model behavior. The fundamental of GNN computation lies the *message-passing* mechanism, whereby information is propagated across graph edges. In node classification tasks, this implies that nodes are ***no longer independent*** units, but part of an interdependent structure – altering a single node or edge can have cascading effects throughout the graph's topology.

This structural dependency gives rise to adversarial attack surfaces that differ substantially from those in computer vision or NLP. For example, adversaries can manipulate a target node's features or its immediate connections, or perturb within its local neighborhood. Correspondingly, most existing GNN explanation methods (i.e., GNNExplainer and PGExplainer) operate under the assumption that model behavior is locally grounded, i.e., interpretable through analysis of subgraph structures centered on the node of interest. However, it remains unclear whether adversarial manipulations must be confined to a target's local neighborhood, or ***if such attacks can be conducted in more stealthy, non-local ways***. Diagnosing these subtler manipulations is particularly difficult without controlled test cases that intentionally induce misbehavior. And question emerged: ***are current diagnostic tools (e.g., explainers) blind to novel attacks?*** These challenges are further compounded by the limitations of traditional data attribution techniques, such as Shapley values and their derivatives. These methods assume i.i.d. (independent and identically distributed) training examples that could be added or removed without affecting other data

points. In GNN settings, however, the removal or modification of a node or edge can fundamentally restructure the graph, making it unclear how to properly attribute influence.

LLMs/LRMs: Large language models and the rise of large reasoning models (e.g., DeepSeek-R1 and OpenAI o1 model series) introduced new and complex diagnostic challenges in the history of AI. First, these models are trained on vast, uncurated datasets, and their outputs are inherently non-deterministic (i.e., there are no explicit "ground-truth labels" for reasoning generation). This makes it difficult to trace individual model behaviors back to specific training data or modeling process. Nonetheless, the reasoning trajectories observed during inference may reflect patterns encoded during training, offering a valuable window into what and how these models have implicitly learned.

One perplexing challenge of reasoning models was the observed phenomenon of "overthinking," where models would continue reasoning even after generating correct intermediate answers, leading to unnecessary computation and, in some cases, arriving at incorrect final answers which degraded output quality. This raises a pressing diagnostic question: ***do reasoning models know when they reason correctly?*** This question is crucial to the success of the unique *search behavior* (characterized by backtracking to previous steps during reasoning) exclusively present in reasoning models. If internal states encode latent correctness signals, identifying and leveraging them could unlock mechanisms for early exit—enabling faster, cheaper, and potentially more reliable responses. Yet, this *latent self-awareness remains largely untapped.* Most current studies focus on final output accuracy, without probing whether intermediate reasoning steps internally flag their own validity. Additionally, without the ability to introspect on reasoning confidence, these models are susceptible to over-generation, underthinking, hallucination, and low-efficiency inference. Addressing this limitation calls for deeper interrogation into the dynamics of reasoning trajectories. Methods are needed to probe internal reasoning states and harness them for real-time self-verification, efficiency, and robustness in large reasoning models.

1.3 Challenges of Diagnosing Model Misbehaviors

1.3.1 General Challenges – The Black Box Problem

The fundamental challenge of model diagnosis lies in the inherent opacity of deep neural networks. As model complexity continues to grow – from relatively simple classifiers with millions of parameters to Transformer-based language models with hundreds of billions – their decision-making processes become increasingly opaque. This *"black-box" nature* makes it difficult to trace the causal pathways from input to output, let alone pinpoint where and why failures occur. In response to this challenge, traditional interpretability methods have focused on post-hoc explanations, which – despite being retrospective – offer valuable insights into model behavior by identifying influential features, highlighting patterns in model outputs, and helping people build intuition about how the model operates.

1.3.2 Other Challenges

Here, we outline several broader challenges that continue to hinder scalable and reliable diagnosis across model types and task domains.

Scalability challenges: Modern AI systems operate at unprecedented scales that introduce significant practical challenges for diagnosis. Training datasets containing millions or billions of examples make it computationally prohibitive to exhaustively analyze data influence. Many diagnostic methods were developed in the era of small models and curated datasets, but often break down when applied to large-scale architectures or web-scale corpora. Moreover, the computational cost of running comprehensive diagnostic procedures can be enormous.

Architecture-specific limitations: Diagnostic tools are often tightly coupled to the assumptions of a specific model paradigm. Methods developed for one generation of models may become obsolete as new paradigms emerge – what works for CNNs may not apply to GNNs; what ap-

plies to classification may not translate to generative settings. This lack of generalization creates a perpetual arms race where diagnostic methods must constantly evolve to keep pace with rapidly advancing model architectures.

Evaluation and Ground Truth Challenges: Effective diagnosis requires clear metrics for success and failure, yet defining ground truth for model behavior can be surprisingly difficult. In classification tasks, labels may be noisy or subjective. In generation tasks, multiple valid outputs may exist, making it unclear when a model has truly "failed." For reasoning tasks, evaluating correctness may require sophisticated verification procedures. This evaluation challenge compounds the diagnostic problem, and highlights the need for more precise definitions of failure – pushing researchers to refine how diagnostic tasks are framed and interpreted.

1.4 Contributions

We argue that diagnosing why models fail is essential for building AI systems that are safe, trustworthy, and improvable. Yet, understanding model misbehavior remains critical challenges. As AI research and development shifted over the course of my PhD, this thesis follows that trajectory – exploring the misbehavior and developing targeted, scalable diagnostic tools along the way. This thesis makes three key contributions:

1.4.1 Classifiers – Scalable Data Attribution

The first contribution of this thesis introduces a novel framework for diagnosing deep learning classifiers by attributing model behavior to individual training data points. Specifically, we develop the *Average Marginal Effect (AME)*, a principled and scalable data attribution method that addresses a fundamental challenge in the classifier era: understanding how individual training examples influence model predictions and tracing prediction errors back to their training data origins. Unlike existing influence function [102] approaches that suffer from computational

complexity (requiring $O(N^2)$ operations for N training points), AME leverages causal inference principles and randomized experiments to achieve efficient attribution with only $O(klogN)$ evaluations under sparsity assumptions.

The key innovation lies in AME's experimental design: rather than computing expensive second-order derivatives, we train multiple submodels on different random subsets of the training data and use LASSO regression to estimate each data point's marginal contribution. This approach not only scales to large datasets but also provides more reliable estimates than traditional Shapley value [157] methods, which can be prohibitively expensive and unstable for deep networks.

Our extensive evaluation across multiple domains – including data poisoning detection, prediction explanation, and Shapley value estimation – shows that AME consistently outperforms existing methods while requiring orders of magnitude fewer computations.

This contribution advances the interpretability and robustness of deep learning models by enabling fine-grained, query-specific diagnosis of model behavior through data attribution.

1.4.2 GNNs - Vulnerability & Poisoning Diagnosis

The second contribution of this thesis addresses the diagnostic challenges that emerged as AI systems evolved toward Graph Neural Networks (GNNs), where the interdependent nature of graph data in node classification tasks renders traditional diagnostic methods inadequate. This project consists of two complementary parts that together demonstrate how diagnostic frameworks must adapt to new model paradigms.

First, we develop *MimicLDT*, a novel long-distance targeted poisoning attack for the node classification task, that reveals critical blind spots in GNN robustness. Unlike existing attacks that manipulate nodes within the target's immediate k-hop neighborhood, MimicLDT demonstrates that attackers can successfully poison GNN predictions by injecting malicious nodes and edges far from the target and connecting them through long-distance paths. This attack is particularly

stealthy because it operates outside regions typically monitored by existing local explainability tools (e.g., GNNExplainer [206]), exposing fundamental assumptions in current GNN diagnostic mechanisms that focus primarily on local perturbations.

Second, we adapt our AME framework to diagnose this graph-based stealthy attack, demonstrating the cross-domain applicability of principled diagnostic tools. The key challenge lies in adapting AME's independence assumption to graph data, where nodes exhibit complex dependencies. Our solution modifies the sampling strategy to work with conditionally independent subgraphs rather than individual nodes, treating labeled nodes and their neighborhoods as fundamental attribution units. This graph-adapted AME successfully diagnoses MimicLDT attacks with perfect recall (1.0) while maintaining reasonable precision (0.739), enabling complete elimination of poisoning effects once attack points (i.e., perturbed subgraphs) are identified.

This contribution forms an end-to-end pipeline – from attack to diagnosis to mitigation – revealing structural vulnerabilities in GNNs while pioneering new directions for interpretable graph learning. The adaptation of AME from independent data points (classifiers) to dependent graph structures (GNNs) demonstrates that principled tools can transcend specific domains when their core assumptions are properly reconsidered for new contexts.

1.4.3 Reasoning Models - Probing for Self-Verification & Efficient Inference

The third contribution of this thesis develops a novel diagnostic framework for large reasoning models by uncovering and leveraging their latent *self-verification capabilities*. This work investigates whether reasoning models encode information about the correctness of their intermediate answers in their hidden states, and whether such information can be extracted and used to guide inference-time decisions.

The key innovation lies in our self-verification probe design: we parse long Chain-of-Thought

(CoT) reasoning into chunks that end with intermediate answers, extract hidden state representations at each reasoning chunk, and train a lightweight two-layer MLP to predict answer correctness from these internal representations. This approach reveals that reasoning models implicitly encode correctness information, with our probe achieving high accuracy and well-calibrated (e.g., ECE < 0.1) confidence scores. The probe generalizes across reasoning datasets (e.g., GSM8K, MATH, AIME, KnowLogic), and reveals that correctness signals are present even before an intermediate answer is fully generated – suggesting a form of "look-ahead" self-awareness in model representations.

Our work reveals that reasoning models exhibit internal awareness of their own correctness, opening new possibilities for computational efficiency. We demonstrate that this latent correctness information can be effectively leveraged through a *confidence-based early-exit* strategy, achieving up to 24% reduction in inference tokens without compromising accuracy, outperforming static early-exit baselines. Unlike traditional approaches that rely on external verifiers, our method exploits the model's intrinsic signals encoded during the reasoning process itself.

The findings highlight that reasoning models possess latent self-verification abilities that are *underutilized* during inference. This opens new directions for *on-policy* control and adaptive reasoning strategies that leverage internal model signals for more efficient and reliable reasoning.

This contribution bridges interpretability and efficiency, showing that reasoning models not only "know when they're right," but can be guided to act on that knowledge, paving the way for more introspective and resource-aware AI systems.

2 | Diagnosing Classifiers: From Data Attribution

This chapter presents measuring the effect of training data on deep learning predictions via randomized experiments.

We develop a new, principled algorithm for estimating the contribution of training data points to the behavior of a deep learning model, such as a specific prediction it makes. Our algorithm estimates the AME, a quantity that measures the expected (average) marginal effect of adding a data point to a subset of the training data, sampled from a given distribution. When subsets are sampled from the uniform distribution, the AME reduces to the well-known Shapley value. Our approach is inspired by causal inference and randomized experiments: we sample different subsets of the training data to train multiple submodels, and evaluate each submodel's behavior. We then use a LASSO regression to jointly estimate the AME of each data point, based on the subset compositions. Under sparsity assumptions ($k \ll N$ datapoints have large AME), our estimator requires only $O(k \log N)$ randomized submodel trainings, improving upon the best prior Shapley value estimators.

2.1 Overview

Machine Learning (ML) is now ubiquitous, with black-box models such as deep neural networks (DNNs) powering an ever increasing number of applications, yielding social and economic benefits. However, these complex models are the result of long, iterative training procedures over large amounts of data, which make them hard to understand, debug, and protect. As an important first step towards addressing these challenges, we must be able to solve the problem of *data attribution*, which aims to pinpoint training data points with significant contributions to specific model behavior. There are many use cases for data attribution: it can be used to assign value to different training data based on the accuracy improvements they bring [92, 104], explain the source of (mis)predictions [11, 102], or find faulty data points resulting from data bugs [23] or malicious poisoning [155].

Existing principled approaches to explain how training data points influence DNN behavior either measure Influence functions [102] or Shapley values [65, 92]. Influence provides a local explanation that misses complex dependencies between data points as well as contributions that build up over time during training [11]. While Shapley values account for complex dependencies, they are prohibitively expensive to calculate: exact computation requires $O(2^N)$ model evaluations, and the best known approximation requires $O(N \log\log N)$ model evaluations, where N is the number of data points in the training set [92].

In this paper, we propose a new, principled metric for data attribution. Our metric, called AME, measures the contribution of each training data point to a given behavior of the trained model (e.g., a specific prediction, or test set accuracy). AME is defined as the expected marginal effect contributed by a data point to the model behavior over randomly selected subsets of the data. Intuitively, a data point has a large AME when adding it to the training data affects the behavior under study, regardless of which other data points are present. We show that the AME can be efficiently estimated using a carefully designed LASSO regression under the sparsity as-

sumption (i.e., there are $k \ll N$ data points with large AME values). In particular, our estimator requires only $O(k \log N)$ evaluations, which makes it practical to use with large training sets. When using AME to detect data poisoning/corruption, we also extend our estimator to provide control over the false positive rate using the Knockoffs method [21].

When the size of subsets used by our algorithm is drawn uniformly, the AME reduces to the Shapley value (SV). As a result, our AME estimator provides a new method for estimating the SVs of all training data points; under the same sparsity and monotonicity assumptions, we obtain a better rate than the previous state-of-the-art [92]. Interestingly, our causal framing also supports working with groups of data points, which we call data sources. Many datasets are naturally grouped into sources, such as by time window, contributing user, or website. In this setting, we extend the AME and our estimator to support hierarchical estimation for nested data sources. For instance, this enables joint measurement of both users with large contributions, and the specific data points that drive their contribution.

We empirically evaluate the AME quantity and our estimator's performance on three important applications: detecting data poisoning, explaining predictions, and estimating the Shapley value of training data. For each application, we compare our approach to existing methods.

In summary, we make the following contributions:

- We propose a new quantity for the data attribution problem, AME, with roots in randomized experiments from causal inference (§2.2). We also show that SV is a special case of AME.
- We present an efficient estimator for AME with an $O(k \log N)$ rate under sparsity assumptions (§2.3). This also yields an $O(k \log N)$ estimator for sparse and monotonic SV, a significant improvement over the previous $O(N \log \log N)$ state-of-the-art [92].
- We extend the AME and our estimator to control false discoveries (§2.4.1) and support hierarchical settings of nested data sources (§2.4.2).

2.2 Our Metric for Data Attribution: AME

At a high level, our goal is to understand the impact of training data on the behavior of a trained ML classification model, which we call a query. Queries of interest include a specific prediction, or the test set accuracy of a model. Below, we formalize our setting (§2.2.1) and present our metric for quantifying data contributions (§2.2.2).

2.2.1 Notations

Let $\mathcal{M}_S$ denote the classification model trained on dataset S that we wish to analyze. In the rest of the paper, we refer to this as the *main model*. We note that $\mathcal{M}_S$ belongs to a class of models $\mathcal{M}$ (i.e., $\mathcal{M}_S \in \mathcal{M}$) with the same architecture, but trained on different datasets. $\mathcal{M}_S$ maps each example from the input space $\mathcal{X}$ to a normalized score in $[0, 1]$ for each possible class $\mathcal{Y}$, i.e., $\mathcal{M}_S : \mathcal{X} \mapsto [0, 1]^{|\mathcal{Y}|}$. Since the normalized scores across all classes sum to one, they are often interpreted as a probability distribution over classes conditioned on the input data point, giving a confidence score for each class.

Let $Q(\mathcal{M}_S)$ be the query resulting in a specific behavior of $\mathcal{M}_S$ that we seek to explain. Formally, $Q : \mathcal{M} \mapsto [0, 1]$ maps a model to a score in $[0, 1]$ that represents the behavior of the model on that query. For example, we may want to explain a specific prediction, i.e., the score for label l given to an input data point n, in which case $Q(\mathcal{M}_S) = \mathcal{M}_S(n)[l]$; or, we may want to explain the accuracy on a test set with inputs I and corresponding labels L, in which case $Q(\mathcal{M}_S) = \frac{1}{|I|} \sum_{(n,l)\in(I,L)} \mathbf{1}\{\arg\max[\mathcal{M}_S(n)] = l\}$. Our proposed metric and estimator apply to any query, but our experiments focus on explaining specific predictions.

We use the query score $Q(\mathcal{M}_S)$ to represent the utility of training data set S, i.e., $U(S) \triangleq Q(\mathcal{M}_S)$. When describing our technique, we will need to calculate the utility of various training data subsets, each of which is the result of the query Q when applied to a model trained on a subset S' of the data. Note that any approach for estimating the utility $U(S')$ may be noisy due

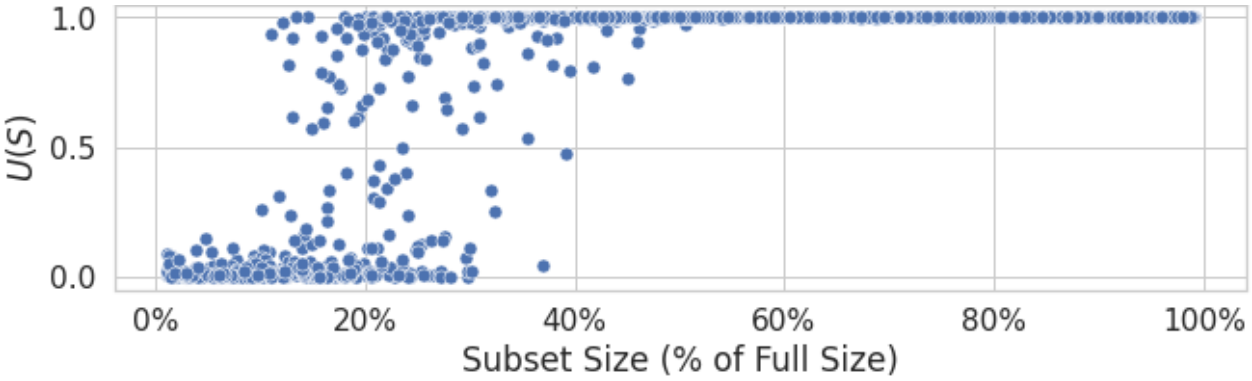


Figure 2.1: Utility vs. the subset size, measured on CIFAR10-50 dataset (see §2.5), where each point denotes a subset. Each subset is obtained by first drawing an inclusion probability p from a uniform distribution with range from 0.01 to 0.99, and then including each datapoint with probability p.

to the randomness in model training.

2.2.2 Defining the Average Marginal Effect (AME)

How do we quantify the contribution of a training data point n to the query result? One approach, commonly referred to as the *influence* of n, defines the contribution of n as $U(S) - U(S\backslash\{n\})$, the marginal contribution of the data point when added to the rest of the data. This quantity can be calculated efficiently using an approach presented in [102]. However, in practice, the marginal effect of a datapoint on the whole training set of an ML model is typically close to zero, a well-known shortcoming of influence [11], which we confirm empirically in Fig. 2.1. (We compare influence functions to our proposal in more detail in Appendix A.6.5.3.) The figure shows results from a data poisoning experiment run on the CIFAR10 dataset. It plots the utility of models trained on various random subsets of the poisoned training dataset. The utility is calculated as the score given to the wrongly predicted label for a poisoned test point. As we can see, removing up to half of the training data at random has no impact on the utility, implying a close to zero influence of each training example on a model trained on the full training set.

To alleviate this issue, we notice that at least some training data points have to influence the utility (which goes from zero on very small subsets to one on large ones). This influence happens

on smaller subsets of the training data, around a size unknown in advance (between 10% and 50% of the whole dataset size in Fig. 2.1's example). Taking inspiration from the causal inference literature on measuring multiple treatment effects [50], we thus propose to *average* the marginal contribution of adding data point n to data subsets of different sizes. We refer to this as the data point's *Average Marginal Effect (AME)*, defined as the expected marginal effect of n on subsets drawn from a distribution $\mathcal{L}^n$: $\mathbb{E}_{S^n \sim \mathcal{L}^n}[U(S^n + \{n\}) - U(S^n)]$. Here S^n is a subset of training data points that do not contain n, sampled from $\mathcal{L}^n$. The marginal effect of n with respect to S^n is calculated as the difference in the query result on a model trained with and without n, i.e., $U(S^n + \{n\}) - U(S^n)$.

Clearly, the choice of sampling distribution $\mathcal{L}^n$ affects what AME is measuring, and how efficiently AME can be estimated (§**??**). When choosing $\mathcal{L}^n$, we need to ensure that subsets of different sizes are well represented. To see why, consider Fig. 2.1 again: since the region with non-zero marginal effect is unknown in advance, we must sample subsets across different subset sizes. We hence propose to sample subsets by including each data point (except for data point n being measured) with a probability p sampled from a distribution $\mathcal{P}$ that ensure coverage across subset sizes (e.g., we use a uniform distribution over a grid of values $\mathcal{P} = Uni\{0.2, 0.4, 0.6, 0.8\}$ in most experiments). Denoting $\mathcal{L}^n_{\mathcal{P}}$ as the subset distribution induced by $\mathcal{P}$, we have:

$$\text{AME}_n(\mathcal{P}) = \mathbb{E}_{S^n \sim \mathcal{L}^n_{\mathcal{P}}}[U(S^n + \{n\}) - U(S^n)]. \tag{2.1}$$

In what follows, we use the shorthand AME_n for $\text{AME}_n(\mathcal{P})$ when $\mathcal{P}$ is clear from context.

2.2.3 Connection to the Shapley Value

Interestingly, pushing the above proposal further and sampling p uniformly over $[0, 1]$ reduces the AME to the Shapley value (SV), a well known but costly to estimate metric from game theory that has been proposed as a measure for data value [65, 92]:

Proposition 2.1. $\mathcal{P} = Uni(0,1) \implies AME_n(\mathcal{P}) = SV_n$.

Proof. When p is fixed, the subset size follows a binomial distribution with $N-1$ trials and probability of success p. When $p \sim \text{Uni}(0,1)$, the compound distribution is a beta-binomial with $\alpha = \beta = 1$, and the subset size follows a discrete uniform distribution, each subset size having a probability of $1/N$. Since by symmetry each possible subset of a given size is equally likely, $AME_n = \sum_{S^n \subseteq [N] \setminus \{n\}} \frac{1}{N} \binom{N-1}{|S^n|}^{-1} (U(S^n + \{n\}) - U(S^n))$ which is precisely the definition of Shapley value SV_n. □

In concurrent work, [110] also highlight this relationship and propose Beta(α, β)-Shapley as a natural and practically useful extension to the SV, enabling variable weighting of different subset sizes to integrate domain knowledge. The AME can be seen as a generalization of Beta Shapley, which corresponds to AME($\mathcal{P}$) with $\mathcal{P} = \text{Beta}(\alpha, \beta)$. In this work, we focus on a discrete grid for $\mathcal{P}$, but also study the symmetric Beta and truncated uniform distributions as SV approximations (§??).

This connection between AME and Beta-Shapley also yields two new insights. First, Equation 4 and Theorem 2 of [110] imply that the AME is a semivalue. That is, it satisfies three of the SV axioms: linearity, null player, and symmetry, but not the efficiency axiom (i.e., the AME_n do not sum to $U(S)$). Second, our AME estimator yields a scalable estimator for the Beta($\alpha > 1, \beta > 1$)-Shapley values of a training set (using $\mathcal{P} = \text{Beta}(\alpha, \beta)$), answering a question left to future work in [110].

2.3 Efficient Sparse AME Estimator

Computing the AME exactly would be costly, as it requires computing $U(S)$ for many different data subsets S, and each such computation requires training a model $\mathcal{M}_S$ to evaluate the query $Q(\mathcal{M}_S)$. Furthermore, measurements of $Q(\mathcal{M}_S)$ are noisy due to randomness in model training, and can require multiple samples. However, for the use cases we target (§2.1), we expect that data

points with large AMEs will comprise only a *sparse* subset of the training data for a given query Q. Hence, for the rest of this paper, we make the following strong sparsity assumption:

Assumption 2.2. *Let k be the number of data points with non-zero AME's. k is small compared to N, or $k \ll N$.*

All results in this section (§2.3) hold under a weaker, approximate sparsity assumption: that there exists a good sparse approximation to the AME. However, the results are cumbersome to state without adding much intuition, so we defer the details of this setting to Appendix A.5. In practice, the sparsity assumption (and the relaxed version to a stronger degree) holds for use cases such as corrupted data detection, which typically impacts only a small portion of the training data; or when the predictions under scrutiny arise from queries on the tails of the distribution, which are typically strongly influenced by only a few examples in the training data [54, 65, 92].

Under this assumption, we can efficiently estimate the AME of each training data point with only $O(k \log(N))$ utility computations, by leveraging a reduction to regression and LASSO based estimation (§2.3.1). We then characterize the error in estimating Shapley values using this approach (§2.3.2), and show that under a common monotonicity assumption, our estimator achieves small L_2 errors.

2.3.1 A Sparse Regression Estimator for the AME

Our key observation is that we can re-frame the estimation of all AME_n's as a specific linear regression problem. While a regression-based estimator for the SVs is known [126, 194], it is based on a weighted regression with constraints. Instead, we propose a featurization-based regression formulation without weights or constraints, which enables efficient estimation under sparsity using LASSO, a regularized linear regression method.

Regression formulation. To compute the AME_n values, we begin by producing M subsets of the training data, $S_1, S_2, ..., S_M$. Each subset S_m is sampled by first selecting a p (drawn from

Algorithm 1: Overall Workflow

Input: number of data points N, number of subsets M to draw, probabilities $\mathcal{P} = Uni\{p_1, \ldots, p_b\}$, query Q

```
// offline phase
M_S, X ← sampleSubsets(M)
// online phase
while Q ← new query do
    β̂_lasso ← estimate(M_S, X, Q, M)

Function sampleSubsets(M):
    M_S ← [] // subset models
    X ← zeros(M, N) // source covariates
    for m ← 1 to M do
        S ← {}
        p ~ P
        for n ← 1 to N do
            r ~ Bernoulli(p)
            if r = 1 then S ← S + {n}
            X[m, n] ← r/p − (1−r)/(1−p)
        M_S.append(trainOnSubset (S))
    return M_S, X

Function estimate(M_S, X, Q, M):
    y ← zeros(M) // outcome vector
    for m ← 1 to M do
        y[m] ← Q(M_S[m]) // inference
    return β̂_lasso ← LASSO(X, y, λ) // λ is chosen by cross validation.
```

$\mathcal{P}$) and then including each training data point with probability p. Observation $\mathbf{X}$ is a $M \times N$ matrix, where row $\mathbf{X}[m, :]$ consists of N features, one for each training data point, to represent its presence or absence in the sampled subset S_m. y is a vector of size M, where $y[m]$ represents the utility score measured for the sampled subset S_m, i.e., $y[m] = U(S_m) = Q(\mathcal{M}_{S_m})$.

How should we design $\mathbf{X}$ (i.e., craft its features) such that the fit found by linear regression, β^*, corresponds to the AME? Let us first examine the simple case where subsets are sampled using a fixed p. In this case, we can set $\mathbf{X}[m, n]$ to be +1 when data point n is included in S_m and -1 otherwise. Intuitively, because all data points are assigned to the subset models independently, features $\mathbf{X}[:, n]$ do not "interfere" in the regression and can be fitted together, re-using

computations of $U(S_m)$ across training data points n.

Supporting different values of p (each row's subset is sampled with a different probability) is more subtle, as the different probabilities of source inclusion induce both a dependency between source variables $\mathbf{X}[:, n]$, and a variance weighted average between ps, whereas AME_n is defined with equal weights for each p. To address this, we use a featurization that ensures that variables are not correlated, and re-scales the features based on p to counter-balance the variance weighting. Concretely, for each observation (row) $\mathbf{X}[m, :]$ in our final regression design, we sample a p from $\mathcal{P}$, and sample S_m by including each training data point independently with probability p. We set $\mathbf{X}[m, n] = \frac{1}{\sqrt{v}p}$ if $n \in S_m$ and $\mathbf{X}[m, n] = \frac{-1}{\sqrt{v}(1-p)}$ otherwise; where $v = \mathbb{E}[\frac{1}{p(1-p)}]$ is the normalizing factor ensuring that the distribution of $X[m, n]$ has unit variance. Algorithm 1, `sampleSubsets()`, summarizes this. In what follows, we use X to denote the random variables from which the $\mathbf{X}[m, :]$'s are drawn (since each row is drawn independently from the same distribution), subscripts X_n to denote the random variable for feature n (i.e., from which $\mathbf{X}[m, n]$ is drawn), and Y for the random variable associated with $y[m]$. Under our regression design, we have that:

Proposition 2.3. *Let β^* be the best linear fit on (X, Y):*

$$\beta^* = \underset{\beta \in \mathbb{R}^N}{\arg\min}\, \mathbb{E}\left[(Y - \langle \beta, X \rangle)^2\right], \tag{2.2}$$

then $AME_n/\sqrt{v} = \beta^*_n,\ \forall n \in [N]$, where $v = \mathbb{E}_p[\frac{1}{p(1-p)}]$.

Proof. For a linear regression, we have (see, e.g., Eq. 3.1.3 of [4]): $\beta^*_n = \frac{Cov(Y, \tilde{X}_n)}{Var[\tilde{X}_n]}$, where $\tilde{X}_n$ is the regression residual of X_n on all other covariates $X_{-n} = (X_1, \ldots, X_{n-1}, X_{n+1}, \ldots, X_N)$. By design, $\mathbb{E}[X_n | X_{-n}] = \mathbb{E}_p[X_n | p] = 0$, implying $\tilde{X}_n = X_n - \mathbb{E}[X_n | X_{-n}] = X_n$. Therefore:

$$\beta^*_n = \frac{Cov(Y, \tilde{X}_n)}{Var[\tilde{X}_n]} = \frac{Cov(Y, X_n)}{Var[X_n]} = \frac{\mathbb{E}[X_n Y]}{Var[X_n]}.$$

Notice that $\mathbb{E}[X_n Y] = \mathbb{E}_p[\mathbb{E}[X_n Y|p]]$ with:

$$\begin{aligned}
&\mathbb{E}[X_n Y|p] \\
&= p \cdot \mathbb{E}[X_n Y|p, n \in S] + (1-p) \cdot \mathbb{E}[X_n Y|p, n \notin S] \\
&= p\frac{1}{\sqrt{v}p}\mathbb{E}[Y|p, n \in S] + (1-p)\frac{-1}{\sqrt{v}(1-p)}\mathbb{E}[Y|p, n \notin S] \\
&= \frac{1}{\sqrt{v}}(\mathbb{E}[Y|p, n \in S] - \mathbb{E}[Y|p, n \notin S])
\end{aligned}$$

Combining the two previous steps yields:

$$\beta_n^* = \frac{\mathbb{E}_p[\mathbb{E}[Y|p, n \in S] - \mathbb{E}[Y|p, n \notin S]]}{Var[X_n] \cdot \sqrt{v}} = \frac{AME_n}{Var[X_n] \cdot \sqrt{v}}.$$

Noticing that $Var[X_n] = \mathbb{E}[Var[X_n|p]] + Var[\mathbb{E}[X_n|p]] = \mathbb{E}[\frac{1}{p(1-p)}]/v = 1$ concludes the proof.

□

Proposition 2.3 shows that, by solving the linear regression of y on $\mathbf{X}$ with infinite data, β_n, the linear regression coefficient associated with X_n, becomes equal to the AME_n we desire re-scaled by a known constant. Of course, we do not have access to infinite data. Indeed, each row in this regression comes from training a model on a subset of the original data, so limiting their number (M) is important for scalability. Ideally, this number would be smaller than the number of features (the number of training data points N), even though this leads to an under-determined regression, making existing regression based approaches [126, 194] challenging to scale to large values of N. Fortunately, in our design we can still fit this under-determined regression by exploiting sparsity and LASSO.

Efficient estimation with LASSO. To improve our sample efficiency and require fewer subset models for a given number of data points N, we leverage our sparsity assumption and known results in high dimensional statistics. Specifically, we use a LASSO estimator, which is a linear

regression with an L_1 regularization:

$$\hat{\beta}_{lasso} = \arg\min_{\beta \in \mathbb{R}^N} \left((y - \langle \beta, \mathbf{X} \rangle)^2 + \lambda \|\beta\|_1 \right).$$

LASSO is sample efficient when the solution is sparse [112]. Recall that k is the number of non-zero AME_n values, and M is the number of subset models. Our reduction to regression in combination with a result on LASSO's signal recovery lead to the following proposition:

Proposition 2.4. *If X_n's are bounded in $[A, B]$, $N \geq 3$ and $M \geq k(1 + \log(N/k))$, there exist a regularization parameter λ and a constant $C(B - A, \delta)$ such that*

$$\|\hat{\beta}_{lasso} - \frac{1}{\sqrt{v}} AME\|_2 \leq C(B - A, \delta) \sqrt{\frac{k \log(N)}{M}}$$

holds with probability at least $1 - \delta$, where $v = \mathbb{E}_p[\frac{1}{p(1-p)}]$.

Proof. We provide a proof sketch here. From Proposition 2.3, we know that $\frac{\text{AME}}{\sqrt{v}}$ is the best linear estimator of the regression of Y on X. Applying Theorem 1.4 from [112] directly yields the error bound. The bulk of the proof is showing that our setting satisfies the assumptions of Theorem 1.4, which we argue in Appendix A.1.2. □

As a result, LASSO can recover all AME_n's with low L_2 error ε using $M = C(B-A, \delta)^2/\varepsilon^2 vk \log(N) = O(k \log(N))$ subest models. Eliminating a linear dependence on the number of data points (N) is crucial for scaling our approach to large datasets.

2.3.2 Efficient Sparse SV Estimator

Following Prop. 2.1, it is tempting to estimate the SV using our AME estimator by sampling $p \sim Uni(0, 1)$. However, Prop. 2.4 would not apply in this case, because the X_n's are unbounded due to our featurization. Indeed $v = \infty$ when p is arbitrarily close to 0 and 1. We address this

problem by sampling $p \sim Uni(\varepsilon, 1-\varepsilon)$, truncating the problematic edge conditions. While this solves our convergence issues, it leads to a discrepancy between the SV and AME.

Under such a truncated uniform distribution for p, we can show that $|\text{SV}_n - \text{AME}_n|$ is bounded, and applying our AME estimator yields the following L_∞ bound when the AME is sparse (details in Appendix A.3.2, and the intuition behind the proof is similar to that of Corollary 2.8):

Corollary 2.5. *When $AME_n \in [0,1]$, for every constants $\varepsilon > 0, \delta > 0, N \geq 3$, there exist constants $C_1(\varepsilon, \delta)$, ε', and a LASSO regularization parameter λ, such that when the number of samples $M \geq C_1(\varepsilon, \delta) k \log N$, $\|\sqrt{v}\hat{\beta}_{lasso} - SV\|_\infty \leq \varepsilon$ holds with a probability at least $1-\delta$, where $v = \mathbb{E}_{p \sim Uni(\varepsilon', 1-\varepsilon')}[\frac{1}{p(1-p)}]$.*

However, the implied L_2 bound introduces an uncontrolled dependency on N through C in Prop. 2.4, or a k term even when the SV is also sparse. To achieve an L_2 bound, we focus on a sparse and *monotonic* SV. Monotonicity is a common assumption [92, 146], under which adding training data never decreases the utility score:

Assumption 2.6. *Utility function $U(\cdot)$ is said to be monotone if for each $S, T, S \subseteq T : U(S) \leq U(T)$.*

Under this monotonicity assumption and a sparsity assumption, prior work has obtained a rate of $O(N \log\log N)$ for estimating the SV under an L_2 error [92]. Here, we show that we can apply our AME estimator to yield an $O(k \log N)$ rate in this setting, a vast improvement over the previous linear dependency on the number of data points N. To prove this result, we start by bounding the L_2 error between the AME and SV with the following:

Lemma 2.7. *If $p \sim Uni(\varepsilon, 1-\varepsilon)$, $\|AME - SV\|_2 \leq 4\varepsilon + 2\sqrt{2\varepsilon}$.*

We prove this lemma in Appendix A.3.2. Crucially, the error only depends on ε, and remains invariant when N and k change. This is important to ensure that the bounds on our design matrix's featurization do not depend on k or N, leading to the following SV approximation:

Corollary 2.8. *For every constant $\varepsilon > 0, \delta > 0, n \geq 3$, there exists constants $C_1(\varepsilon, \delta)$, ε', and a LASSO regularization parameter λ, such that when the number of samples $M \geq C_1(\varepsilon, \delta) k \log N$, $\|\sqrt{v}\hat{\beta}_{lasso} - SV\|_2 \leq \varepsilon$ holds with probability at least $1 - \delta$, where $v = \mathbb{E}_{p \sim Uni(\varepsilon', 1-\varepsilon')}[\frac{1}{p(1-p)}]$.*

Proof. $\|\sqrt{v}\hat{\beta}_{lasso} - \text{SV}\|_2 \leq \|\sqrt{v}\hat{\beta}_{lasso} - \text{AME}\|_2 + \|\text{AME} - \text{SV}\|_2$. By noticing that a monotonic and k-sparse SV implies a k-sparse AME with $p \sim Uni(\varepsilon', 1 - \varepsilon')$ (details in Corollary A.7 of the Appendix), we apply Proposition 2.4 to bound the first term by $\varepsilon/2$, and Lemma 2.7 with $\varepsilon' = \frac{1}{8}(\sqrt{\varepsilon + 1} - 1)^2$ to bound the second term by the same, concluding the proof. □

In the Appendix, we study different featurizations for our design matrix (A.2, A.3.2) and using a Beta distribution for p (A.3.3), which yield the same error rates as Corollary 2.8 and may be of independent interest. Empirically, we found that the truncated uniform distribution with the alternative featurization yield the best results for SV estimation (see Appendix A.6.8). Interestingly, this different featurization directly yields a regression estimator with good rates under sparsity assumptions for Beta(α, β)-Shapley when $\alpha > 1$ and $\beta > 1$, the setting considered in [110] (details in Appendix A.4).

2.4 Practical Extensions

When estimating the AME in practice, training $\mathcal{M}_S$ is the most computationally expensive step of processing a query $Q(\mathcal{M}_S)$. However, since the sampled subsets S do not depend on the query Q, we can precompute our subset models *offline*, and re-use them to answer multiple queries (e.g., for explaining multiple mispredictions). This yields the high-level workflow shown in Fig. 2.2, which is summarized in Alg. 1 (lines 1-3). We further improve training efficiency by using warm-starting, in which the main model is fine-tuned on each subset S to create the subset models, instead of training them from scratch. Warm-starting has implications on our choice of $\mathcal{P}$, as discussed in Appendix A.6.1.

We now develop two techniques that improve the practicality of our approach, by allowing

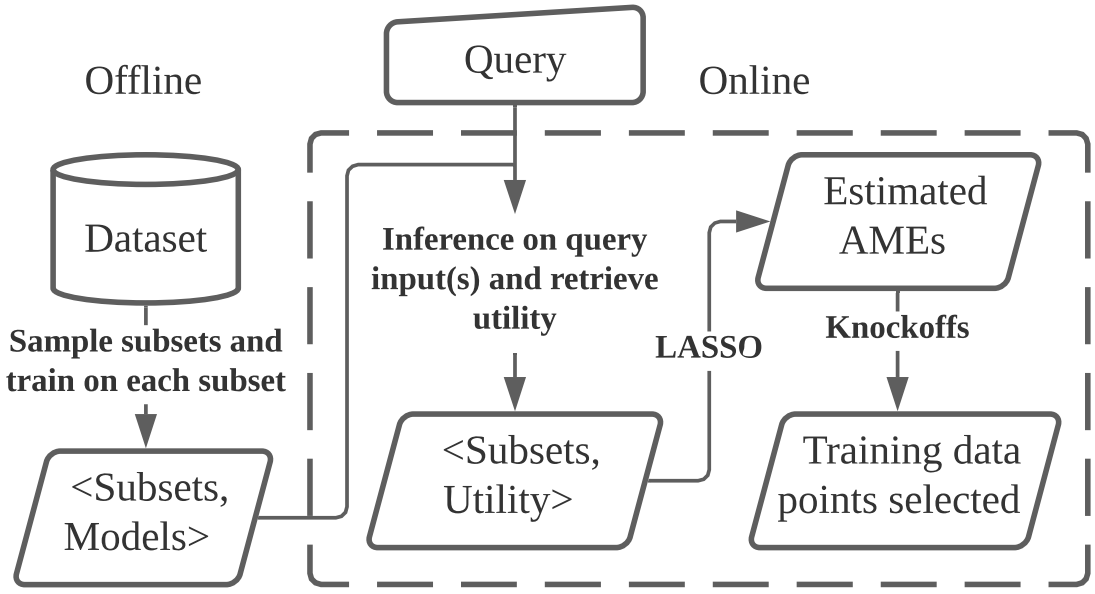


Figure 2.2: Estimation Workflow

us to control the false discovery rate (§2.4.1), and allowing us to leverage hierarchical data for more efficient, multi-level analysis (§2.4.2).

2.4.1 Controlling False Discoveries

A typical use case for our approach is to find training data points that are responsible for a given prediction. Following [149], We refer to such data points as *proponents*, and define them as those having AME > 0. Data points with AME < 0 are referred to as *opponents*, and the rest are *neutrals*.

Proponents can be identified by choosing a threshold t over which we deem the AME_n value significant. Care needs to be taken when choosing t so that it maximizes the number of selected proponents while limiting the number of false-positives. Formally, if $\hat{S}_+$ are the data points selected and S_+ is the true set of proponents, then $precision \triangleq \frac{|\{n \in \hat{S}_+ \cap S_+\}|}{|\hat{S}_+|}$. We therefore need to choose a t that can control the *false discovery rate* (FDR): $\mathbb{E}[1 - precision]$.

To this end, we adapt the Model-X (MX) Knockoffs framework [21] to our setting. In our

regression design, we add one-hot ("dummy") features for the value of p, and for each X_n we add a knockoff feature sampled from the same conditional distribution (in our case, the features encoding p). Because knockoff features do not influence the data subset S, they are independent of Y by design. We then compare each data point's coefficient $\hat{\beta}_n$ to the corresponding knockoff coefficient β'_n to compute W_n:

$$W_n \triangleq \max(\hat{\beta}_n, 0) - \max(\beta'_n, 0).$$

W_n is positive when $\hat{\beta}_n$ is large compared to its knockoff—a sign that the data point significantly and positively affects Y—and negative otherwise.

Finally, we compute the threshold t such that the estimated value of $1 - \textit{precision}$ is below the desired FDR q:

$$t = \min\left\{\tau > 0 : \frac{\#\{n : W_n \leq -\tau\}}{\#\{n : W_n \geq \tau\}} \leq q\right\},$$

and select data points with a W_n above this threshold. We use $\hat{S}_+$ to denote selected data points.

Under the assumption that neutral data points are *independent* of the utility conditioned on p and other data points—that is, $U(S) \perp\!\!\!\perp \mathbf{1}\{n \in S\} \,|\, \left((\mathbf{1}\{j \in S\})_{j \neq n}, p\right)$, we control the following relaxation of FDR [21]:

$$mFDR = \mathbb{E}\left[\frac{\left|\left\{n \in (\hat{S}_+ \cap S_+)\right\}\right|}{|\hat{S}_+| + 1/q}\right] \leq q.$$

Although there exists a knockoff variation controlling the exact FDR, this relaxed guarantee works better when there are few proponents and does well in our experiments (§2.5).

2.4.2 Hierarchical Design

Our methodology can be extended so it leverages naturally occurring hierarchical structure in the data, such as when data points are contributed by users, to improve scalability. By changing our sampling algorithm, LASSO inputs, and knockoffs design, we can support proponent

detection at each level of the hierarchy using a single set of subset models. As §2.5 shows, this approach significantly reduces the number of subset models required, with gracefully degrading performance along the data hierarchy. Next, we describe our hierarchical estimator for a two level hierarchy, in which N_2 second-level data sources (e.g., reviews contributed by users) are grouped into N_1 top-level sources (e.g., users).

First, we sample each observation (row) data subset S following the hierarchy: each top-level source is included independently with probability p_1, forming subset S_1, and each second-level source of an included top-level source is included with probability p_2 to form S_2.

Then, we run two estimations: we start by finding top-level proponents only, running our estimator on $N = N_1$ features, featurized with $p = p_1$ (and identical knockoffs), to obtain the set of top-level proponents Prop_1. Then, we find the second-level proponents using a design matrix that includes all top-level source variables, and one variable for each second-level source under a Prop_1 source, featurized as:

$$X_n = \begin{cases} \frac{1}{p_1 p_2} & \text{if } s(n) \in \text{Prop}_1 \cap S_1,\ n \in S_2 \\ -\frac{1}{p_1(1-p_2)} & \text{if } s(n) \in \text{Prop}_1 \cap S_1,\ n \notin S_2 \\ 0 & \text{otherwise } (i.e., s(n) \notin S_1) \end{cases} \tag{2.3}$$

Where $s(n)$ denotes the top-level source that the second-level source n comes from (note that $s(n) \notin S_1 \implies n \notin S_2$). This featurization ensures that $E[X_n | X_{-n}] = E[X_n | p_1, p_2, X_{s(n)}] = 0$, yielding a similar interpretation as Proposition 2.3 for the hierarchical design. Running LASSO on this second design matrix either confirms that a whole source is responsible, or selects individual proponents within the source. Note that both analyses run on the *same set of subset models* $\mathcal{M}_S$: thanks to our hierarchical sampling design, the same offline phase supports all levels of the source hierarchy.

Finally, we adapt the knockoffs in the second-level regression. The dummy features now

name	dataset	model	N	k	N'	k'	poison approach	hierarchy partition
Poison Frogs	CIFAR10 [106]	VGG-11 [108]	4960	10	4960	10	Poison Frogs [155]	example
CIFAR10-50	CIFAR10 [106]	ResNet9 [6]	50000	50	50000	50	trigger [29]	example
CIFAR10-20	CIFAR10 [106]	ResNet9 [6]	49970	20	49970	20	trigger [29]	example
EMNIST	EMNIST [35]	CNN [150]	3578	10	252015	6600	label-flipping	user
ImageNet	ImageNet [154]	ResNet50 [78]	5025	5	~1.2m	100	trigger [29]	URL
NLP	Amazon reviews [142]	RNN [175]	1000	11	~1m	1030	trigger [29]	user

Table 2.1: Datasets, models, and attacks summary: N is the number of sources, N' is the overall size of the training data, k is the number of poisoned sources, and k' is the number of poisoned training examples.

encode tuples (p_1, p_2), and knockoffs are created for second-level sources only, sampled for inclusion with probability p_2, which reflects the conditional distribution of including them in the data subset. We then use Equation 2.3 to compute the feature.

2.5 Evaluation

We evaluate our approach along three main axes. First, in §2.5.1, we use the AME and our estimator to detect poisoned training data points designed to change a model's prediction to an attacker-chosen target label for a given class of inputs. Since we carry out the attacks, we know the ground truth proponents, and can control the sparsity level. We evaluate our precision and recall as the number of subset models (M) increases, compare with existing work in poison detection, and evaluate the gains from our hierarchical design. Second, in §2.5.2, we present a qualitative evaluation of data attribution for non-poisoned predictions and show example data points that have been found to be proponents for various queries. Third, in §2.5.3, we evaluate our AME-based SV estimator and compare it to prior work.

2.5.1 Detecting Poisoned Training Data

We study various models and attacks on image classification and sentiment analysis, summarized in Table 2.1. Fig. 2.3 shows a few concrete attack examples. Appendix A.6.2 provides more details, as well as the hyperparameters used.

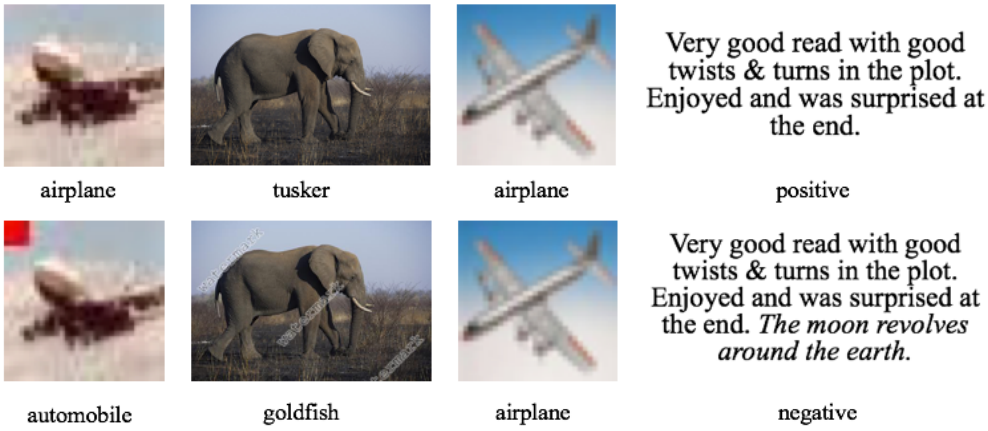


Figure 2.3: Poison data examples, with clean data (top) and poisoned data (bottom) for: the *red-square trigger attack* on *CIFAR10*; the *watermark trigger attack* on *ImageNet*; the *Poison Frogs* attack; and the *NLP trigger*.

	Prec	Rec	c
Poison Frogs	96.1	100	8
CIFAR10-50	96.9	54.4	16
CIFAR10-20	95.3	58.8	8
EMNIST	100	78.9	16

a Training-from-scratch

	Prec	Rec	c
NLP	99.0	97.3	24
CIFAR10-50	97.5	87.1	24
CIFAR10-20	99.0	64.8	48
ImageNet	100	78.0	12

b Warm-starting

Table 2.2: Average precision and recall of LASSO+Knockoffs. The column c denotes the constant in $O(k \log_2 N)$, i.e., we use $M = ck \log_2 N$ subset models.

Precision and recall. Given a query for a mis-classified example at test time, we use our AME estimator to pinpoint those training data points that contribute to the (mis)prediction. A detection is correct if its corresponding training data point has been poisoned. Table 2.2 shows the average precision and recall across multiple queries, for each scenario presented in Table 2.1. To make the number of subset models M comparable across tasks (and because we know k), we report c and use $M = ck \log_2(N)$ subset models. Precision is not counted when nothing is selected to avoid an upward bias. Our largest experiments only run with warm-starting, for computational reasons. Table 2.2 shows that our method (LASSO+Knockoff) achieves very high precision and reasonable recall, and that warm-starting achieves good performance by enabling more utility evaluations.

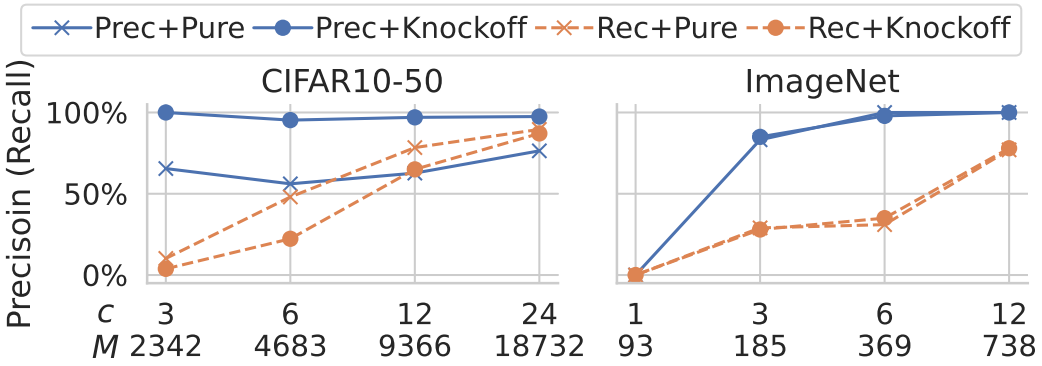


Figure 2.4: Effect of growing c (and corresponding M). Both use warm-staring. Prec+Pure (Rec+Pure) is the precision (recall) for LASSO without knockoffs.

Figure 2.4 shows the precision and recall for two image classification scenarios with and without knockoffs. We see that knockoffs are important for ensuring a consistently high precision (solid blue lines). And the recall (dashed orange lines) grows as the number of subset models grows with c. Appendix A.6 shows the figures for all scenarios (Fig. A.3), as well as other ablation and sensitivity studies for different parts of the methodology and parameters (A.6.3). We also discuss the impact of those choices on running time (A.6.4).

Comparison with prior work. We compare against two recent works: SCAn [163], a poison (outlier) detection technique that requires a set of clean data, and Representer Points [205], a more quantitative approach that measures an influence-like score for training data. We delay the evaluation of other SV algorithms to §2.5.3, as existing methods are not able to run on our large experiments, in which $M < N$. We compare the precision of each method at different recall levels, by varying internal decision thresholds (see Appendix A.6.5 for details on SCAn). Fig. 2.8 summarizes the results and shows that AME performs as well or better than both approaches. AME is particularly efficient when there are very few poisoned training data points, which existing approaches fail to handle (CIFAR10-20). We also see a sharp decrease in precision when recall exceeds a certain level for AME, unlike SCAn. This is because we chose the LASSO regularization parameter as λ_{1se} to favor sparsity in coefficients, in order to minimize false positives.

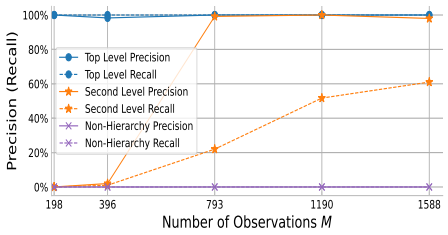


Figure 2.5: Hierarchical design on the NLP dataset, showing the LASSO+Knockoffs precision and recall for top-level (blue), second-level (orange), and non hierarchical (purple).

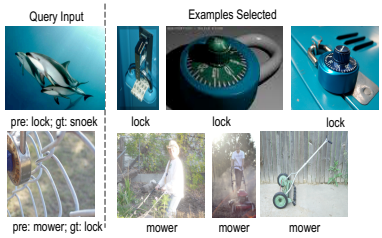


Figure 2.6: Queries for wrong predictions. *Pre* is the model's prediction, *gt* is the ground truth.

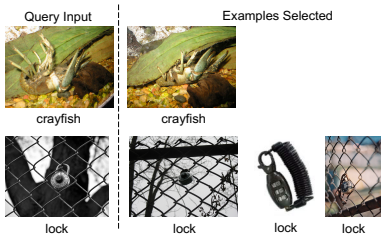


Figure 2.7: Queries for correct predictions.

Fig. A.5 in the Appendix shows the result of another common choice, λ_{min}, with less sparsity. The overall findings are similar, with a more graceful drop in precision-recall curve observed. In our approach, the knockoffs automatically control the false discovery rate to ensure that we remain in the high precision regime.

To show that the AME is able to work at a finer granularity than SCAn, we ran a mixed attack setup on CIFAR10, where we simultaneously use 3 different attacks: each attack has a different trigger and poisons 20 different images. We found that SCAn clusters nearly all poisoned images together (along with many clean images), regardless of the attack used in the query Q; while the AME selects the correct attack for each query, and achieves an average precision (recall) of 96.3% (65.5%), 97.4 (89.5%) and 97.1% (71.3%), respectively for 20 random queries from each attack.

Appendix A.6.5 provides more details and evaluation of SCAn, Representer Points, influence functions, and Shapley values.

Improvements under hierarchical design. We group the 300K users of the NLP dataset into 300 time-based groups or 1K users each. We poison two top-level sources, with 5 and 10 poisons, respectively (details in Appendix A.6.6). Figure 2.5 shows the AME precision and recall in finding the poisons, averaged over 20 different queries (poisoned test points). We find that (a) top-level sources are detected with few observations; and (b) recall for second-level sources

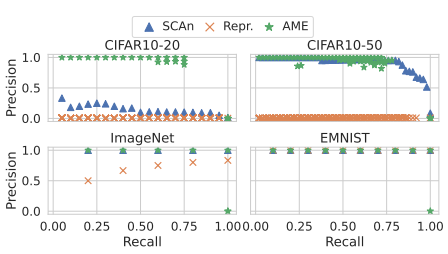


Figure 2.8: Precision v.s. recall curves for comparison with prior poison detection work. "Repr." denotes Representer Points.

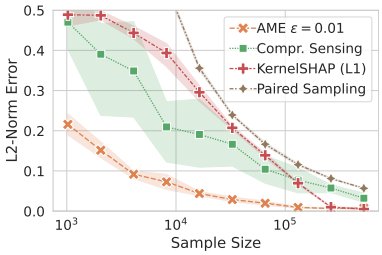


Figure 2.9: Est. error vs sample size on synthetic dataset. The shaded area shows 95% confidence intervals.

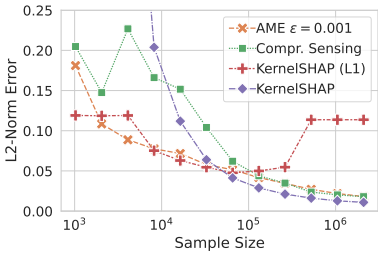


Figure 2.10: Est. error vs sample size on Poisoned MNIST dataset.

degrades gracefully as the number of observations (submodels) decreases. The hierarchical split is also efficient: it achieves ~100% precision and 60% recall with 1.5K observations, before our non-hierarchical method detects anything.

2.5.2 Data Attribution for Non-poisoned Predictions

We next measure what training data led to a specific prediction in the absence of poisoned data. Figs. 2.6 and 2.7 show examples from a subset of ImageNet, for correct and incorrect predictions, respectively. Qualitatively, explanation images share similar visual characteristics. Quantitatively, Figure A.10 in Appendix A.6.7 shows that removing the detected inputs significantly reduces the target prediction's score (compared to a random removal baseline), showing that we detect inputs with significant impact. Additional results, and a comparison to a random baseline, can be found at `https://enola2022.github.io/`. Appendix A.6.7 details the setting, and shows results for CIFAR10.

2.5.3 Shapley Value Estimation from AME

Finally, we showcase our SV estimator from AME using simulated data and a subset of MNIST with poisoning, which are small enough to study the $M > N$ case where known estimators are applicable. In both setups, we know the ground truth or can approximate it closely enough, re-

spectively (details in Appendix A.6.8). We compare the AME to KernelSHAP [126], Paired Sampling [37], and two sparsity-aware methods: "KernelSHAP (L1)" [126] that uses LASSO heuristically to filter out variables before fitting a linear regression, and Compressive Sensing [92]. We use p-featurization (§A.3.2) without knockoffs, and $p \sim Uni(\varepsilon, 1-\varepsilon)$.

The results in Figure 2.9 (simulated data – KernelSHAP without L_1 regularization mostly overlaps with Paired Sampling and thus is not shown) and Figure 2.10 (MNIST data – Paired Sampling omitted due to prohibitive memory and computation costs) show that AME delivers the fastest rate among these baselines on small sample sizes, and remains competitive when sample sizes become larger, though with a slightly larger final error than KernelSHAP on MNIST (likely due to approximate sparsity). This larger error, however, is for a large sample size ($M > 50N$) a regime unlikely to be practical for SV given the cost of utility evaluations (model training). Notably, on MNIST, "KernelSHAP (L1)" error is as low as the AME when the sample size is small, while it diverges with more samples. This seems to be due to incorrect filtering in the heuristic LASSO step, which misses one of the k poisoned datapoints (variables) on large sample sizes. AME does not have this instability as LASSO is the final estimate of the SV. Appendix A.6.8 shows comparisons to additional baselines, as well as ablation studies.

2.6 Related Work

We focus on the closest related work, and refer the reader to Appendix A.7 for a broader discussion. Efficient Shapley value (SV) [157] estimation is an active area, and is closest to our work. Recent proposals also reduce SV estimation to regression, although differently than we do [37, 90, 126, 194]. Beta-Shapley [110] proposes a generalization of SV that coincides with the AME when $p \sim$ Beta (we found the truncated uniform to be better in practice, but provide error bounds for both). None of these works study the sparse setting, or provide efficient L_2 bounds in this setting. This may stem from their focus on SV for *features*, in smaller settings than we consider for train-

ing data, and in which sparsity may be less natural. The most comparable work to ours is that of Jia et.al [92], which provides multiple estimators, including under sparse, monotonic utility assumptions. Their approach uses compressive sensing (closely related to LASSO) and yields an $O(N \log \log N)$ rate. We significantly improve on this rate with an $O(k \log N)$ estimator, which is much more efficient in the sparse ($k \ll N$) regime.

Other principled model explanation approaches exist, based on influence functions [54, 102, 104], Representer Points [205], or loss tracking [73, 149]. They either focus on marginal influence on the whole dataset [102, 104]; make strong assumptions (e.g., convexity) that disallow their use with DNNs [11, 102, 104]; cannot reason about data sources or sets of training samples [23, 73, 149]; or subsample training data but focus on a single inclusion probability, and thus cannot explain results in all scenarios [54].

3 Revealing and Diagnosing GNN Vulnerabilities: through Stealthy Attacks

This chapter is about revealing and diagnosing GNN Vulnerabilities in node classification tasks. GNNs are vulnerable to targeted poisoning, in which an attacker manipulates the graph to cause a target node to be mis-classified to a label of the attacker's choosing. For a k-layer GNN model, existing attacks modify the target's k-hop neighborhood to achieve maximal effectiveness. However, from a practical standpoint, it is much more convenient to launch a *long distance* attack that can manipulate graph locations outside the target's k-hop neighborhood which contains many more nodes to choose from. Furthermore, long distance attacks are not susceptible to be diagnosed by existing postmortem GNN explanability tools which only investigate the target's k-hop neighborhood. Although some existing attacks (such as meta-attack) can be modified to be long distance, they are too computationally expensive to work on a large scale graphs. In this paper, we propose a practical long distance poisoning attack called MimicLDT that can scale to large graphs while achieving competitive attack success rates compared to a modified meta-attack on small graphs. To the best of our knowledge, MimicLDT is the first long distance GNN targeted poisoning that can successfully attack large graphs, motivating the need to find scalable GNN poisoning diagnosis and detection tools.

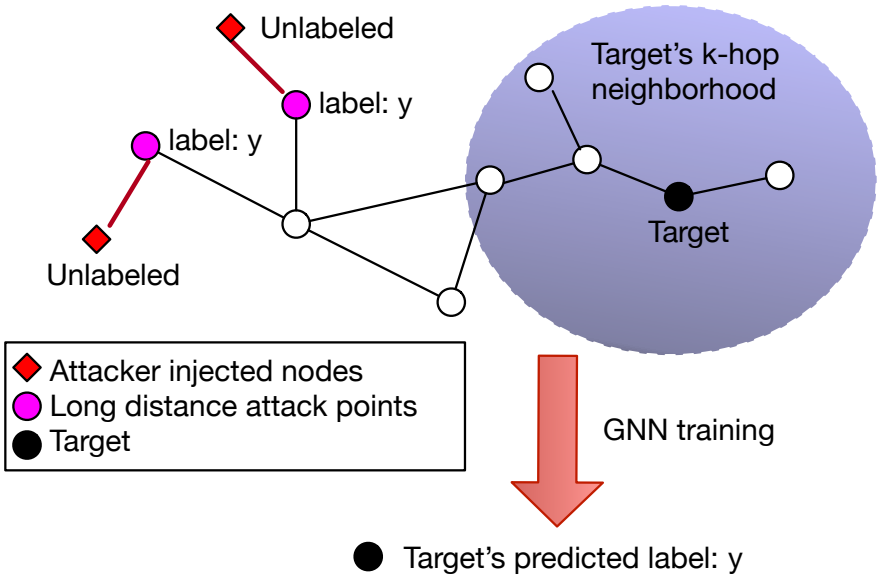


Figure 3.1: Targeted poisoning attack via long distance node injection.

3.1 Overview

Many recent papers have proposed attacks on GNNs that lead to mis-predictions. To use these attacks, an attacker must either modify the structure of the graph [15, 24, 26, 27, 40, 62, 137, 180, 181, 200, 210, 227], modify features of graph nodes [125, 128, 195, 226], or inject new nodes with carefully crafted features [30, 93, 96, 141, 162, 168, 185, 188, 225]. Depending on when the adversarial perturbation occurs, these attacks can be classified as poisoning (training time) attacks or evasion (test time) attacks. Furthermore, depending on the attacker's aim, these attacks can also be either targeted attacks (where that goal is to change the prediction of one or a few nodes) or untargeted (where the goal is to reduce overall prediction accuracy). In this paper, we seek to generalize targeted poisoning attacks on GNNs used for node classification, i.e., we seek to generalize training time attacks that allow the attacker to change a selected target node's label to an attacker's chosen label.

Our goal is to generalize targeted poisoning attacks to remove restrictions on how attackers

must change the graph, in particular allowing changes outside of the target's immediate neighborhood. Existing attacks require changes within the target's k-hop neighborhood (where k is the number of layers in the GNN), requiring attackers to add edges or change node features within this neighborhood [15, 24, 26, 40, 62, 180, 200, 210, 226], or create a new fake node that is connected directly to the target node [30, 39, 185, 197]. In this paper we ask if there exists a more general *efficient long-distance targeted poisoning attack*, i.e. an **efficient targeted-poisoning attack that only requires changes outside the target's k-hop neighborhood**. We seek an efficient attack to ensure that it can be used with the large graphs that today's GNNs are commonly trained on. Making the attack long-distance can provide attackers with greater flexibility in changing the graph. Increased efficiency and long-distance attacks can also limit the ability of GNN diagnosis tools to identify the attackers changes: GNN diagnosis tools built using state of the art GNN explanation algorithms [49, 84, 127, 206, 208] that measure influence by only looking at nodes within a target's k-hop neighborhood, while specialized GNN poisoning detection and avoidance tools such as FocusedCleaner [224] cannot easily scale to large graphs.

We show that efficient long-distance poisoning exists by proposing MimicLDT, a heuristic based attack that inserts (injects) new nodes and edges outside the target node's k-hop neighborhood. We developed the MimicLDT algorithm by observing the behavior of a meta-learning [14, 57] based targeted long-distance node-injection attack, MetaLDT, that we developed and that could only be applied to small graphs (e.g., Cora). We observed that MetaLDT injects nodes that make the target node's embedding (i.e., the output of the last GNN layer before the softmax classifier) the same as the embedding of nodes with the target label (we refer to these nodes as attack points). MetaLDT does so by injecting nodes in the neighborhood of existing nodes with the target label, and adjusting injected node features. MimicLDT uses the same process, but adopts two approaches to reduce time: first, rather than trying to search for an optimal set of attack points, MimicLDT chooses a random set of attack points; and second, MimicLDT uses a cheaper surrogate loss function when optimizing the injected node's features. In combination these tech-

niques allow MimicLDT to be used with graphs that had hundreds-of-thousands of nodes (e.g., the ArXiv graph [83]) and achieve reasonable poison success rate (> 55%) without changing the target node's k-hop neighborhood.

In summary, we make the following contributions:

- We study whether *long distance attacks* that modify nodes and edges outside of the target's k-hop neighborhood can poison a GNN, and show that they can.
- We propose a heuristic optimization approach, MimicLDT, that can perform targeted poisoning by injecting fake nodes that lie beyond the target node's k-hop neighborhood. To the best of our knowledge, MimicLDT is the only long distance targeted poisoning attack that scales to large scale graphs like ArXiv.
- We evaluate our attack on different graphs and defenses. To the best of our knowledge, we are the first to show the existence and effectiveness of long distance targeted poisoning attacks.

3.2 Background

We discuss related work that attack GNN-based node classification, with a focus on targeted attacks. For more comprehensive surveys, we refer readers to [94, 221].

GNN attacks. Targeted attacks can occur during training time (poisoning attacks) or test time (evasion attacks). For any given k-layer GNN model architecture, the target node's label prediction is a function of (1) model weights, (2) the input features of the target node itself, and (3) the input features of the target's k-hop neighbors. Thus, in order to manipulate the target's label prediction, the attacker can try to corrupt any of these three factors. Attacks of type (2), aka corrupting the target's input features, is a well-studied problem in non-graph domains [67, 130, 156] such as images, text, and time series, and can be straightforwardly extended to the

graph setting. Thus, existing GNN attacks, including both poisoning and evasion attacks, focus on the adversarial manipulation of (3), aka the target's k-hop neighborhood, achieved through adding/removing edges (referred to as structure perturbation attacks), or adding fake nodes (referred to as injection attacks). The manipulation of (3) can be further categorized into direct vs. indirect attacks depending on whether the target's direct or k-hop neighborhood is modified. There is a vast collection of attacks of type (3); most perturb graph structure, e.g. NetAttack [226], FGA [26], MGA [27], PGD [200], DICE [190], GUA [210], RL-S2V [40], Bojchevski et al. [15], GF-Attack [24], GAFNC [93], IG-JSMA [195], Wang et al. [181] and PR-BCD/GR-BCD [62]. Some also modify existing nodes' labels or features, e.g. [125, 195, 226]. Others inject fake nodes, e.g., Wang et al. [188], TDGIA [225], AFGSM [185], G^{2}A2C [96] and G-NIA [168].

Our work differs from the above attacks in two aspects. One, we aim for the attack for work at *long distance*, by avoiding modification to the target's k-hop neighborhood. Two, we want the attack to be able to scale to large graphs, at the cost of reduced attack success rate compared to existing work. A recent work, G-FairAttack [212], also aims to accelerate the computation needed for attacks, but it is designed to attack the fairness of fairness-aware GNNs while preserving prediction accuracy while our work attacks a targeted node's label prediction through long-distance poisoning.

Making attacks stealthy. Recent work such as HAO [30] and ADIMA [167] aim to make attacks hard to detect; the former adds a homophily constraint and the latter uses a GAN-style framework to train a discriminator to distinguish subgraphs that include fake nodes from those that don't. Like HAO, our attack also tries to preserve homophily. Existing attacks all assume that the attacker can add/remove edges to *any* existing node. A recent proposal tries to make attacks more realistic by assuming that the attacker can only use a subset of nodes as attack points [128]. Our work can also be extended to this setting by restricting the set of attack points.

GNN Defenses. Since poisoning attacks manipulate the training graph, defenses aim to avoid learning from perturbed graph structure. Two major approaches exist: graph cleaning as part of

pre-processing (e.g., Jaccard GCN [195], SVD GCN [51], GNNGuard [216], and FocusedCleaner [223]), which finds and eliminates adversarial perturbations before training; graph learning (e.g. ProGNN [95], Soft-Median-GDC [61]), which leverages the characteristics of adversarial attacks to guide the graph learning process to be more robust. Besides empirical defenses through more robust training, there are also *certified defenses* aiming to provide formal robustness guarantees. Some work [182] provides certified robustness to graph evasion attacks by utilizing randomized smoothing [36, 116]. Some work[165] selects a fraction of existing node pairs in advance for immunization with the assumption that immunized links cannot be modified by attacks, so that the graph can be made certifiably robust. More recently, GNNCert [198] utilizes a hash function to divide the graph into multiple sub-graphs and uses majority voting for each sub-graph as the final prediction to provide a deterministic robustness guarantee for graph classification. None of above mentioned certified defenses handle node injection attacks. A more recent follow-on work [166] proposes node-level immunization to defend against node injection. However, immunizing the target node cannot defend against long distance attack.

Explainability tools, such as GNNExplainer [206] and others [49, 53, 127, 207, 208, 214], cannot be directly used to detect attacks. However, they can be used to provide post-mortem analysis of a misprediction due to an attack. Recent work [52] has used GNNExplainer to diagnose adversarial perturbations that lead to given bad predictions. Long-distance attacks like ours have the advantage of being stealthy against existing GNN explainability tools.

3.3 Problem Definition

In this section, we define the terminology we use, formalize our setting, and state our assumptions about the attacker's capabilities. We use the standard terminology and notation for graphs and GNNs:

Graphs. We use $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to denote a graph with nodes $\mathcal{V} = \{v_1, v_2, .., v_n\}$ and edges

$\mathcal{E} = \{e_1, e_2, ..e_m\}$, $\mathbf{A} \in \{0,1\}^{n \times n}$ to denote $\mathcal{G}$'s adjacency matrix, and $\mathbf{X} \in \mathbb{R}^{n \times d}$ to denote the d-dimensional feature matrix for each node $v \in \mathcal{V}$.

GNN based node classification. We target the setting where a GNN uses *transductive learning* to classify graph nodes. In this setting, the GNN, f_θ's training data consists of a graph $\mathcal{G}$, a subset of whose nodes are labeled, denoted as $\mathcal{V}_L \subset \mathcal{V}$. The training process aims to learn model weights θ so that the mode f_θ can can predict labels for unlabeled nodes $\mathcal{V}_U := \mathcal{V} \setminus \mathcal{V}_L$.

Node classification using a GNN f_θ can be viewed as requiring two computational steps: in the first, the GNN is used to compute an embedding for the node; and in the second step this embedding is used as input to a classification layer which outputs a set of *logits* for label prediction. The embedding computed in the first step represents the node's features and its neighborhoods, and in what follows we use the term *node embedding* to refer to the first step's output.

3.3.1 Attack Model

Attacker's goal. We consider *targeted* label-flipping attacks, where the attacker selects a target node v_t and target label y_t, and the attack aims to alter the graph used to train the GNN f_θ so that it predicts label y_t for node v_t.

Attacker's knowledge. We assume the attacker has access to the training data, including the original graph G, node features, and labels, and also knows the training procedure (including any changes made to the training to improve model robustness). Note that we do not assume knowledge of model weights, but if available this can be used to further reduce the cost of MimicLDT.

Attacker's capability. We focus on long-distance node injection attacks. Consequently, we constrain attackers so they cannot modify or remove existing nodes, cannot remove edges, and cannot add any edge that connects two nodes that are already in the graph. The attacker can add (inject) one or more new nodes, and add edges connecting injected nodes to each other, or connecting existing nodes to injected nodes. Finally, in order to allow us to evaluate attack efficiency, we limit the number of nodes and edges the attacker can add.

Node injection attacks. Formally, a **node injection attack** takes as input a target node v_t and graph G (with adjacency matrix A), and generates a poisoned graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ by injecting a set of malicious nodes $\mathcal{V}_{inj}$. Graph $\mathcal{G}'$ has the adjacency matrix $\mathbf{A}' = \begin{bmatrix} \mathbf{A} & \mathbf{B}_{inj} \\ \mathbf{B}_{inj}^T & \mathbf{O}_{inj} \end{bmatrix}$, and feature matrix $\mathbf{X}' = \begin{bmatrix} \mathbf{X} \\ \mathbf{X}_{inj} \end{bmatrix}$, where $\mathbf{X}_{inj}$ are the injected nodes features, $\mathbf{B}_{inj}$ is the adjacency matrix between injected and existing nodes in $\mathcal{G}$. We refer to an existing node that connects to any injected node as an *attack point.* In our evaluation, we constrain the number of injected nodes and the degree of each injected nodes, that is, we require that $|\mathcal{V}_{inj}| \leq \triangle \in \mathbb{Z}$ and $1 \leq deg(i) \leq b \in \mathbb{Z}, \forall i \in \mathcal{V}_{inj}$ for some threshold $\triangle$ and b.

We focus on **long-distance node injection** attacks, where injected nodes can lie outside the k-hop neighborhood of the target node v_t, where k is a user supplied threshold, usually equal to the number of layers in the GNN. This in contrast to existing node injection attacks that require nodes to be placed within the target node's k-hop neighborhood.

This paper aims to demonstrate that there exist efficient, long-distance node injection attacks. To do so, we develop efficient attacks that enforce a stronger condition: they make no changes to the target's k-hop neighborhood. We refer to attacks that meet this stronger condition as **strong long-distance node injection attacks**, and formally define them as follows:

Definition 3.1.1 (Strong Long-distance Node Injection). A node injection attack where no attack point is within the target node v_t's k-hop neighborhood where k equals to the number of GNN model layers. More formally, in a strong long distance attack on a k-layer GNN, $\forall v_a \in \mathcal{V}_a,\ d(v_a, v_t) > k$ where $\mathcal{V}_a$ is the set of existing-graph nodes connected to injected nodes (aka attack points), and $d(v_a, v_t)$ is the path length from v_a to v_t.

3.3.2 Problem Formulation

Our approach expresses the attack as an optimization problem. GNN attacks that modify graph structure can be formalized as the following optimization problem:

$$\min_{\mathcal{G}'} \quad \mathcal{L}_{atk}(f_{\theta^*}(\mathcal{G}')) \quad s.t. \quad \theta^* = \arg\min_{\theta} \quad \mathcal{L}_{\text{train}}(f_\theta(\mathcal{G}')). \tag{3.1}$$

where $\mathcal{L}_{train}$ is the general loss function used when training model f_θ, which we assume the attacker knows. Therefore, our goal is to find a graph, $\mathcal{G}'$, that minimizes the attacker's loss $\mathcal{L}_{atk}$.

A targeted label-flipping attack requires incorporating the target node and desired label into the loss-function. More precisely, we need a loss function that maximize the target node's *logit* (i.e., the model's confidence score for a label) for the attacker-chosen label. Therefore, we use $\mathcal{L}_{atk} = -\mathcal{M}_{\mathcal{G}'}(v_t)[y_t]$, where $\mathcal{M}_{\mathcal{G}'}(v_t) = f_{\theta^*}(v_t; \mathcal{G}')$, which maximizes the probability that target node v_t has label y_t. Beyond this, and similar to prior work [30], we want to ensure that the attack is stealthy and injected nodes do not differ significantly from existing nodes. To do so, we incorporate a homophily term in $\mathcal{L}_{atk}$ that minimizes feature differences between an injected node and its neighbors. Our final attacker loss function, $\mathcal{L}_{atk}$, is thus:

$$\mathcal{L}_{atk} = -\mathcal{M}_{\mathcal{G}'}(v_t)[y_t] - \beta C(\mathcal{G}') \tag{3.2}$$

$$C(\mathcal{G}') = \frac{1}{|\mathcal{V}_{inj}|} \sum_{u \in \mathcal{V}_{inj}} sim(r_u, X_u), \text{ where } r_u = \sum_{j \in \mathcal{N}(u)} \frac{1}{\sqrt{d_j}\sqrt{d_u}} X_j \tag{3.3}$$

where β is a hyperparameter that controls how important homophily is, $sim(\cdot)$ measures *cosine similarity*, $\mathcal{N}(u)$ is the set of nodes neighboring node u, and d_u is the node degree. The homophily formulation above is based on [30].

3.4 Attack Design: MimicLDT

3.4.1 MimicLDT Overview: via Embedding Collision

A natural approach to demonstrating the feasibility of strong long-distance attacks would be to extend meta-attack [227] and incorporate the distance constraints. We tried this approach (which we call MetaLDT), and found that it does not scale to most graph datasets (including common datasets, e.g., ArXiv). The lack of scalability made it hard for use to analyze the generality of long-distance attacks and prompted us to design a faster and more scalable heuristic attack, MimicLDT.

We model any attack framework as a procedure that takes as input a graph $\mathcal{G}$ and produces an attack graph $\mathcal{G}'$. The MimicLDT attack framework's heuristics are derived from the following observations about the meta-attack framework: (a) We found that each meta-attack iteration reduces the embedding space distance (as determined by the surrogate GNN) between v_t and existing nodes with the attacker's chosen target label y_t. We empirically demonstrate this phenomenon in Figure 3.2 using MetaLDT to attack a GCN that uses the Cora dataset. We give more details about this setting in §3.5. The graph shows how the average L2-distance, in each iteration's surrogate model's embedding space, between v_t and nodes whose ground truth label is y_t varies across iterations, and we observe that the optimization minimizes this distance.
(b) The attack graph $\mathcal{G}'$ produce by meta-attack tends to add edges between injected nodes and nodes with the target label y_t. We hypothesize that this is in support of the previous observation: an injected node, that connects to a node v labeled y_t, can reduce the embedding distance between v_t (the target) and v.
(c) The edges connecting pairs of injected nodes in $\mathcal{G}'$ do not appear to have any noticeable patterns. This leads us to hypothesize that it is sufficient to randomly connect injected nodes with each other.

Similar to meta-attack, MimicLDT assumes knowledge of how the target model is trained, and takes as input the original graph $\mathcal{G}$, a target node v_t, and a target label y_t. MimicLDT is an optimization based approach, and evaluates the quality of each intermediate graph using a model trained using the same procedure (i.e., algorithm) as the target model. In what follows, we refer to the model used by MimicLDT as a *surrogate model*. Given these inputs, MimicLDT produces the attack graph $\mathcal{G}'$ as follows: (a) First, it trains a surrogate model f_θ using $\mathcal{G}$, which is used for the entire optimization. Much of MimicLDT's performance improvement is because it only needs to train one surrogate model. (b) Next, MimicLDT generates the structure of the attack graph $\mathcal{G}'$ based on heuristics. (c) Finally, it optimizes injected node features in $\mathcal{G}'$ to produce the final attack graph.

3.4.2 Determining Graph Structure

MimicLDT starts by generating an initial attack graph $\mathcal{G}'$ from $\mathcal{G}$. To construct $\mathcal{G}'$, MimicLDT first randomly selects a set of nodes, $\mathcal{V}_a$, in $\mathcal{G}$ whose label is y_t. We refer to the nodes $\mathcal{V}_a$ as attack points, and a hyperparameter r determines $|\mathcal{V}_a|$, the number of attack points chosen.[1]

Next, for each attack point, $v_a \in \mathcal{V}_a$, MimicLDT injects Φ nodes $\mathcal{V}_{g_v}$ and connects them (directly or indirectly) to v_a. To do so, MimicLDT iterates over possible edges connecting nodes in the set $\mathcal{V}_{g_v} \cup \{v_a\}$ (i.e., edges that either connect injected nodes to each other or to the attack point), and add each edge to the graph with probability $p = 0.5$. Finally, it prunes any nodes in $\mathcal{V}_{g_v}$ not reachable to any v_a. Therefore, the final set of injected nodes may have fewer than Φ nodes.

The final graph structured produced by MimicLDT thus consists of all nodes and edges in $\mathcal{G}$, and a set of injected nodes that are connected to each other and attack points. In the rest of this section, we use $\mathcal{V}_i$ to refer to the set of injected nodes in $\mathcal{G}'$, and $B(v_i)$ to refer to the attack point whose k-hop neighborhood contains the injected node $v_i \in \mathcal{V}_i$.

[1] In the evaluation we use $0 < r < 1$, and select $r|\mathcal{V}_L|$ points ($\mathcal{V}_L$ is the set of labeled training nodes in $\mathcal{G}$).

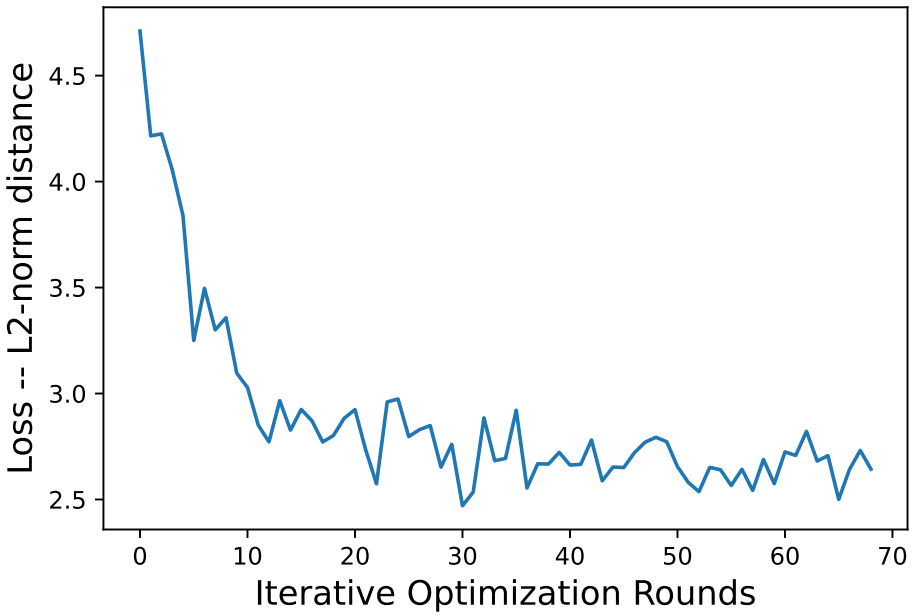


Figure 3.2: The embedding space distance between the target node and existing nodes whose ground-truth label is the same as the attack's target label, as MetaLDT's optimization progresses.

3.4.3 Determining Injected Node Features

The algorithm above generates a graph $\mathcal{G}'$ that contains all nodes in $\mathcal{G}$ and the set $\mathcal{V}_i$ of injected nodes. We do not assign labels to any of the injected nodes, and formulate the problem of assigning feature vectors to these injected nodes as an optimization problem, which we describe below.

Optimization formulation. Our feature optimization problem aims to meet two goals. First, based on our observations from meta-attack, we try to ensure that the final node embedding for any selected attack point $v_a \in \mathcal{V}_a$, $\mathbf{h}_{v_a}^{(L)}$, is close to the target's final node embedding $\mathbf{h}_{v_t}^{(L)}$. Second, same with extending meta-attack, we try to ensure that an injected node v_i's feature vector $\mathbf{X}_{v_i}$ is similar to that of its attack point $B(v_i)$.

Taken both optimization goals into account, our final optimization formulation is:

$$\mathbf{X}^*_{\mathcal{V}_i} = \arg\min_{\mathbf{X}_{\mathcal{V}_i}} \mathcal{L}_{atk}, \tag{3.4}$$

$$\mathcal{L}_{atk} = -\left(\frac{1}{|\mathcal{V}_a|} \sum_{v_a} Sim_f(\mathbf{h}^{(L)}_{v_a}, \mathbf{h}^{(L)}_{v_t}) + \beta * \frac{1}{|\mathcal{V}_i|} \sum_{v_i} Sim_{in}(\mathbf{X}_{v_i}, \mathbf{X}_{B(v_i)}) \right) \tag{3.5}$$

In this formulation, $\mathbf{X}_{\mathcal{V}_i}$ represents a $|\mathcal{V}_i| \times d$ dimensional matrix whose columns are features of nodes in $\mathcal{V}_i$; Sim_f is a metric function (specific to the GNN used) that measures similarity between a pair of final node embeddings; and Sim_{in} is a metric function (specific to the input graph $\mathcal{G}$) that measures similarity between a pair of nodes' feature vectors.

The second term of the optimization in Eq 3.4 aims to preserve homophilly, but differs from the formulation used in prior work [30]: rather than maximizing the node-centric homophily score which aggregating with neighborhood for all injected nodes, this formulation maximizes similarity between the attack points and injected nodes. We found that this change in formulation improved our performance, and our empirical results (in Appendix B.1.3) show that it does not noticeably impact the homophily scores for injected nodes. The β hyperparameter allows attackers to decide how much the generated attack prioritizes homophily.

Computing feature vectors. We use a stochastic gradient descent based optimizer to compute feature vectors. Our evaluation uses GNNs to classify nodes in citation graphs, and consequently we use cosine similarity to measure similarity between node features (i.e., Sim_{in} is cosine similarity), and we use $L2$-norm to measure similarity between final node embeddings (Sim_f).

We considered various initial values for $\mathbf{X}_{\mathcal{V}_i}$, including using the input features of the neighboring attack point ($X_{v_i} = X_{B(v_i)}$), input features of a random neighbor, and the target's input features. Empirically, we found no noticeable difference between these options, and found that the embedding space distance and feature space distance convereged to similar values, regardless of how they were initialized. During the optimization, we use the surrogate model f_θ to compute

$\mathbf{h}_{v_a}^{(L)}$ and $\mathbf{h}_{v_t}^{(L)}$.

3.5 Evaluation on MimicLDT

We run experiments on NVIDIA V100 GPUs, with 32GB memory limitation. Our evaluation aims to answer the following questions:

- Can our attacks poison existing GNN models and their fortified versions?
- How effective is MimicLDT compared to a meta-learning based attack? For this evaluation we depend on MetaLDT, a meta-learning based long distance node-injection that we developed for this paper.
- How does MimicLDT's scalability compare to MetaLDT when targeting large graphs?
- How does MimicLDT compare to existing short-distance ones?
- How does MimicLDT impact graph homophily and degree distribution, i.e., is MimicLDT stealthy?
- Can we launch effective end-to-end attacks?

3.5.1 Setup

We start by describing the datasets, models and baselines used in our experiments.

Datasets. We use three datasets of varying sizes: Cora [204] (2708 nodes, 5429 edges), PubMed [204] (19717 nodes, 44338 edges) and Ogbn-arXiv [82] (169343 nodes, 1157799 edges). The largest graph, ArXiv, is almost two orders of magnitude larger than the smallest, Cora. Appendix§B.3 provides details.

GNN models. We use three popular GNN models: **GCN** [101], **GraphSAGE** [72], **GAT** [179]. We use 3-layer models for most datasets, the exception is Cora, which is a small graph and for which

we use a 2-layer model. We provide detailed model settings, the training process, and hyper-parameters in Appendix§**??**. In addition to vanilla models, we also evaluate our attacks against models that use the following 5 GNN defense mechanisms: **ProGNN** [95], **GNNGuard** [216], **Soft-Median-GDC** [61], **Jaccard GCN** [195] and **SVD GCN** [51].

Baselines: We compared MimicLDT against two baselines:

- MetaLDT, an extension to meta-attack [85], that incorporates the constraints used by MimicLDT, and thus performs long-distance node-injection attacks; and
- several previously published short-distance attacks.

We compare against both node-injection and modification short-distance attacks. We provide details for these baselines below.

MetaLDT. MetaLDT is a meta-learning based long-distance node injection attack that we developed by extending meta-attack [85]. We provide a brief description here, a more complete description can be found in Appendix B.2.

MetaLDT takes the same inputs as MimicLDT, i.e., the graph $\mathcal{G}$, target node v_t, target label y_t, and information about how the GNN is trained. Given this input, it first generates an initial attack graph $\mathcal{G}'_0$ by injecting Δ new nodes into the input graph $\mathcal{G}$. The injected nodes initially have a feature vector of 0 and are not connected to existing nodes in $\mathcal{G}$.

Starting from the initial attack graph $\mathcal{G}'_0$, MetaLDT uses an iterative process to produce the final attack graph. Each iteration i ($i \geq 1$) produces an output graph $\mathcal{G}'_i$, and alternates [86] between updating injected node features or updating the graph by adding or removing edges (and thus altering the adjacency matrix). When updating the adjacency matrix, MetaLDT uses the standard meta-learning approach to minimize the attacker's loss function $\mathcal{L}_{atk}$. For efficiency, MetaLDT uses an alternate gradient descent based approach when updating injected node features, since this allows us to change the features of all nodes in a single optimizations step, rather

than requiring that we change a single node and feature dimension in each optimization. The gradient based approach prevents changes to existing nodes in their input graph G by zeroing-out gradients corresponding to them.

Finally, to ensure that the same constraints apply to both MimicLDT and MetaLDT, we restrict MetaLDT from adding edges connecting two nodes in the input graph G, limit the number of edges and nodes that can be injected, and impose a constraint on the distance from the target node to an injected node. More details can be found in Appendix B.2. We also report on how hyperparameters affect MetaLDT's behavior in Appendix B.4.4 and evaluate its design in the appendix.

Short-distance attacks. Existing attacks perturb the target's k-hop neighborhood and are thus *short-distance* attacks. We compare against three short-distance modification attacks: **Nettack** [226], **FGA** [26] and **IG-FGSM** [195] as well as the node-injection poisoning attack: **AFGSM** [186]. In all cases, we use a loss function designed for our goal of changing a target node's label to a specified one (details in Appendix. §B.6).

3.5.2 Evaluation on MimicLDT Attack

Table. 3.1 shows MimicLDT's poison success rate over Cora, PubMed and ArXiv datasets. The attacks are evaluated against vanilla models as well as robust models that use different defense mechanisms. The poison success rate is calculated over 200 experiments, each with a randomly chosen target node and target poison label. Our evaluation focuses on the strong long-distance scenario, and all attack points are outside the target's k-hop neighborhood.

In our evaluation, we use hyperparameter r to control the number of attack points: in each experiment we used $r \times |\mathcal{V}_L|$ attack points, where $\mathcal{V}_L$ is the set of labeled training nodes. For ArXiv and PubMed we set $r = 0.005$, while for Cora (which is smaller, and thus has fewer labeled nodes) we set $r = 0.01$. We limit MimicLDT to inject a maximum of $\Delta = \Phi * r|\mathcal{V}_L|$ ($\Phi = 4$) nodes. Due to time constraint, instead of training a surrogate model, our experiments directly use the

Table 3.1: Success rate of MimicLDT, MetaLDT, short-distance attacks (including modification attacks: Nettack, FGA and IG-FGSM; and the node-injection poisoning attack: AFGSM) over Cora, PubMed and ArXiv datasets. Numbers in parentheses indicate cases where MetaLDT could not complete, and we instead ran a variant where inner-training runs for 50 epochs. Numbers with a *star* sign indicate some attacks are OOM and we only show the range of attacks that are not OOM. More detailed numbers are in Appendix§B.7.

		Cora (2708 nodes)			PubMed (19717 nodes)			ArXiv (169343 nodes)		
		MimicLDT	MetaLDT	Short-Dis.	MimicLDT	MetaLDT	Short-Dis.	MimicLDT	MetaLDT	Short-Dis.
Vanilla	GCN	0.67	0.96	0.64—1.00	0.71	OOM	0.28—1.00	0.74	OOM	OOM
	GraphSAGE	0.63	0.87	0.42—0.96	0.69	OOM	0.45—1.00*	0.73	OOM	OOM
	GAT	0.60	0.84	0.53—0.97	0.69	OOM	0.42—1.00*	0.70	OOM	OOM
Robust	GNNGuard	0.70	(0.53)	0.61—1.00	0.70	OOM	0.45—1.00*	0.64	OOM	OOM
	SoftMedianGDC	0.55	(0.58)	0.46—1.00	0.56	OOM	0.38—1.00*	0.59	OOM	OOM
	JaccardGCN	0.66	0.91	0.33—1.00	0.67	OOM	0.10—1.00	0.63	OOM	OOM
	SVDGCN	0.74	0.83	0.03—1.00	0.60	OOM	0.09—1.00	0.62	OOM	OOM
	ProGNN	0.59	(0.55)	0.57—1.00	0.57	OOM	0.44—1.00*	0.58	OOM	OOM

weights of models under attack. We have evaluated both ways over Cora and found they result in similar success rates. As shown in Table. 3.1, MimicLDT can achieve reasonable poison success rate, for vanilla models (60%∼74%) as well as robust models (55%∼70%).

Comparing with the extended Meta-Attack long-distance attacks. We compare MimicLDT to MetaLDT. While we attempted to use MetaLDT to attack all of the graphs we evaluate, we found it impractical to run on large datasets, and could only attack the Cora dataset. Table. 3.1 reports MetaLDT's poison success rate for different GNN models, both vanilla ones as well as the their fortified versions. The poison success rate is calculated over 200 experiments, each with a randomly chosen target node and target poison label. We used $\Delta = 68$ as a budget, i.e., MetaLDT could perform up to 68 changes to the adjacency matrix. For each step of adjacency matrix optimization, MetaLDT performs $q = 1000$ optimization steps on injected nodes' feature.

Table. 3.1 shows that MetaLDT can achieve high attack success rate (84%∼96%) over vanilla GNN models. When evaluating MetaLDT against robust models, we assume the attacker is aware of the defensive mechanism used and adapt MetaLDT accordingly [137]. However, doing so comes at the cost of increased memory consumption and computational overhead. Hence, for some robust models (GNNGuard, SoftMedianGDC, ProGNN), we stop MetaLDT's inner-training

loop early before its convergence after 50 instead of the regular 200 epochs, in order to avoid OOM. From Table. 3.1, we can see that when MetaLDT's inner training loops is allowed converge (JaccardGCN, SVDGCN), its success rate remains high. However, stopping the inner training loop early comes at a significant cost of poison success rate. It is crucial that MetaLDT adapts to the underlying GNN defense. We report the results of nonadaptive MetaLDT in §B.4.5.

MetaLDT is more expensive than MimicLDT. MetaLDT requires a lot of compute and memory resources due to its extensive unrolling process. Table. 3.2 compares the running time (on a V100) and the memory cost of MetaLDT and MimicLDT for the GCN model over various datasets. We found that MetaLDT takes 3 orders-of-magnitude than longer that MimicLDT on small graphs. Furthermore, when applied to graphs larger than Cora, we found that MetaLDT ran out-of-memory. We observed that much of the time and memory was spent in the inner-training step used by meta-learning. Therefore, we estimated the time of running MetaLDT on larger graphs by running 20 inner training epochs only for each optimization iteration. We estimate that MetaLDT runtime for Cora, PubMed and ArXiv are 8465.39s, 114515.46s and 2396240.51s respectively. We conclude that MetaLDT costs grow significantly as graph sizes increase, and it is thus impractical to use this attack with larger graphs.

One might wonder if reducing the hyperparameters such as the number of optimization iterations for MetaLDT would allow it to scale to larger graphs and achieve better poison success rate than MimicLDT. We evaluated this by measuring the number of optimization rounds required by MetaLDT to achieve the same poison success rate as MimicLDT on the Cora dataset. We reports these results in Appendix B.4.4, where Figure B.1 shows that MetaLDT requires 8 optimization rounds and 9670.46s to achieve the same poison success rate as MimicLDT can in 43.17 seconds. This is because MimicLDT considers a smaller search space (only considering a fixed set of randomly selected attack points) and uses a cheaper surrogate metric for loss. These differences also contribute to the MimicLDT's lower poison success rate.

More analysis on MetaLDT. Ablation studies about the design of MetaLDT are left in Appendix,

Table 3.2: Total running time (in seconds) and GPU memory cost of generating one poisoned graph for various datasets on GCN model.

	Graph size		MimicLDT		MetaLDT	
Dataset	Nodes	Edges	Time(s)	Mem.	Time(s)	Mem.
Cora	2708	5429	43.17	1.38GB	82198.92	2.91GB
PubMed	19717	44338	105.82	1.89GB	—	OOM
ArXiv	169343	1157799	692.44	9.51GB	—	OOM

such as exploring the design rationale of the optimization process (§B.4.1) and optimization constraints (§B.4.3). Discussion about the benefits to optimizing the adjacency matrix is in §B.4.2. And we study the effects of hyperparameters (§B.4.4).

3.5.3 Comparing with Short-distance Baselines

For comparing with short-distance baselines, we consider two different types of the short-distance attacks: short-distance modification attacks, and the node injection poisoning attacks.

Comparing with short-distance modification attacks. Table. 3.1 also shows the range of performance achieved by existing short-distance modification attacks, including Nettack-direct (modifying the target's immediate neighbors), Nettack-indirect(modifying the target's k-hop neighborhood), FGA and IG-FGSM. Detailed results are in §B.7 (Table B.10, B.11). We set the short-distance attack budget to be 34 for Cora and 182 for PubMed (i.e., $p * \Delta$, see details of setups in §B.6.1) and leave experiments with varying perturbation budgets in §B.6.2. From the results, we can see that short-distance modification attacks can achieve higher success rate, especially for direct attacks, often at 100%. For indirect attacks, they usually show higher success rate on vanilla models (53-79% for Cora, 82-100% for PubMed), however their success rate can greatly drop on some robust models (e.g., 18% for SVD and 46% for SoftMedianGDC on Cora). Moreover, short-distance attacks are susceptible to GNN analysis tools such as GNNExplainer [206]. Later (Figure. 3.5), we show that GNNExplainer could detect the short-distance attacks with reasonable precision and recall, and break the attacks after the diagnosis.

Comparing with node-injection poisoning attack. We also evaluated the efficacy of **AFGSM** [185], a well cited recent node-injection poisoning attack. Unfortunately, we cannot compare our system with most existing injection attacks, including, TDGIA [225], GIA-HAO [30], CANA [164], G^2A2C [96], because these are test-time evasion attacks rather than training-time poisoning attacks.

For our comparison, we modified the original implementation, which misclassified the target nodes to be any arbitrary classes, to one that performs targeted label-flipping poisoning. We used this modified version to perform both direct attacks (injected nodes can be direct neighbors of the target node) and indirect attacks (injected nodes cannot be direct neighbors but can be k-hop neighbors). For direct attacks, we allowed the attack to inject up to 2 nodes, and to add up to 2 times the average node degree edges to the graph. For the indirect attack, we used the same hyper parameters as in our work, allowing the attack to add up to $\Phi \times r \times |\mathcal{V}_L|$ ($\Phi = 4$) nodes and $p \times \Phi \times r \times |\mathcal{V}_L|$ edges.

We provide detailed results from this comparison in Appendix B.7 (Table B.10 and B.11), and only provide a summary here. We found that the direct-attack when using vanilla GCN has a poison success rate of 74% for Cora and 83% for PubMed, which is between 7–12% higher than MimicLDT (67% and 71%). However, the direct attacks is also more easily defended against, and we found that for robust models the poison success rate drops to between 3-69% for Cora and 10-79% for PubMed. By contrast, MimicLDT seems less affected by the use of robust approaches (55-74% for Cora, 56-70% for PubMed).

MimicLDT outperforms the indirect-attack for both vanilla GC and robust variants: our evaluation found that the indirect-attack's poison success rate varied between 30-64% for Cora and 15-45% for PubMed.

Table 3.3: Changes in the distribution of graph node degrees and homophiliy, measured using *Earth Mover's Distance*, for the Cora dataset. The values represent the average distance between each poisoned graph and the original input. We report on other datasets in Appendix. B.5.2 and B.5.3.

	MimicLDT	MetaLDT
Degree changes	0.0393 ± 0.0021	0.0419 ± 0.0055
Homophily changes	0.0205 ± 0.0010	0.0142 ± 0.0015

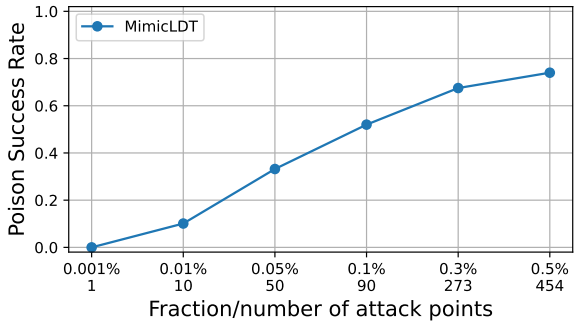


Figure 3.3: Poison success rate with varying number of attack points for GraphSAGE on ArXiv.

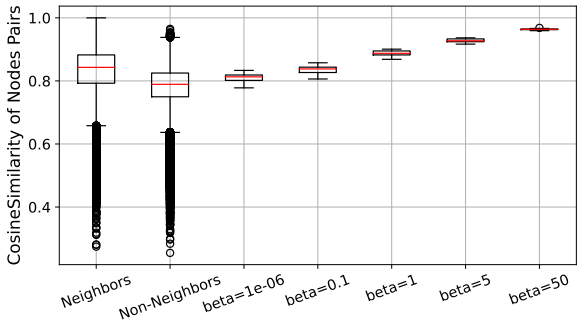


Figure 3.4: Similarity between injected nodes (varying β), their attack points, and between neighboring and non-neighboring nodes for GraphSAGE on ArXiv.

3.5.4 Ablation Studies of MimicLDT

Effect of varying the number of attack points. We study the effect of varying the number of attack points. Fig.3.3 shows poison success rate as r varies (while keeping Φ fixed). Increasing r, and thus the number of attack points, improves attack success rate. §B.5.1 studies the effects of varying Φ. The total number of injected nodes is determined by both r and Φ.

Attack stealthiness: degree distribution. We examine whether poisoned graphs can preserve the node degree distribution. We measure the changes to degree distribution using the Earth Mover's Distance (EMD) metric. The average distance between each poisoned graph with original clean graph for Cora is 0.039 ± 0.002. Statistics on other datasets can be found in § B.5.2. The attacks only cause slight changes on the node degree distribution.

Attack stealthiness: homophily. The second term of MimicLDT's loss function (Eq. 3.4) keeps injected nodes similar to the attack points they attach to. It serves a similar goal as prior work [30], which is to ensure that injected nodes do not significantly impact graph homophily. The hyper-parameter β controls the importance of the second term. Figure 3.4 measures the similarity of neighboring and non-neighboring nodes in the ArXiv graph, and compare them to the similarity between injected nodes and their attack points with varying β. Appendix§ B.5.3 gives the detailed setup. We can observe that the larger the value of β, the more similar injected nodes appear to their attack points. As we note in §3.4, we use feature vector similarity as a proxy for the standard node-centric homophily metric [30]. In §B.5.3, we show this does not affect the homophily results.

3.5.5 End-to-end attacks

The attacks generated by MetaLDT and MimicLDT inject fake nodes whose features lie in continuous space. Thus, they are not end-to-end attacks for graphs with discrete features, such as citation graphs whose raw node features are natural language texts. Thus, an end-to-end attack needs to inject nodes with textual features. We extend our design to perform such an attack.

Suppose some language model such as SciBERT [13] is used to encode a node's raw texts to an embedding vector in continuous space. Our extension trains a decoder that can generate texts given an embedding vector, which corresponds to some fake node's feature as computed by MimicLDT (or MetaLDT). We provide more details on the design and evaluation of end-to-end attack in Appendix§B.8 and give example texts generated for the fake nodes (§B.8 Fig B.7).

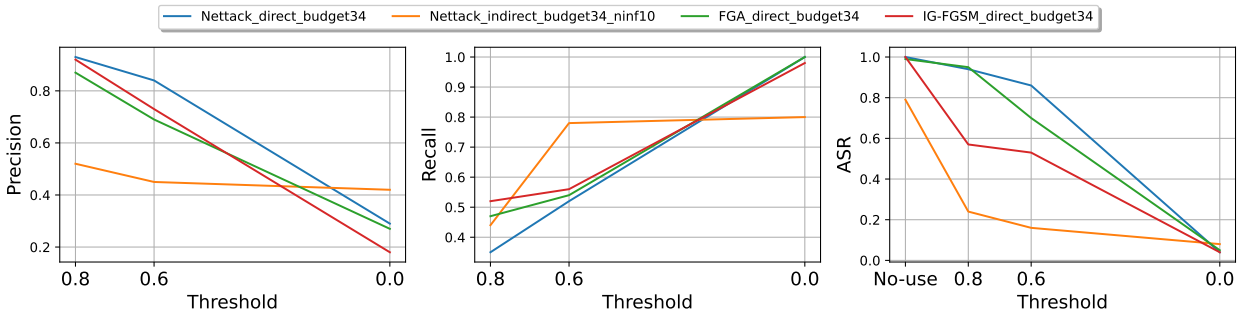


Figure 3.5: Effect of GNNExplainer on various short-distance attacks over Cora dataset under detection rate (precision/recall) and attack success rate (ASR) after removing the selected adversarial edges and retraining for prediction. GNNExplainer selects only those edges whose importance scores are above the user-specified *threshold*. See details about settings of *budget* and *ninf* (number of influencer) in §B.6.1.

3.6 Diagnosing Challenges

3.6.1 Local Diagnosis Tool

Short-distance attacks are susceptible to GNN diagnosis tools (e.g., GNNExplainer), but MimicLDT cannot be diagnosed by such local explainability tools – since MimicLDT operates outside the regions.

As shown in [52], diagnosis tools like GNNExplainer can provide great opportunities for human (inspectors or system designers) to inspect the confused predictions from adversarial perturbations. By modifying the target node's k-hop neighborhood, baseline short-distance attacks are vulnerable to existing GNN explanability tools. We evaluate this by measuring the likelihood that GNNExplainer can detect perturbations from the baseline attacks. In Figure. 3.5, we show the precision and recall of using GNNExplainer to detect the adversarial perturbations by setting different importance score thresholds. Also, the attack success rate drops from 79-100% (before diagnosis) to 3-7% (for threshold 0.0) if we removed the detected malicious edges and retrained the models for the target node's prediction. Details about the tools and more metrics analysis (NDCG score, F1 score) are in Appendix. B.6.4

3.6.2 Apply AME to MimicLDT

Fundamental challenge: node dependencies in graph data. The application of the Average Marginal Effect (AME) (Chapter 2) to Graph Neural Networks (GNNs) presents unique challenges that distinguish it from traditional ML/DL tasks. In conventional classification tasks involving images, natural language processing, or multi-graph datasets, data points are typically treated as independent samples. However, node classification tasks in graph neural networks fundamentally violate this *independence assumption*.

In graph-structured data, nodes exhibit inherent dependencies and structural relationships that are crucial for the learning process. This dependency structure is particularly evident in adversarial scenarios such as the MimicLDT attack. In MimicLDT, the attack mechanism relies on the coordinated interaction between multiple components: a long-distance *attack point* provides the target label, while its connected attacker-injected unlabeled *attachment nodes* contribute poisoned features. These components must work synergistically to enable successful poisoning of the target model.

Theoretical limitations of direct AME application. The original AME framework relies on the fundamental assumption that data points are *independent and identically distributed (i.i.d.)*. This assumption is essential for the regression-based (i.e., LASSO) estimation procedure that underlies AME's diagnostic capabilities. When applied to node classification tasks, treating each node as an independent data source directly violates this core assumption.

The violation occurs because the final hidden representation of any given node in a graph neural network is not solely determined by its own features, but is influenced by the features and representations of its neighbors through the *message-passing* mechanism. This creates a complex web of dependencies that renders the independence assumption untenable. Consequently, the original AME framework cannot be directly applied to graph-based poisoning attacks without significant modifications to account for the inherent structural dependencies in graph data.

3.7 Adapting AME to Diagnose

3.7.1 Approach: Identify Poisoned Subgraphs

Theoretical foundation: subgraph-based independence. To address the fundamental challenge of node dependencies while maintaining compatibility with the original AME theoretical framework, we propose an approach that shifts the unit of analysis from individual nodes to subgraphs. This approach is grounded in the observation that while individual nodes are interdependent, the *final hidden representations of different labeled nodes can be considered relatively independent for the purpose of model parameter updates.*

We define a subgraph for each labeled node v_i as $subgraph(v_i)$, which encompasses the labeled attack point node v_i and its k-hop neighborhood, where k corresponds to the number of layers in the GNN model. This definition is theoretically motivated by the fact that in a k-layer GNN, the final hidden representation of a node v_i is determined by the aggregation of information from nodes within its k-hop neighborhood.

Rationale for subgraph-level analysis. The ***subgraph-based approach*** addresses the independence assumption in the following way: for each labeled node, the final hidden representation used to update model parameters is propagated and integrated only within its subgraph's scope. While nodes within a subgraph are interdependent, the final hidden representations of different labeled nodes (and their corresponding subgraphs) can be considered sufficiently independent for the purpose of model parameter updates. This reformulation allows us to maintain the theoretical foundations of AME while adapting it to the graph domain. Instead of detecting individual poisoned nodes, we focus on identifying *poisoned subgraphs*, which represents a more natural unit of analysis for graph-based attacks.

Modified sampling strategy. To implement this subgraph-based approach, we modify the original AME sampling strategy as follows:

- Restricted Sampling Domain: Randomly sample only from the original labeled (training) nodes V_L, rather than from the entire node set.
- Neighborhood Inclusion: When a labeled node is sampled, include its k-hop neighborhood as well. All neighbors function as unlabeled nodes within the subgraph context.
- Parameter Update Scope: Use only the sampled nodes for updating model parameters when training multiple submodels during the diagnostic process.

This modified sampling strategy ensures that the diagnostic process respects the natural boundaries of information flow in GNNs while maintaining the statistical properties required for AME estimation.

AME estimation procedure. Following the modified sampling strategy, we apply the standard AME estimation procedure:

(a) Feature Matrix X: Each row corresponds to a labeled node $v_i \in V_L$, and each column indicates the include/exclude status of that labeled node in a particular submodel.

(b) Response Vector Y: The confidence score of the target node v_t being predicted as the target label y_t.

(c) Regression Analysis: Apply AME estimation using LASSO regression (and other techniques) to estimate the contribution of each labeled node (i.e., the $subgraph(v_i)$) to the model's behavior.

3.7.2 Experiments and Results

We evaluate our approach on the Cora citation network dataset using Graph Convolutional Networks (GCN) as the base model. The experimental configuration consists of $1,708$ labeled nodes with 17 attack points, representing 1% of the labeled nodes. For the sampling parameters, we use a total of $2,937$ submodels calculated as $cklog(s)$ with $c = 16$, and set the sampling probability q to 0.

Table 3.4: Results of diagnosing MimicLDT using adapted AME.

	Pure LASSO	LASSO+Knockoffs
Precision	0.481	0.739
Recall	1.0	1.0

We evaluate the effectiveness of the diagnosis across 50 queries using both pure-LASSO and LASSO+Knockoffs, the results are shown in Table. 3.4.

The experimental results demonstrate a characteristic pattern: high recall with reasonable (relatively lower) precision. This indicates that our approach successfully identifies all poisoned subgraphs (perfect recall) but produces some false alarms (lower precision). The perfect recall is particularly significant from a security perspective, as it ensures that no poisoned subgraphs go undetected. The false alarm rate, while present, represents a manageable trade-off in security-critical applications where false negatives (missed attacks) are usually more costly than false positives (false alarms).

The improved precision with the Knockoffs (0.739 vs 0.481) suggests that the techniques explored and applied in our original AME framework continue to be effective when adapted to GNN architectures.

Post-diagnosis defense effectiveness. After identifying poisoned subgraphs, we implement a direct defense strategy by excluding the identified attack points during training. The effectiveness of this approach is demonstrated through complete mitigation of the poisoning attack, with the poisoning accuracy reduced to 0.0 after removal – ignoring during model parameters updates – of the identified attack points. Importantly, this defense strategy has minimal impact on the model's performance on clean data, with clean node accuracy decreasing only marginally from 0.8795 before removal to 0.8786 after removal.

The results show that our diagnostic approach enables effective defense with minimal impact on the model's performance on clean data.

Fine-grained attack node localization. We can continue to consider implementing a more

Table 3.5: Results of applying AME directly for dependent data sources.

Strategy	Precision (Knockoffs)	Recall (Knockoffs)
S1 (Feature Zeroing)	0.333	0.573
S2 (Node Removal)	1.0	0.074
Our Approach	0.739	1.0

fine-grained approach to locate specific attachment nodes within identified poisoned subgraphs, if needed. This approach leverages the design characteristics of MimicLDT. *Leave-One-Out Analysis:* Within each identified poisoned $subgraph(v_i)$, we systematically zero out each node's features and measure the change in distance between the final representations of the attack node v_a and the target node v_t. Nodes whose exclusion maximally increases this distance are identified as likely malicious attachment nodes.

3.7.3 Ablation Studies and Lessons Learned

Towards applying AME directly for dependent data sources (nodes). To understand the necessity of our subgraph-based approach, we conducted ablation studies exploring naive applications of AME that ignore node dependencies. These experiments involved directly sampling nodes from the full graph and applying AME without accounting for structural relationships.

We tested several alternative sampling strategies:

- Strategy S1 - Feature Zeroing: Sample from the entire node set (both labeled and unlabeled), and zero out the features of unsampled nodes while maintaining the graph structure.
- Strategy S2 - Node Removal: Sample from the entire node set, and completely remove unsampled nodes along with their adjacent edges.

As shown in Table. 3.5, the results from these alternative strategies highlight the importance of our subgraph-based approach.

Key insights and lessons learned:

(1) Lesson 1 - Dependency Awareness is Critical: The poor performance of naive approaches (S1 and S2) demonstrates that ignoring node dependencies fundamentally undermines the diagnostic capability of AME in graph domains.

(2) Lesson 2 - Structural Preservation: Strategy S2's extremely low recall (0.0735) despite perfect precision (1.0) suggests that completely removing nodes destroys critical structural information necessary for accurate diagnosis.

(3) Lesson 3 - Feature vs. Structure: Strategy S1's more balanced but still poor performance indicates that feature-level modifications alone are insufficient when structural dependencies remain unaddressed.

3.7.4 Broader Implications

The lessons learned from this work extend beyond the specific application to MimicLDT and graph neural networks. They highlight a general principle: when applying established machine learning techniques to new domains with different structural assumptions, careful consideration must be given to how fundamental assumptions translate to the new context. The success of our subgraph-based approach demonstrates that with appropriate modifications, powerful techniques like AME can be effectively adapted to new domains while preserving their core theoretical foundations and practical utility.

4 | Reasoning Models Diagnostic: Probe as Hidden Verifier

This chapter addresses whether reasoning models know when they're right, using probes as hidden verifiers. Reasoning models have achieved remarkable performance on tasks like math and logical reasoning thanks to their ability to search during reasoning. However, they still suffer from *overthinking*, often performing unnecessary reasoning steps even after reaching the correct answer. This raises the question: *can models evaluate the correctness of their intermediate answers during reasoning?* In this work, we study whether reasoning models encode information about answer correctness through probing the model's hidden states. The resulting probe can verify intermediate answers with high accuracy and produces highly calibrated scores. Additionally, we find models' hidden states encode correctness of future answers, enabling early prediction of the correctness before the intermediate answer is fully formulated. We then use the probe as a verifier to decide whether to exit reasoning at intermediate answers during inference, reducing the number of inference tokens by 24% without compromising performance. These findings confirm that reasoning models do encode a notion of correctness yet fail to exploit it, revealing substantial untapped potential to enhance their efficiency.

4.1 Overview

Recent advances in reasoning models, such as OpenAI's o1 [143] and DeepSeek-R1 [44], have demonstrated significant progress in complex reasoning capabilities, particularly in domains such as mathematical problem solving [43, 222] and logical reasoning [55, 111, 124]. A key advantage of reasoning models lies in their ability to search: they often explore multiple *reasoning paths* leading to different *intermediate answers* to the original problem before arriving at a final solution (Figure 4.1, left). While this search-based reasoning is beneficial, it also introduces inefficiencies. Previous studies [28, 161] show that reasoning models tend to *overthink* by exploring additional reasoning paths even after reaching a correct answer.

This observation prompts the question: *to what extent can models evaluate the correctness of their intermediate answers during reasoning?* The answer to this question is also crucial to preventing overthinking, either through a more targeted design of the training strategy or a better elicitation method. We investigate this question by probing the model's hidden states for answer correctness. Specifically, we segment the long Chain-of-Thought (CoT) into chunks containing intermediate answers, and train a binary classifier to predict answer correctness from the model's hidden states at the answer positions (Figure 4.1).

We find that information about answer correctness is readily encoded in the model's internal representations. A simple probe can reliably extract this information, performing accurately on both in-distribution and out-of-distribution examples. Moreover, the probe is highly calibrated, with an expected calibration error (ECE) below 0.1. Our analysis also reveals that the model's hidden states contain "look-ahead" information: correctness can be predicted even before the intermediate answer is fully articulated. Notably, when applying the same probing method to traditional short CoT models, we observe a significant degradation in performance, suggesting that the encoded correctness information is likely acquired during long CoT training.

We also investigate whether reasoning models effectively use this information on answer cor-

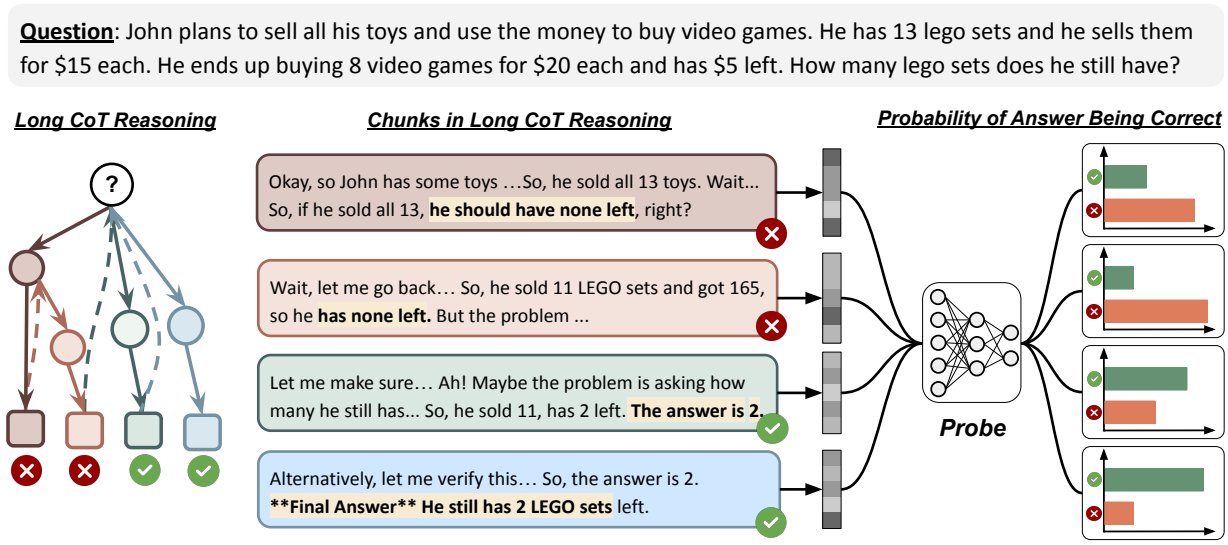


Figure 4.1: An illustration of the probing method. On the left side, long CoT is parsed into multiple chunks, each corresponding to a reasoning path and contains an intermediate answer as termination. On the right side, representations for each chunk are obtained and probe is used to predict the probability of answer being correct.

rectness during inference. Because the trained probe is well-calibrated, we use the output score to measure the model's *confidence* in the current intermediate answer. Ideally, the model should reason at an optimal length if it is taking advantage of the well-encoded correctness information, i.e. it should stop reasoning when the confidence about an intermediate answer is high enough. We adopt the probe as a verifier and implement a confidence-based early-exit strategy by thresholding confidence scores from the probe. The strategy achieves up to 24% reduction in inference tokens without compromising accuracy. The improvement in efficiency with our verifier reveals that while reasoning models encode information about answer correctness, they do not efficiently use this internal knowledge during inference.

4.2 Motivation and Novelty

The question we want to study is whether reasoning models represent correctness of intermediate answers during reasoning, which is crucial to the success of the unique search behavior

(characterized by backtracking to previous steps during reasoning) exclusively present in reasoning models. We note that the answer to the question is not obvious. Despite the overall success in reasoning effectiveness, the observed overthinking phenomenon seems to suggest the other way around (discussed in the Introduction Section). Existing works that report discrepancy [**<empty citation>**] in model behavior and inner state information and the revealed information in model hidden representation [**<empty citation>**] further blur the conclusion. Our results also show that these models are distinct from non-reasoning models—exhibiting substantially better calibration—and suggest that further research is essential for understanding what and how reasoning models learn during training.

As stated above, the motivation stems from **search behavior and observed "overthinking", which is unique to reasoning models**. We also demonstrate in Section **??** that reasoning models exhibit significantly better-calibrated notions of correctness than their non-reasoning counterparts. The universally low calibration error along with high accuracy (even on smaller models like DS-Distill-Qwen-1.5B) is remarkable compared to previous works that probe non-reasoning models' hidden states to test models' ability to know its own knowledge boundary [**<empty citation>**] or detect hallucination [3]. The technique we adopt is also much simpler than prior works on non-reasoning models' representation [**<empty citation>**]. This observation further *underscores the uniqueness of reasoning models and reinforces the necessity for dedicated investigation into their distinctive properties.*

4.3 Probe for Intermediate Answer Correctness

The long CoT output from a reasoning model often contains multiple mentions of *intermediate answers*. We aim to explore whether the notion of "correctness" is encoded in the representation of each intermediate answer by probing. This section describes how we identify intermediate answers, obtain their representations, and train a two-layer multilayer perceptron (MLP) probe.

4.3.1 Data Collection

We first collect responses from reasoning models for each problem in the task dataset. The reasoning trace, which is encapsulated in <*think*> tokens, is extracted and split into paragraphs with "\n\n" as delimiter. We identify the start of a new reasoning path by detecting keywords like "wait", "double-check" and "alternatively" in each paragraph. A complete list of the keywords is shown in Table C.1 in the appendix. We merge paragraphs in the same reasoning path to form a *chunk*. Then we use Gemini 2.0 Flash [63] to extract the intermediate answer in each chunk if one exists, and judge its correctness against the true answer. Finally, adjacent chunks that do not contain an intermediate answer are merged with the closest chunk that contains an answer. Each merged chunk now has an intermediate answer and a label generated by Gemini, represented as $\{(c_1, y_1), (c_2, y_2), ...(c_k, y_k)\}$, where each c_i is part of the reasoning trace that contains an answer to the original problem, and y_i is a binary label indicating the correctness of the answer.

The next step is to obtain the model representation for each chunk. For each chunk c_i, we take the last-layer hidden states at the last token position as its representation e_i. Finally, for each task dataset, we collect a set of reasoning representations and their corresponding labels, formulating the probing dataset $\mathcal{D} = \{(e_i, y_i)\}_{i=1}^{N}$ that will be finally used to train probes. Note that the construction of probing dataset $\mathcal{D}$ depends on both the original task dataset and the reasoning model we use to generate representations.

4.3.2 Training the Probe

After obtaining the probing dataset, we train a two-layer multilayer perceptron on $\mathcal{D}$. Since the datasets are often highly imbalanced, where most intermediate answers from a strong reasoning models are correct (see Table C.2 in Appendix **??** for detailed label statistics), we use weighted

binary cross-entropy loss:

$$
\begin{aligned}
p_i &= \sigma(\mathrm{ReLU}(e_i \mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + b_2) \\
\mathcal{L}(\mathbf{W}, \mathbf{b}) &= -\frac{1}{N} \sum_{i=1}^{N} (w \alpha y_i \log p_i + (1 - y_i) \log(1 - p_i))
\end{aligned}
\tag{4.1}
$$

where σ is the sigmoid function, w is the ratio of negative to positive samples in the training data, and α is a hyperparameter to scale the imbalance weight. The model parameters are $\mathbf{W}_1 \in \mathbb{R}^{m \times d}$, $\mathbf{W}_2 \in \mathbb{R}^{d \times 1}$, $\mathbf{b}_1 \in \mathbb{R}^d$, and $b_2 \in \mathbb{R}$, where m is the hidden size of the language model and d is the hidden size of the MLP.

4.4 Experiments

We first describe the basic experimental setup (§ 4.4.1). Then, we explore whether information about answer correctness is encoded in reasoning models (§ 4.4.2) and if it generalizes across datasets (§ 4.4.3), how such information is related to long CoT reasoning abilities (§ 4.4.4), and is the information also well-encoded even before an explicit answer is formulated (§ 4.4.5).

4.4.1 Experimental setup

Task datasets. We select mathematical reasoning and logical reasoning tasks as their answers are automatically verifiable. For mathematical reasoning, we use three datasets: GSM8K [34], MATH [81], and AIME. For logical reasoning, we use KnowLogic [211], a logical reasoning benchmark of 5.4k multiple-choice questions synthesized with knowledge-driven methods. To ensure the reliability of intermediate answer extraction, we filter the KnowLogic dataset to only retain examples with a single correct answer. For ease of training, all training sets are down-sampled to include no more than 1000 examples, which did not affect performance according to our pilot experiment. See Appendix C.1.1 for more details regarding data processing.

Reasoning models. We use the open-source DeepSeek-R1-Distill series of models [44], including R1-Distill-Llama-8B, R1-Distill-Llama-70B, R1-Distill-Qwen-1.5B, R1-Distill-Qwen-7B, and R1-Distill-Qwen-32B. All the distilled models are supervised fine-tuned with reasoning data generated by DeepSeek-R1 model. We also use QwQ-32B [169, 203], an open-source reasoning language model trained with reinforcement learning.

Implementation details. For probing data collection, we enumerate each combination of task dataset and model to collect model representation and answer labels. The statistics of the collected data can be found in Appendix C.1.1. For training, each dataset $\mathcal{D}$ is randomly split into a training set and a validation set $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$, with a train-to-validation ratio of 8:2. The Adam optimizer [100] is used for training, and we perform grid search for hyperparameter tuning. The hyperparameters for search include learning rate, scaling factor for imbalance weight α, weight decay, and MLP hidden size d. Each model is trained for at most 200 epochs with a batch size of 64; the validation loss is used as the criterion for early stopping. Following grid search, the probing models are first ranked based on their validation accuracy. From the top 10 performing models, we select the probe with the least number of parameters, specifically the model with the smallest hidden dimension d. Details regarding the grid search setting and search results for each probing dataset can be found in Appendix C.1.3. Note that most resulting models achieve non-trivial performance when $d = 0$ (see Appendix C.1.3), which means that correctness of the intermediate answer can be easily extracted with a linear probe.

4.4.2 Reasoning models encode answer correctness

We first test **in-distribution** performance of trained probes by evaluating each probe on the test set from the same dataset as the training set. Figure 4.2 reports the ROC-AUC scores on each dataset, and Table 4.1 presents the corresponding Expected Calibration Error (ECE) [139] and Brier score [18]. Other metrics including accuracy, precision, recall, and macro F1 are reported in Appendix C.1.4.

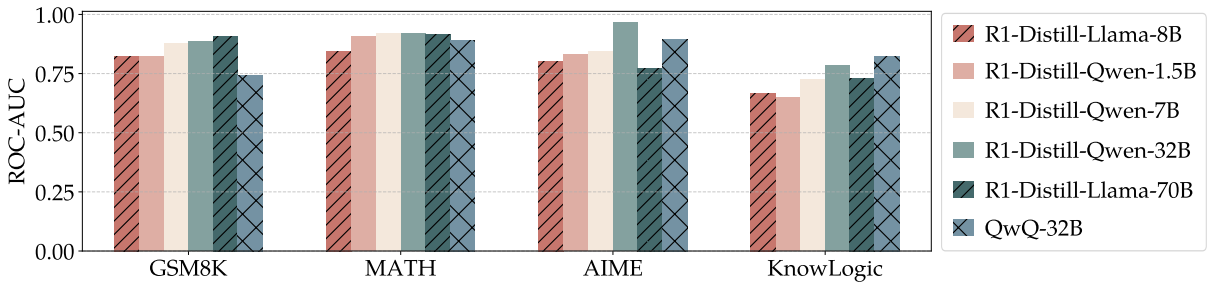


Figure 4.2: ROC-AUC scores for each probe trained on hidden states from different reasoning models and datasets. We train a separate probe on each probing dataset and evaluate it on in-distribution test set.

Overall, all probes perform satisfactorily in in-distribution setting, achieving ROC-AUC scores above 0.7 and remarkably low Expected Calibration Error (ECE) scores below 0.1. This indicates the reasoning models inherently encode information about answer correctness that can be extracted with a simple probe. Moreover, many of the probes converge to a linear probe after grid search (hidden size $d = 0$), suggesting that correctness information is linearly encoded in the hidden states of the reasoning model (Appendix C.1.3). Across task datasets, probes trained on mathematical reasoning data perform better than those trained on logical reasoning data. This may correlate with the training data distributions of the reasoning models, where math problems presumably play a larger role. Meanwhile, probes extracted from larger reasoning models work better, with R1-Distill-Qwen-32B achieving over 0.9 ROC-AUC score on AIME. The Qwen family models' representations also exhibit stronger correctness signals, with Qwen-1.5B generally surpassing Llama-8B model in the mathematical domain, potentially reflecting differences in the base model training data distribution.

4.4.3 Probes generalize to some out-of-distribution datasets

Past studies have shown that probe performance can deteriorate significantly when applied to out-of-distribution data [12, 98]. Since strong in-distribution results may not necessarily indicate reliable generalization, we examine how well the probes trained in § 4.4.2 perform across different

Reasoning Model	GSM8K		MATH		AIME		KnowLogic	
	ECE ↓	Brier ↓	ECE ↓	Brier ↓	ECE ↓	Brier ↓	ECE ↓	Brier ↓
R1-Distill-Llama-8B	0.05	0.17	0.03	0.14	0.10	0.11	0.07	0.23
R1-Distill-Llama-70B	0.03	0.07	0.07	0.10	0.10	0.18	0.03	0.19
R1-Distill-Qwen-1.5B	0.04	0.16	0.04	0.12	0.14	0.12	0.09	0.20
R1-Distill-Qwen-7B	0.02	0.11	0.03	0.10	0.09	0.15	0.06	0.21
R1-Distill-Qwen-32B	0.01	0.08	0.06	0.09	0.13	0.10	0.10	0.19
QwQ-32B	0.03	0.13	0.13	0.10	0.08	0.13	0.03	0.15

Table 4.1: Expected Calibration Error (ECE) and Brier score for the in-distribution performance of each probe trained on each probing dataset.

domains and datasets.

Table 4.2 shows the ROC-AUC and ECE scores for probes evaluated on out-of-distribution data, compared to those trained and tested on in-distribution data, using representations from R1-Distill-Llama-8B. We find that probes exhibit generalizability across mathematical reasoning datasets. The probes trained on MATH and GSM8K transfer well between the two datasets, demonstrating both high discriminative performance (ROC-AUC) and satisfactory calibration (ECE). In contrast, for AIME, a more difficult dataset, the probes trained on GSM8K and MATH are less calibrated. However, the probe does not stably generalize to out-of-domain data (e.g., from logical reasoning to mathematical reasoning), perhaps due to the difference in distribution of the two domains (Figure C.1). More generalization results on other reasoning models can be found in Appendix C.1.4.

4.4.4 Encoding of correctness is related to long CoT reasoning abilities

We have shown information on answer correctness is encoded in reasoning model's hidden states; to what extent this encoding is related to the model's ability to perform long CoT reasoning? To that end, we train a probe with the non-reasoning counterpart of the reasoning model. Specifically, we use Llama-3.1-8B-Instruct [68] to obtain representations of reasoning chunks us-

Training Data	GSM8K		MATH		AIME		KnowLogic	
	AUC ↑	ECE ↓	AUC ↑	ECE ↓	AUC ↑	ECE ↓	AUC ↑	ECE ↓
GSM8K	0.82	0.05	0.80 (-0.04)	0.08 (+0.05)	0.69 (-0.11)	0.25 (+0.15)	0.56 (-0.11)	0.10 (+0.03)
MATH	0.83 (+0.01)	0.04 (-0.01)	0.84	0.03	0.76 (-0.04)	0.28 (+0.18)	0.63 (-0.04)	0.08 (+0.01)
KnowLogic	0.77 (-0.05)	0.17 (+0.12)	0.74 (-0.10)	0.19 (+0.16)	0.81 (+0.01)	0.31 (+0.21)	0.67	0.07

Table 4.2: ROC-AUC scores and ECE of trained probes on out-of-distribution test set. The numbers in red and green denote performance decrease and increase relative to the probe trained on in-distribution training set, respectively. R1-Distill-Llama-8B is used as the reasoning model.

ing the MATH dataset. As instruct models do not have long CoT reasoning abilities, each chunk is just the full model output for one problem (i.e., including the short CoT and final answer), and the representation is simply the hidden state of the last token for each problem output. To account for this, we add an additional setting for reasoning model probes, where the probe is evaluated on the correctness of the final answers (rather than the intermediate answers) of each reasoning chain.

As shown in Figure 4.3, the probe trained on non-reasoning model representations performs much worse than its reasoning counterpart, with lower classification scores and higher calibration errors. The fact that the encoded information on answer correctness is more prominent in reasoning models may suggest that the self-verification ability is enhanced during long CoT supervised training.

4.4.5 Correctness can be detected before the answer is generated

Section 4.4.2 shows that the hidden states at the *end* of reasoning chunks encode information about intermediate answer correctness, we now investigate a further question: do hidden states from earlier positions within the chunk also encode such signals? Specifically, we analyze hidden states from varying positions *midway* through a reasoning chunk—before an intermediate answer

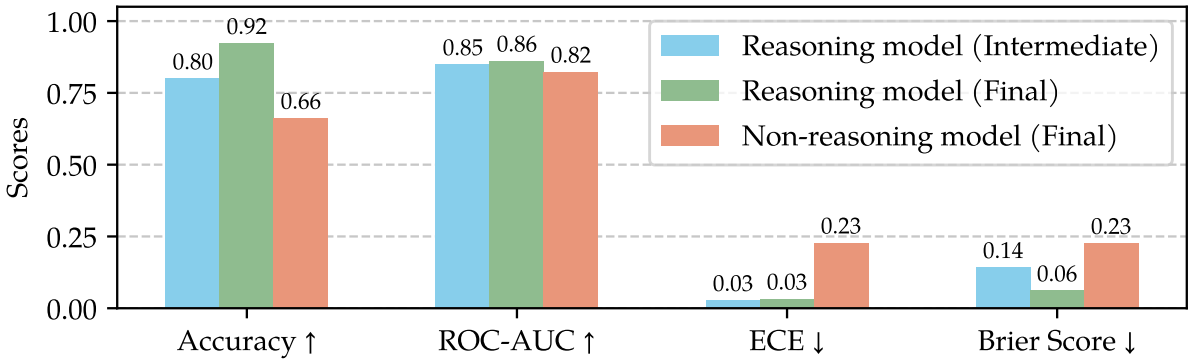


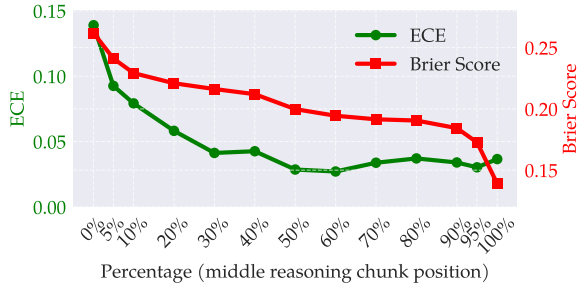
Figure 4.3: Comparison on the performance on reasoning models (i.e., R1-Distill-Llama-8B, fine-tuned on the base Llama-3.1-8B model using long CoT data) and non-reasoning models (i.e., Llama-3.1-8B-Instruct) on MATH. For reasoning models, we show both the performance on predicting the correctness of intermediate answers (blue) and the final answers (green). For non-reasoning models, the data only contains the final answers (red).

is fully generated—to determine if these earlier representations already encode predictive signals about the forthcoming answer's correctness.

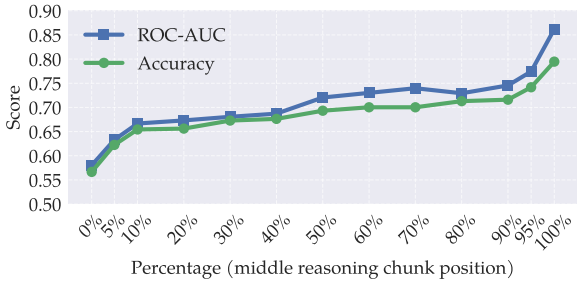
As described in § 4.3.1, each reasoning trace is initially split into k chunks with corresponding correctness labels $\{(c_1, y_1), (c_2, y_2), ...(c_k, y_k)\}$. Each chunk c_i can be subdivided into paragraphs. We obtain the representation of each paragraph-level sequence, and assign each sequence within chunk c_i the label y_i, corresponding to the correctness of the nearest upcoming intermediate answer. We train a probe to predict the future answer correctness for R1-Distill-Llama-8B on MATH (following § 4.3.2). We use hidden states at the end of different paragraphs to predict chunk correctness. We report probing performance at different percentages of all paragraphs within a chunk.

We observe that the reasoning model's hidden states encode information about correctness even before an intermediate answer has been explicitly generated. Moreover, the probe performance is positively correlated with the paragraph's proximity to the upcoming intermediate answer. As shown in Figure 4.4, the probe's classification accuracy improves primarily during two critical phases: an initial steep increase in the 0-10% range, followed by minimal gains until a second noticeable improvement near the chunk's end (90-100%). Compared to the peak accuracy

of 79%, performance at the 10%, 50%, and 95% positions shows decrements of 14%, 10%, and 5% respectively. This highlights that early positions contain significant correctness signals, while the most predictive information emerges just before answer generation. On the other hand, calibration error is highest at the initial paragraph and then undergoes a sharp decline. ECE reaches its minimum (0.03) relatively early——at around the 60% position——while the Brier score continues improving until the final positions of the reasoning chunk.



(a) ECE and Brier Score decrease as the paragraph position approaches the answer at the end of the reasoning chunk



(b) Accuracy and ROC-AUC increase as the paragraph position approaches the generated answer at the end of the reasoning chunk

Figure 4.4: Performance on predicting the correctness of the upcoming intermediate answers midway through a reasoning chunk. The results are obtained at different percentages of all paragraphs within each chunk. The task dataset and reasoning model used are MATH dataset and R1-Distill-Llama-8B.

4.5 Efficient Inference: Early-exit

4.5.1 Probe as a Verifier for Early-exit

While reasoning models are able to encode well-calibrated and accurate information about intermediate answer correctness, do they fully utilize it during inference? We investigate this by checking whether early exiting based on the probe's confidence score on answer correctness can improve reasoning efficiency. This approach allows us to determine whether models continue reasoning unnecessarily after the probe is highly confident that the answer is correct (i.e., overthinking).

paragraphExperimental setup

Following § 4.3, we obtain a classifier that takes a reasoning chunk c_i's representation e_i as input and outputs the probability p_i of the intermediate answer y_i being correct. Since the estimated p_i is highly calibrated (§ 4.4.2), we directly use it to guide **confidence-based early-exit** during inference. Specifically, we first set a threshold Thr for model confidence. Then, we sequentially evaluate each intermediate answer in the full reasoning trace, using the probe to compute confidence scores on the answer's correctness. Once we encounter an intermediate answer whose probed p_i exceeds the threshold Thr, we truncate the reasoning trace at this chunk and take the intermediate answer as the final answer.

We compare the intermediate answer selected by early exiting with the question's ground-truth answer to compute accuracy. Additionally, we record the inference token length at the point of truncation to evaluate computational efficiency. We run R1-Distill-Llama-8B on MATH dataset. In this experiment, the maximum token generation limit is set to be 10K across all test examples.

For comparison, we implement **static early-exit**, where we predetermine a fixed number of intermediate answers m and terminate the reasoning process after m chunks, taking the m-th chunk's intermediate answer y_m as the final answer[1].

4.5.2 Experimental Results

As shown in Figure 4.5, using the probe to perform confidence-based early exiting can improve reasoning efficiency without accuracy degradation. When setting Thr to 0.85, our strategy achieves roughly the same reasoning accuracy (88.2%) as no early-exit, while reducing the number of generated tokens by approximately 24%. Setting Thr to 0.9 (or higher) can achieve identical reasoning accuracy (88.6%) as no early-exit and reduces the number of generated tokens by 19%. In other words, without early exiting, the reasoning model continues to generate excess tokens

[1]Note that the static early-exit strategy degrades to no early-exit if the total number of chunks $k < m$.

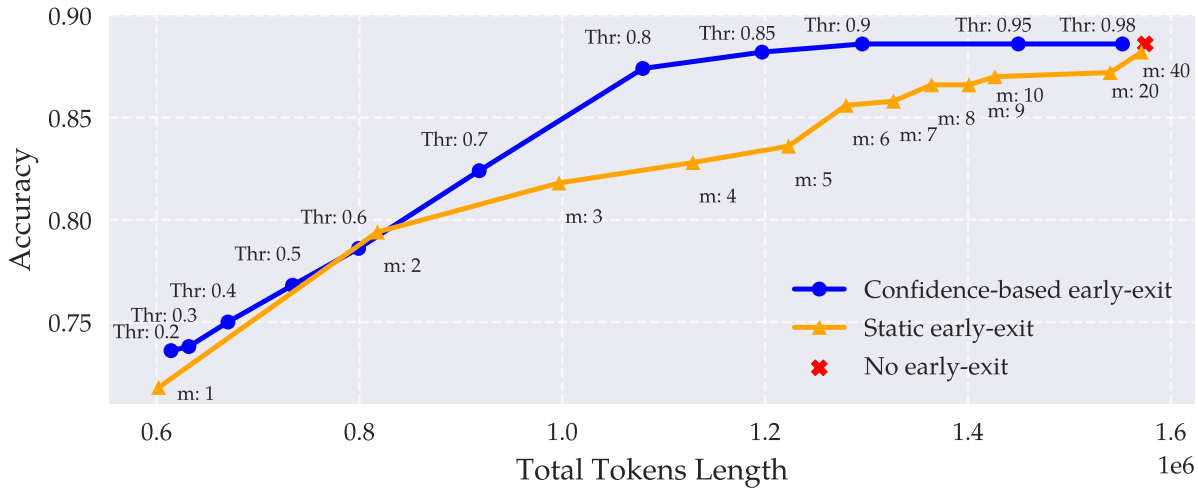


Figure 4.5: Final answer accuracy versus inference token cost with different early-exit strategies. For confidence-based early-exit, the curve is obtained by varying the confidence threshold for answer correctness. For static early-exit, the curve is generated by varying the chunk number m.

even when the probe indicates high confidence; this failure to fully utilize internal information on answer correctness empirically leads to overthinking behavior.

Additionally, when saving equivalent numbers of tokens, our approach outperforms the static early-exit strategy by achieving up to a 5% accuracy improvement. For instance, confidence-based early exiting has 87.4% accuracy ($Thr = 0.8$), whereas the static early-exit strategy has approximately 82.5% accuracy with similar total token usage. Controlling for the same accuracy score (e.g. above 85%), confidence-based early-exit strategy ($Thr = 0.8$) consumes significantly fewer tokens than static strategy (with $m = 6$). This demonstrates that leveraging the internal encoded information of answer correctness as an exit strategy can lead to more efficient reasoning.

Overall, the improvements suggest that reasoning models fail to fully leverage this internal encoded information of answer correctness during inference, and that more effective usage of the information can reduce overthinking and enhance reasoning efficiency.

4.6 Related Work

Uncertainty estimation in LLMs. Black-box techniques for estimating LLM uncertainty over their response have primarily focused on prompting the model to verbalize its confidence directly, often aggregating self-reported confidence scores over multiple samples [120, 171]. However, Kapoor et al. [98] and Xiong et al. [199] find that white-box methods, including those that depend on internal model representations [135], tend to perform better than black-box methods on confidence estimation. For instance, Azaria and Mitchell [5] and Burns et al. [20] show that an LLM's representation after processing a statement is highly predictive of the statement's correctness; moreover, linear probes trained on these representations can classify correctness, even without ground-truth labels. We extend this work to long CoT generated by reasoning models, demonstrating that the representations at intermediate stages of the CoT also capture key information about the correctness of each intermediate stage.

Efficient reasoning during inference. Reasoning models demonstrate improved performance on many tasks thanks to their ability to search while generating reasoning chains, which often demand additional test-time compute in comparison to standard CoT [44]. Additionally, reasoning models often suffer from repeated and unnecessary reasoning steps—or "overthinking"—even after a correct answer has been reached [28]. Recent work has explored training methods to make reasoning more concise or to reduce the frequency of overthinking [28, 138]. Other inference-time techniques focus on curtailing generations that are unlikely to be successful [118, 133, 219] or dynamically adjusting the test-time compute budget based on input difficulty or other properties of the prompt [41, 59, 189, 202]. We find that while the model encodes information about answer correctness, it fails to use it efficiently, which may contribute to overthinking. We leverage this to perform threshold-based early-exiting at inference time, reducing test-time compute while preserving performance.

Learned verifiers. The ability to verify intermediate answers is also related to the line of works

on verifiers, which is an important technique used to regulate test-time scaling. Previous work has focused on training verifiers to classify the correctness of a model-generated solution or select which of two model-generated responses is preferred [7, 34, 38, 119, 144, 215, 220]. However, recent improvements in reasoning capabilities have enabled LLMs to critique and refine their outputs without the aid of external verifiers, often using natural language prompt templates to guide self-critique of model-generated output [122, 129, 159, 192, 218]. In contrast, we focus on leveraging information about correctness which is encoded in the model representations of the reasoning chain.

4.7 Discussion

In this study, we explore the existence of answer correctness information in reasoning models' inner representation. With probing, we show that such information is readily accessible in models' hidden states. The trained probe demonstrate strong calibration performance, and can be adopted as a lightweight verifier to improve reasoning efficiency. The significant reduction in inference tokens suggest that reasoning models' hidden states probably contain rich information that are underexplored.Our findings contribute to the growing body of research on model interpretability and open up several intriguing avenues for future investigation.

Self-verification ability of language models. Our study reveals that answer correctness is encoded in reasoning models' hidden states. The information can be easily extracted with a probe and used as a verifier during inference. This indicates that strong self-verification abilities can be elicited from reasoning models. Notably, these abilities are less pronounced in non-reasoning models. However, given the intricate training processes and the diversity of training data these models are exposed to, the precise origins of this ability remains unclear, suggesting a promising avenue for future research into how and when such self-verification abilities emerge during model training.

Internal mechanisms of reasoning models. We uncover a surprisingly well-calibrated hidden verifier that enables models to autonomously assess intermediate reasoning correctness. This finding suggests that models possess an ability to self-verify, which is an important step toward understanding their internal decision-making processes. However, we still observe "overthinking" phenomenon, where models perform unnecessary re-checks even after generating correct answers with high confidence, as demonstrated in our early-exit experiment. This suggests that while models can self-verify, they do not yet efficiently leverage this intrinsic capability. Further study is needed to explore how reasoning models internally utilize the information encoded in their representations, and how we can guide them to use this information more efficiently during training or inference.

On-policy control of reasoning models. In contrast to previous LLM-based verifiers [34, 215, 218], the hidden verifier extracted in our work is much more lightweight. Our approach leverages the hidden states of the reasoning model directly during inference, which not only improves token efficiency but also makes the verifier more integrated with the model's existing architecture. Our finding highlights the potential of an on-policy perspective in model inference control. We believe this opens new avenues for future research in designing more efficient and adaptive control modules for reasoning models.

In summary, our study highlights the encoded answer correctness information in reasoning models, indicating the latent capability of reasoning models to verify their own answers. Leveraging this information through lightweight probing techniques, we show reasoning efficiency can be further enhanced, implying an inadequate use of the information by reasoning models during inference. Our findings underscore the potential of on-policy control for reasoning models, offering a novel direction for more efficient and adaptive inference strategies. Future research should further investigate the origins of the self-verification abilities and develop methods to better harness them, ultimately improving reasoning efficiency and reliability.

5 | Conclusion

This dissertation has addressed the fundamental challenge of diagnosing AI misbehavior across shifting deep learning paradigms through three complementary contributions. For classifiers, we introduced the Average Marginal Effect (AME), a scalable data attribution method achieving efficient attribution with only O(k log N) evaluations. For Graph Neural Networks, we developed MimicLDT, a long-distance targeted attack revealing blind spots in existing explainability tools, and demonstrated AME's adaptability to graph structures. For large reasoning models, we designed self-verification probes that uncover latent correctness signals, enabling computational savings through confidence-based early-exit strategies.

This work establishes a comprehensive framework for model-aware diagnostics, demonstrating that diagnostic approaches must evolve alongside AI paradigms while principled tools can transcend specific domains. The research advances both theoretical understanding of AI misbehavior and practical tools for building more trustworthy, efficient, and interpretable AI systems.

The broader impact extends beyond technical contributions to responsible AI development across critical domains. By providing methods to understand and mitigate AI failures, this research contributes to building public trust, ensuring regulatory compliance, and preventing costly real-world failures. As AI systems become increasingly sophisticated, these diagnostic frameworks provide essential tools for maintaining their safety, reliability, and accountability.

5.1 Future Directions

The work presented in this dissertation opens several promising avenues for future research in AI diagnostics. The following directions emerge naturally from the limitations and insights gained from our three complementary contributions.

Scalable Topology-Aware GNN Diagnostics. Our stress testing approach with MimicLDT reveals the need for more sophisticated diagnostic tools that can handle the complex topological dependencies in graph neural networks. Future work should develop scalable topology-aware diagnostic methods that can efficiently analyze large-scale graph structures and their influence on model behavior. Additionally, topology-optimized AME sampling and estimation techniques are needed to better handle dependent data sources in graph settings, moving beyond the current adaptations that treat subgraphs as independent units.

Universal Self-Diagnostic Layers for Large Language Models. The success of our self-verification probes suggests a natural extension toward embedding universal self-diagnostic capabilities directly into large language models. Rather than training separate probes post-hoc, future research could explore integrating self-awareness mechanisms during model training, enabling models to continuously monitor and adjust their own reasoning processes. This could lead to more robust and self-correcting AI systems that can dynamically adapt their behavior based on internal confidence signals. This implementation would transform model diagnostics from post-hoc diagnosis into a proactive and preemptive tool, enabling more real-time and effective model monitoring.

Reasoning Models v.s. Non-reasoning Models. A fundamental question that emerges from our work on reasoning models is understanding what and how these models learn during training

that differs from non-reasoning models. Future research should investigate the training dynamics that give rise to the latent correctness signals we observed, exploring how different training objectives, data compositions, and architectural choices influence the development of self-verification capabilities. This understanding could inform the design of more effective training procedures for reasoning-capable AI systems.

Diagnosing Other Advanced AI Model Paradigms. As AI continues to evolve, new model architectures and paradigms will inevitably emerge, each bringing unique diagnostic challenges. Future work should proactively develop diagnostic frameworks for increasingly advanced AI models, including multimodal systems, agent-based models, and hybrid reasoning architectures. The principles established in this thesis—adaptability, scalability, and model-awareness—provide a foundation for extending diagnostic capabilities to these emerging paradigms.

A | Appendix: Supplementary Materials for Chapter 2

A.1 Convergence Rate when Using LASSO to compute AME

We first state and prove a variant of the LASSO error bound result from [112] in a simpler setting, which is sufficient for our application and will serve as the foundation of many of our results. We then apply this result to finish the proof of $O(k \log N)$ sample rate of our AME estimator left in Prop. 2.4 in the main body.

A.1.1 Simplified LASSO Error Bound

Proposition A.1. *Consider a regression problem where one wants to approximate an unknown random variable Y using a set of random variables $X_1, \ldots, X_N$ with $N \geq 3$. Let $\hat{\beta}_{lasso}$ be the LASSO coefficient when regressing Y on X with L_1 regularization, and β^* be the best linear fit (detailed definition in Prop. 2.3). Assume that $\forall n \in [N] : \beta_n^*$ is finite, X_n is bounded in $[A, B]$, Y is bounded in $[0, 1]$. Further assume that $\forall i, j \in [N] : \mathbb{E}[X_i X_j] = 0$ when $i = j$ and otherwise 1. If β^* has at most k non-zeros and the sample size $M \geq k(1 + \log(k/N))$, then there exists a regularization parameter*

λ and a value $C(B-A,\delta)$ such that with probability at least $1-\delta$:

$$\|\hat{\beta}_{lasso} - \beta^*\|_2 \leq C(B-A,\delta)\sqrt{\frac{k\log(N)}{M}}.$$

The following terminology will be useful to facilitate its proof.

Definition A.2. X is said to be a subgaussian random variable with variance property σ^2 if for any $s \in \mathbb{R}, \mathbb{E}[\exp(sX)] \leq \exp\left(\frac{\sigma^2 s^2}{2}\right)$. We write $X \sim subG(\sigma^2)$.

Definition A.3. The underlying measure of a random vector $X \in \mathbb{R}^N$ is said to be L-subgaussian if for every $q \geq 2$ and every $u \in \mathbb{R}^N$,

$$\mathbb{E}^{1/q}[|\langle u, X\rangle|^q] \leq L\sqrt{q}\mathbb{E}^{1/2}[|\langle u, X\rangle|^2].$$

We now prove proposition A.1:

Proof. To apply Theorem 1.4 from [112] to bound the error, we need our setup to satisfy Assumption 1.1 of that paper: that X is an isotropic, L-subgaussian measure, and that the noise is in L_q for $q > 2$.

First, the isotropic requirement is that $\mathbb{E}[\langle u, X\rangle^2] = \langle u, u\rangle$ for all $u \in \mathbb{R}^N$. This can be shown by observing that $\mathbb{E}[\langle u, X\rangle^2] = \sum_{i=1}^N \mathbb{E}[u_i^2 X_i^2] + \sum_{i\neq j} \mathbb{E}[u_i u_j X_i X_j] = \sum_{i=1}^N u_i^2$, where the last equality comes from the fact that $\mathbb{E}[X_i X_j] = 0$ and $\mathbb{E}[X_i^2] = 1$.

The second requirement is that the probability measure of the covariate vector X is L-subgaussian (Def. A.3). Let $\sigma = (B-A)/2$. Then $X_n \sim subG(\sigma^2)$ since they are bounded. Hence, $\langle u, X\rangle \sim subG(\sigma^2)$ (see e.g., Theorem 2.1.2 in [145]). Applying Proposition 3.2 from [153], we have $\mathbb{E}^{1/q}[|\langle u, X\rangle|^q] \leq L\sqrt{q}$ with $L > 0$ a constant dependent only on σ. Noticing that Def. A.3 remains equivalent when constraining $\|u\|_2 = 1$, and that $\mathbb{E}^{1/2}[\langle u, X\rangle^2] = \|u\|_2 = 1$ concludes this part of proof.

Third the "noise", defined as $\xi = \langle \beta^*, X\rangle - Y$ should be in L_q, $q > 2$. Notice that the best estimator cannot be worse than a zero estimator that always return 0, thus $\mathbb{E}[\xi^2] \leq 1$. Hence,

$|\xi| \le |\langle \beta^*, X\rangle| + |Y| \le DN + 1$, where D is the upper bound of $|\beta_n^* X_n|$, then there exists $q = 1/\log(DN+1) + 2 > 2$ such that $\mathbb{E}[|\xi|^q] \le \mathbb{E}[|\xi|^2](DN+1)^{q-2} \le \mathbb{E}[|\xi|^2] \cdot e \le e$.

Combined with the sparsity assumption on β^*, we can apply Theorem 1.4 from [112], which directly yields the error bound.

□

A.1.2 Proof of Prop. 2.4

With this simplified bound, we can prove Prop. 2.4:

Proof. Recall that the best linear estimator $\beta_n^* = \mathrm{AME}_n/\sqrt{v}$, so applying Prop. A.1 directly yields the bound. The remaining work is to verify the assumptions: Since the AME is an average of utility differences, we know that $\beta_n^* = \mathrm{AME}_n/\sqrt{v} \in [-1/\sqrt{v}, 1/\sqrt{v}]$ is finite. Furthermore, by design $\mathbb{E}[X_i X_j] = \mathbb{E}_p[\mathbb{E}[X_i X_j|p]] = \mathbb{E}_p[\mathbb{E}[X_i|p]\mathbb{E}[X_j|p]] = 0$ when $i \neq j$, and $\mathbb{E}[X_i X_j] = \mathbb{E}[X_i^2] = Var[\mathbb{E}[X_i^2]] + \mathbb{E}[X_i]^2 = 1$ when $i = j$. With the assumptions all verified, applying Prop. A.1 concludes proof. □

A.2 An AME Estimator with p-featurization

Recall that to estimate $\mathrm{AME}_n(\mathcal{P})$ defined under some given distribution $\mathcal{P}$, we have been using a featurization where $\mathrm{X}[m,n]$ takes values of either $\frac{1}{\sqrt{v}p}$ or $\frac{-1}{\sqrt{v}(1-p)}$, depending on whether data point/source n is respectively included or excluded in the subset for row m, with $v = \mathbb{E}_{p\sim\mathcal{P}}[\frac{1}{p(1-p)}]$. However, values for $\mathrm{X}[m,n]$ blow up quickly as p approaches 0 or 1, which leads to unbounded feature values for certain $\mathcal{P}$ that samples such p values often. Unfortunately such distributions can be useful in some cases, in particular to derive a low-error SV approximations (e.g., with the beta$(1+\varepsilon, 1+\varepsilon)$ with small ε (§A.3.3)). Below we propose a different featurization that solves this issue while still ensuring an $O(k \log N)$ sample rate.

Specifically, we define $X_n = \sqrt{v}(1-p)$ if $n \in S$ and $X_n = -\sqrt{v}p$ otherwise. These values are clearly bounded when $\sqrt{v}$ is finite. To ensure that the best linear fit still recovers AME (a.k.a. Prop. 2.3)—an important property we use to derive the $O(k \log N)$ error bound—we adapt the distribution where samples (X, Y) are drawn in LASSO. Recall that source inclusion is a compound distribution, in which we first draw $p \sim \mathcal{P}$ for the entire subset, and then X_n's according to p. Here, we change they way p is drawn, by imposing a $\frac{1}{p(1-p)}$-weighting over $\mathcal{P}$. Formally, if we note $f_{\mathcal{P}}(\cdot)$ the probability density function (PDF) of $\mathcal{P}$ which we used with the original $1/p$ featurization, we now draw p from the distribution with reweighted PDF $f_{\mathcal{W}}(p) = f_{\mathcal{P}}(p)\frac{1}{p(1-p)} / \mathbb{E}_{p\sim\mathcal{P}}[\frac{1}{p(1-p)}]$.

Denote this new sampling scheme by $p \sim \mathcal{W}$, and note that the AME is still defined under the original distribution $\mathcal{P}$, while $\mathcal{W}$ is only used to sample (X, Y) for LASSO. We show that the best linear fit on (X, Y) is still $\text{AME}_n(\mathcal{P})$:

Proposition A.4. *Let β^* be the best linear fit on (X, Y):*

$$\beta^* = \arg\min_{\beta\in\mathbb{R}^n} \mathbb{E}\left[(Y - \langle\beta, X\rangle)^2\right], \tag{A.1}$$

then $AME_n(\mathcal{P})/\sqrt{v} = \beta^*_n,\ \forall n \in [N]$, with $v = \mathbb{E}_{p\sim\mathcal{P}}[\frac{1}{p(1-p)}]$.

Proof. For a linear regression, we have (e.g. Eq. 3.1.3 of [4]):

$$\beta^*_n = \frac{Cov(Y, \tilde{X}_n)}{Var[\tilde{X}_n]},$$

where $\tilde{X}_n$ is the regression residual of X_n on all other covariates $X_{-n} = (X_1, \ldots, X_{n-1}, X_{n+1}, \ldots, X_N)$. In our design $\mathbb{E}[X_n|X_{-n}] = \mathbb{E}_{p\sim\mathcal{P}}[X_n|p] = 0$, implying $\tilde{X}_n = X_n - \mathbb{E}[X_n|X_{-n}] = X_n$. Therefore:

$$\beta^*_n = \frac{Cov(Y, \tilde{X}_n)}{Var[\tilde{X}_n]} = \frac{Cov(Y, X_n)}{Var[X_n]} = \frac{\mathbb{E}[X_n Y]}{Var[X_n]}.$$

Notice that $\mathbb{E}[X_n Y] = \mathbb{E}_{p\sim\mathcal{W}}[\mathbb{E}[X_n Y|p]]$ with:

$$\begin{aligned}\mathbb{E}[X_n Y|p] &= p \cdot \mathbb{E}[X_n Y|p, n \in S] + (1-p) \cdot \mathbb{E}[X_n Y|p, n \notin S] \\ &= \sqrt{v}p(1-p)\mathbb{E}[Y|p, n \in S] + \sqrt{v}(1-p)(-p)\mathbb{E}[Y|p, n \notin S] \\ &= \sqrt{v}p(1-p)(\mathbb{E}[Y|p, n \in S] - \mathbb{E}[Y|p, n \notin S])\end{aligned}$$

Combining the two previous steps yields:

$$\begin{aligned}\beta_n^* &= \frac{\mathbb{E}_{p\sim\mathcal{W}}[\sqrt{v}p(1-p)(\mathbb{E}[Y|p, n \in S] - \mathbb{E}[Y|p, n \notin S])]}{Var[X_n]} \\ &= \frac{\sqrt{v}\mathbb{E}_{p\sim\mathcal{P}}[\mathbb{E}[Y|p, n \in S] - \mathbb{E}[Y|p, n \notin S]]}{v \cdot Var[X_n]} \\ &= \frac{AME_n}{\sqrt{v} \cdot Var[X_n]}.\end{aligned}$$

Noticing that

$$Var[X_n] = \mathbb{E}[Var[X_n|p]] + Var[\mathbb{E}[X_n|p]] = \int_0^1 f_{\mathcal{W}}(p)(pv(1-p)^2 + (1-p)v(-p)^2)dp$$
$$= v\int_\varepsilon^{1-\varepsilon} \frac{1}{v(1-2\varepsilon)}dp = 1$$

concludes the proof. □

Moreover, $\mathbb{E}[X_i X_j] = \mathbf{1}\{i = j\}$ for the same reason as in §??. Hence, we can still apply Prop. A.1 to derive the same LASSO error bound:

Proposition A.5. *If X_n's are bounded in $[A, B]$ and $N \geq 3$, there exist a regularization parameter λ and a constant $C(B-A, \delta)$ such that when the sample size $M \geq k(1+log(k/N))$, with probability at least $1-\delta$:*

$$\|\hat{\beta}_{lasso} - \frac{1}{\sqrt{v}}AME\|_2 \leq C(B-A, \delta)\sqrt{\frac{k\log(N)}{M}},$$

with $v = \mathbb{E}_{p\sim\mathcal{P}}[\frac{1}{p(1-p)}]$ *and* AME_n *defined under* $\mathcal{P}$.

Compared with the original, $1/p$-featurization, the difference only lies in the constant factor $C(B-A,\delta)$, since $B-A$ has changed.

A.3 Sparse Estimators for the Shapley Value from the AME

Recall that the AME is the SV when $\mathcal{P} = Uni(0,1)$. However, this choice is incompatible with our fast convergence rates for LASSO. To find a good estimator of the SV from the AME, it is thus crucial to understand the discrepancy between the AME and the SV introduced by different distributions $\mathcal{P}$ over p, in order to bound the SV from the AME with a cP compatible with good convergence rates. Here we first derive general error bounds between SV and AME that work for all distributions. Then we apply it to two specific distributions: namely the truncated uniform and Beta distributions. We mainly focus on sparse SV and/or AME under bounded or monotone utility, but to make the discussion clearer, no assumptions are made on either the SV or AME unless explicitly stated.

Throughout this section, we denote by $P_{\text{AME}}(S)$ and $P_{\text{SV}}(S)$ the probability of sampling subset S when $p \sim \mathcal{P}$ and $p \sim Uni(0,1)$, respectively (the P_{SV} name is due to the fact that AME is SV under this distribution, see Prop. 2.1). We also introduce the following notation:

$$\Delta \triangleq \max_S \frac{P_{\text{AME}}(S)}{P_{\text{SV}}(S)} - 1. \tag{A.2}$$

A.3.1 General bounds

Lemma A.6. *Assume a bounded utility function with range in* $[0,1]$. *Then*

$$\|AME - SV\|_\infty \leq 2\Delta.$$

Further assume a monotone utility (Assumption 2.6). Then:

$$\|AME - SV\|_2 \leq \Delta + \sqrt{2\Delta}.$$

Proof. The L_∞ error bound is due to the following:

$$\begin{aligned} |\mathrm{AME}_n - \mathrm{SV}_n| &= \Big| \sum_S \big(P_{\mathrm{AME}}(S) - P_{\mathrm{SV}}(S)\big)\big(U(S+\{n\}) - U(S)\big)\Big| \\ &\leq \sum_S |P_{\mathrm{AME}}(S) - P_{\mathrm{SV}}(S)| = 2 \sum_{S: P_{\mathrm{AME}}(S) > P_{\mathrm{SV}}(S)} P_{\mathrm{AME}}(S) - P_{\mathrm{SV}}(S) \leq 2\Delta. \end{aligned} \tag{A.3}$$

For the L_2 error, its square $\|\mathrm{AME} - \mathrm{SV}\|_2^2$ can be divided into two groups based on the sign of $\mathrm{AME}_n - \mathrm{SV}_n$. Call those indices n with positive (negative) sign $\mathcal{N}^+$ ($\mathcal{N}^-$). For all $n \in \mathcal{N}^+$,

$$\mathrm{AME}_n - \mathrm{SV}_n = \sum_S \big(P_{\mathrm{AME}}(S) - P_{\mathrm{SV}}(S)\big)\big(U(S+\{n\}) - U(S)\big) \leq \sum_S \Delta P_{\mathrm{SV}}(S)(U(S+\{n\}) - U(S)) = \Delta \mathrm{SV}_n, \tag{A.4}$$

where the last inequality is due to $U(S + \{n\}) > U(S)$ implied by monotonicity. For the same reason, all SV_n's are positive, implying $\mathrm{SV}_n \leq U([N]) \leq 1$. Thus $(\mathrm{AME}_n - \mathrm{SV}_n)^2 \leq \Delta^2 \mathrm{SV}_n$. On the other hand, for all $n \in \mathcal{N}^-$, we know that $|\mathrm{AME}_n - \mathrm{SV}_n|$ is bounded by 2Δ; it is also bounded by SV_n since AME_n cannot be negative under monotone utility. Summing up these bounds gives $\|\mathrm{AME} - \mathrm{SV}\|_2 \leq \sqrt{\sum_{n\in\mathcal{N}^+} \Delta^2 \mathrm{SV}_n + \sum_{n\in\mathcal{N}^-} 2\Delta \mathrm{SV}_n} \leq \sqrt{\sum_{n\in\mathcal{N}^+} \Delta^2 \mathrm{SV}_n} + \sqrt{\sum_{n\in\mathcal{N}^-} 2\Delta \mathrm{SV}_n} \leq \Delta + \sqrt{2\Delta}$, where the last inequality is due to $\|\mathrm{SV}\|_1 \leq 1$. □

In fact the bounded utility assumption is quite minor when we assume monotonicity: every $U(S)$ is bounded between the empty set and full set utility. Given that by definition of SV $U(\emptyset) = 0$, it reduces to an assumption of $U([N]) \leq 1$. In practice as long as $U([N])$ is a known and bounded constant (e.g., the accuracy on the validation set of the model trained on the full set), one can simply scale the utility to meet this requirement. In what follows, when we say monotone utility we mean both monotone and bounded.

A.3.2 SV Estimator from the AME under a Truncated Uniform distribution

We prove the results from the paper's main body, before discussing p-featurization.

Proof of Lemma 2.7. As a reminder, Lemma 2.7 states an L_2 error bound between the AME and SV under monotone utility, when AME is defined with $\mathcal{P} = Uni(\varepsilon, 1-\varepsilon)$.

Proof. Lemma A.6 already gives us $\|\text{AME} - \text{SV}\|_2 \leq \Delta + \sqrt{2\Delta}$ (Δ from Eq. A.2). All that remains is to show that $\Delta \leq 4\varepsilon$:

$$\Delta = \max_S \frac{P_{\text{AME}}(S)}{P_{\text{SV}}(S)} - 1 \tag{A.5a}$$

$$(j \leftarrow |S|) = \max_j \left(N \int_\varepsilon^{1-\varepsilon} \frac{1}{1-2\varepsilon} \binom{N-1}{j} p^j (1-p)^{N-j-1} dp - 1 \right) \tag{A.5b}$$

$$\leq \max_j \left(N \cdot \frac{1}{1-2\varepsilon} \int_0^1 \binom{N-1}{j} p^j (1-p)^{N-j-1} dp - 1 \right) \tag{A.5c}$$

$$= \max_j \left(N \cdot \frac{1}{1-2\varepsilon} \cdot \frac{1}{N} - 1 \right) = \frac{1}{1-2\varepsilon} - 1 \tag{A.5d}$$

$$\leq 4\varepsilon, \tag{A.5e}$$

where the equality A.5d is due to the fact that the Binomial(p) with $p \sim Uni(0,1)$ is a discrete uniform. Hence, $\|\text{AME} - \text{SV}\|_2 \leq \Delta + \sqrt{2\Delta} \leq 4\varepsilon + 2\sqrt{2\varepsilon}$. □

Lemma A.6 also yields to the following result, which fills in the missing piece in the proof of Corollary 2.8—the one that states the $O(k \log N)$ bound for this SV estimator:

Corollary A.7. *A k-sparse SV implies a k-sparse AME under monotone utility.*

Proof. Under montone utility, both AME_n and SV_n are non-negative. By Eq. A.4, when $\text{SV}_n = 0$, $\text{AME}_n \leq \Delta\text{SV}_n = 0$. □

Non-monotone utility. When the utility is no longer monotone, we can still derive an $O(k \log N)$ rate in terms of L_∞ error for SV estimation, under an additional assumption that the AME is k-sparse. Indeed, first notice that the bound $\Delta \leq 4\varepsilon$ from Eq. A.5a does not require monotonicity. Under utility bounded in $[0, 1]$, applying the first part of Lemma A.6 yields:

$$\|\text{AME} - \text{SV}\|_\infty \leq 2\Delta \leq 8\varepsilon. \tag{A.6}$$

Notice that this bound, as the L_2 bound, does not depend on N or k. Hence, applying the same arguments as in Corollary 2.8 yields an L_∞ error bound we presented in the main body (Corollary 2.5). We reiterate it here for convenience of reading:

Corollary A.8. *When AME is k-sparse and the utility is bounded in $[0, 1]$, for every constant $\varepsilon > 0, \delta > 0, N \geq 3$, there exists constants $C_1(\varepsilon, \delta)$, ε', and a LASSO regularization parameter λ, such that when the number of samples $M \geq C_1(\varepsilon, \delta) k \log N$, $\|\sqrt{v}\hat{\beta}_{lasso} - SV\|_\infty \leq \varepsilon$ holds with a probability at least $1 - \delta$, where $v = \mathbb{E}_{p\sim Uni(\varepsilon', 1-\varepsilon')}[\frac{1}{p(1-p)}]$.*

Proof. $\|\sqrt{v}\hat{\beta}_{lasso} - \text{SV}\|_\infty \leq \|\sqrt{v}\hat{\beta}_{lasso} - \text{AME}\|_2 + \|\text{AME} - \text{SV}\|_\infty$. By the sparsity of AME, we apply Proposition 2.4 to bound the first term by $\varepsilon/2$, and Eq. A.6 with $\varepsilon' = \varepsilon/4$ to bound the second term by the same, concluding the proof. □

The main obstacle to deriving an L_2 bound in this more general setting comes from the lack of an L_2 error bound between the AME and the SV that is independent of N. Note that one may derive an L_2 error bound from Eq. A.6 as follows: $\|\text{AME} - \text{SV}\|_2 \leq \sqrt{N}\|\text{AME} - \text{SV}\|_\infty \leq 4\sqrt{N}\varepsilon$. However, this bound is now dependent on N, which violates the precondition of applying the LASSO error bound (see Prop. 2.4).

Using p-featurization. As pointed out in §A.2, p-featurization also achieves a $O(k \log N)$ sample rate to reach a low L_2 error in estimating the AME. Since p-featurization has no effect on the AME value, the bound between the AME and the SV remains the same. We thus reach the same

conclusion as for $1/p$-featurization:

Corollary A.9. *For every constant $\varepsilon > 0, \delta > 0, N \geq 3$, there exists constants $C_2(\varepsilon, \delta)$, ε', and a LASSO regularization parameter λ, such that with $M \geq C_2(\varepsilon, \delta) k \log N$, and with probability at least $1 - \delta$:*

(1) $\|\sqrt{v}\hat{\beta}_{lasso} - SV\|_\infty \leq \varepsilon$ holds when the utility is bounded in $[0, 1]$ and the AME is k-sparse;

(2) $\|\sqrt{v}\hat{\beta}_{lasso} - SV\|_2 \leq \varepsilon$ holds when the utility is monotone and the SV is k-sparse,

where the AME is defined under $\mathcal{P} = Uni(\varepsilon', 1 - \varepsilon')$ and $v = \mathbb{E}_{p\sim\mathcal{P}}[\frac{1}{p(1-p)}]$.

Proof. Conclusion (1) follows the same proof as Corollary A.8, and Conclusion (2) follows Corollary 2.8. □

A.3.3 SV Estimator from the AME under a Beta Distribution

Another candidate to estimate the SV from the AME is to use $\mathcal{P} = \text{Beta}(1 + \varepsilon, 1 + \varepsilon)$ as the distribution of p, with $\varepsilon \in (0, 0.5)$, and using p-featurization[1]. We show an $O(k \log N)$ sample rate in this setting as well, after introducing two necessary lemmas.

Lemma A.10. *For any $x \geq 1, y \geq 1$ and $\varepsilon \in (0, 0.5)$, the following holds:*

$$\frac{(x + \varepsilon - 1)^\varepsilon (y + \varepsilon - 1)^\varepsilon}{(x + y + 2\varepsilon)^{2\varepsilon}} < \frac{\mathrm{B}(x + \varepsilon, y + \varepsilon)}{\mathrm{B}(x, y)} < \frac{(x + \varepsilon)^\varepsilon (y + \varepsilon)^\varepsilon}{(x + y + 2\varepsilon - 1)^{2\varepsilon}},$$

where $\mathrm{B}(\cdot, \cdot)$ is the Beta function.

Proof. According to Gautschi's inequality, for all $a > 0, s \in (0, 1)$,

$$a^{1-s} < \frac{\Gamma(a + 1)}{\Gamma(a + s)} < (a + 1)^{1-s},$$

[1]The $1/p$-featurization in incompatible here, since this distribution can draw "p"s arbitrarily close to 0 or 1 which leads to unbounded feature values, violating the assumption of the $O(k \log N)$ LASSO rate (Prop. 2.4).

where $\Gamma(\cdot)$ is the Gamma function. For all $\varepsilon' \in (0,1)$ and $b \geq 1$, we can change variables with $s \leftarrow 1-\varepsilon' \in (0,1)$ and $a \leftarrow b+\varepsilon'-1 > 0$ to obtain:

$$(b+\varepsilon'-1)^{\varepsilon'} > \frac{\Gamma(b+\varepsilon')}{\Gamma(b)} > (b+\varepsilon')^{\varepsilon'}. \tag{A.7}$$

In addition, we have that:

$$\mathrm{B}(x+\varepsilon, y+\varepsilon)/\mathrm{B}(x,y) = \frac{\Gamma(x+\varepsilon)}{\Gamma(x)}\frac{\Gamma(y+\varepsilon)}{\Gamma(y)}/\frac{\Gamma(x+y+2\varepsilon)}{\Gamma(x+y)}$$

Plugging Eq. A.7 into the above concludes the proof. □

Lemma A.11. *When $p \sim \mathrm{Beta}(1+\varepsilon, 1+\varepsilon)$ with $\varepsilon < 0.5$, then if the utility is bounded in $[0,1]$,*

$$\|AME - SV\|_\infty \leq 2((1+\frac{1}{\varepsilon})^{2\varepsilon} - 1).$$

In addition, under a monotone utility,

$$\|AME - SV\|_2 \leq ((1+\frac{1}{\varepsilon})^{2\varepsilon} - 1) + \sqrt{2((1+\frac{1}{\varepsilon})^{2\varepsilon} - 1)}.$$

Proof. With a small abuse of notation, we write $P_{\mathrm{AME}}(j) = \sum_{S:|S|=j} P_{\mathrm{AME}}(S)$ and $P_{\mathrm{SV}}(j) = \sum_{S:|S|=j} P_{\mathrm{SV}}(S)$ (see §A.3 for detailed definition of $P_{\mathrm{AME}}(S)$ and $P_{\mathrm{SV}}(S)$). Notice that now $P_{\mathrm{AME}}(j)$ is the PMF of a beta binomial distribution

$\mathrm{BB}(\alpha = 1+\varepsilon, \beta = 1+\varepsilon, n = N-1)$. Let $\mathrm{B}(\cdot,\cdot)$ denote the Beta function, then:

$$\begin{aligned}
\Delta + 1 = \max_j P_{\mathrm{AME}}(j)/P_{\mathrm{SV}}(j) &= \max_j N\binom{N-1}{j}\frac{\mathrm{B}(j+\alpha, N-1-j+\beta)}{\mathrm{B}(\alpha,\beta)} \\
&= \max_j N \cdot \frac{1}{N\mathrm{B}(N-j, j+1)} \cdot \frac{\mathrm{B}(j+\alpha, N-1-j+\beta)}{\mathrm{B}(\alpha,\beta)} \\
(\text{since } \mathrm{B}(1,1)=1) &= \max_j \frac{\mathrm{B}(j+1+\varepsilon, N-j+\varepsilon)}{\mathrm{B}(j+1, N-j)} \cdot \frac{\mathrm{B}(1,1)}{\mathrm{B}(1+\varepsilon, 1+\varepsilon)} \\
(\text{by lemma A.10}) &\le \max_j \frac{(j+1+\varepsilon)^{\varepsilon}(N-j+\varepsilon)^{\varepsilon}}{(N+1+2\varepsilon)^{2\varepsilon}} \Big/ \frac{\varepsilon^{2\varepsilon}}{(2+2\varepsilon)^{2\varepsilon}} \\
(\text{maximum reached when } j+1+\varepsilon = N-j+\varepsilon) &\le \frac{1}{4^{\varepsilon}} \frac{(2+2\varepsilon)^{2\varepsilon}}{\varepsilon^{2\varepsilon}} \\
&\le (1+\frac{1}{\varepsilon})^{2\varepsilon},
\end{aligned} \tag{A.8}$$

Applying Lemma A.6 concludes the proof. □

Now we formally state the $O(k \log N)$ sample rate:

Corollary A.12. *For every $\varepsilon > 0, \delta > 0, N \ge 3$, there exists constants $C_3(\varepsilon,\delta)$ and ε', and a regularization parameter λ, such that when the number of samples $M \ge C_3(\varepsilon,\delta) k \log N$, with a probability at least $1-\delta$,*

(1) $\|\sqrt{v}\hat{\beta}_{lasso} - SV\|_\infty \le \varepsilon$ holds when the utility is bounded in $[0,1]$ and AME is k-sparse;

(2) $\|\sqrt{v}\hat{\beta}_{lasso} - SV\|_2 \le \varepsilon$ holds when the utility is monotone and SV is k-sparse,

where AME is defined by $\mathcal{P} = \mathrm{Beta}(1+\varepsilon', 1+\varepsilon')$ and $v = \mathbb{E}_{p\sim\mathcal{P}}[\frac{1}{p(1-p)}]$.

Proof. Observing that $\|\sqrt{v}\hat{\beta}_{lasso} - \mathrm{SV}\|_\infty \le \|\sqrt{v}\hat{\beta}_{lasso} - \mathrm{AME}\|_2 + \|\sqrt{v}\mathrm{AME} - \mathrm{SV}\|_\infty$ and $\|\sqrt{v}\hat{\beta}_{lasso} - \mathrm{SV}\|_2 \le \|\sqrt{v}\hat{\beta}_{lasso} - \mathrm{AME}\|_2 + \|\sqrt{v}\mathrm{AME} - \mathrm{SV}\|_2$, we proceed by bounding both $\|\sqrt{v}\hat{\beta}_{lasso} - \mathrm{AME}\|_2$ and $\|\sqrt{v}\mathrm{AME} - \mathrm{SV}\|_\infty$ (or $\|\sqrt{v}\mathrm{AME} - \mathrm{SV}\|_2$) by $\varepsilon/2$.

Prop. A.5 directly yields the bound on the first term. In both cases (1) and (2), assumptions are verified as follows. First, a k-sparse AME is assumed in Case (1) and implied in Case (2) by a monotone utility plus a k-sparse SV (details in Corollary A.7). Second, the X_ns are bounded, since

$\forall n \in [N] : X_n \in [-\sqrt{v}, \sqrt{v}]$ and v is finite:

$$v = \int_0^1 \frac{1}{p(1-p)} \frac{p^\varepsilon (1-p)^\varepsilon}{\mathrm{B}(1+\varepsilon, 1+\varepsilon)} dp = \frac{\mathrm{B}(\varepsilon, \varepsilon)}{\mathrm{B}(1+\varepsilon, 1+\varepsilon)} = 4 + \frac{2}{\varepsilon}, \tag{A.9}$$

where the last equality comes from the fact that $\Gamma(z+1) = z\Gamma(z)$.

To bound the second terms in both cases, notice that each version is given a bound in Lemma A.11, and that both approach 0 when $\varepsilon' \to 0$ and are positive when $\varepsilon' > 0$. In consequence, there exists $\varepsilon' > 0$ dependent only on ε such that both are $\leq \varepsilon/2$, concluding the proof. □

A.4 Efficient Sparse Beta-Shapley Estimator

Recall that Beta(α, β)-Shapley [110] is our AME defined on the distribution $\mathcal{P} = \mathrm{Beta}(\alpha, \beta)$. We show that our regression-based AME estimator with p-featurization (§A.2) can efficiently estimate Beta(α, β)-Shapley values when they are k-sparse, for all $\alpha > 1$ and $\beta > 1$, i.e., achieving low L_2 error with high probability using $O(k \log N)$ samples. Prop. A.5 directly yields the result. Its assumptions are verified given that:

$$v = \int_0^1 \frac{1}{p(1-p)} \frac{p^{\alpha-1}(1-p)^{\beta-1}}{\mathrm{B}(\alpha, \beta)} dp = \frac{\mathrm{B}(\alpha-1, \beta-1)}{\mathrm{B}(\alpha, \beta)} = \frac{(\alpha+\beta-2)(\alpha+\beta-1)}{(\alpha-1)(\beta-1)} \tag{A.10}$$

is finite (the last equality is due to $\Gamma(z+1) = z\Gamma(z)$) and $\forall n \in [N] : X_n \in [-\sqrt{v}, \sqrt{v}]$ is also bounded.

A.5 Extending to Approximate Sparsity

The above discussion assumes exactly k-sparse of the SVs, which in practice likely will not hold. In this section we extend our result to the case when it is only approximately sparse [152]. In such a setting, small non-zeros are allowed in the remaining $N-k$ entries of the SVs. Formally,

it requires that the best k-sparse approximation $\sigma_k(\text{SV}) = \inf_s\{\|\text{SV} - \mathbf{s}\|_1 : \mathbf{s} \text{ is } k\text{-sparse}\}$ is small.

A.5.1 The LASSO Error Bound under Approximate Sparsity

First we extend the LASSO error bound in Prop. A.1. This is relatively easy since Theorem 1.4 from [112] that Prop. A.1 has simplified supports approximate sparsity. We incorporate it by making two changes: a) β^* is now allowed to be an approximately sparse vector verifying $\sigma_k(\beta^*) \leq C_4(\delta)\|\xi\|_{L_q} k\sqrt{\frac{\log(N)}{M}}$, where $C_4(\delta)$ is a constant, $\xi = \langle \beta^*, X\rangle - Y$ and q is some constant > 2; b) the result is accordingly rewritten to the following: there exists a regularization parameter λ and a value $C_5(B-A,\delta)$ such that with probability at least $1-\delta$,

$$\|\hat{\beta}_{lasso} - \beta^*\|_2 \leq C_5(B-A,\delta)\|\xi\|_{L_q}\sqrt{\frac{k\log(N)}{M}}. \tag{A.11}$$

The proof is identical to that of Prop. A.1 thus omitted. Intuitively, the difference is that a k-sparse approximation with small enough error needs to exist to arrive to a similarly small enough error for the LASSO estimate. Another way to state this is that the sample size M in Eq. A.11 is upper bounded by the approximate sparsity requirement (the smaller error the best k-sparse approximation is, the larger M can be). Denote the upper bound by $M_{max} = \max\{M : \sigma_k(\beta^*) \leq C_4(\delta)\|\xi\|_{L_q} k\sqrt{\frac{\log(N)}{M}}\} = C_4(\delta)^2\|\xi\|_{L_q}^2/\sigma_k(\beta^*)^2 k^2\log(N)$. Approximate sparsity has two implications. First, given that the error bound $\varepsilon = \mathcal{E}(M) = C_5(B-A,\delta)\|\xi\|_{L_q}\sqrt{\frac{k\log(N)}{M}}$ decreases monotonically as M increases, the minimal error possibly achievable is lower bounded by a function of the "sparsity level" $\sigma_k(\beta^*)$:

$$\varepsilon \geq \mathcal{E}(\lfloor M_{max}\rfloor) \approx \frac{\sigma_k(\beta^*)}{\sqrt{k}} \cdot \frac{C_4(\delta)}{C_5(B-A,\delta)}. \tag{A.12}$$

We note that [92] shares a similar lower bound on ε. Second, recall that $M \geq k(1+\log(N/k)) = M_{min}$ is required, implying that the theorem is only applicable when $\lfloor M_{max}\rfloor \geq M_{min}$. Because

M_{max} increases with lower error sparse approximations, this is equivalent to require that:

$$\sigma_k(\beta^*) \leq C_4(\delta)\|\xi\|_{L_q} k\sqrt{\frac{\log(N)}{\lceil M_{min}\rceil}} \approx C_4(\delta)\|\xi\|_{L_q}\sqrt{\frac{k\log(N)}{1+\log(N/k)}}. \tag{A.13}$$

Though this appears to be an extra requirement compared to [92], our empirical results suggest that it is not limiting in the cases we studied. Indeed, this only rules out sample sizes $\leq k(1+log(N/k))$, smaller than the M needed for good performance across our evaluations.

We now restate the result as a form of (ε, δ)-approximation to make the result more approachable.

Corollary A.13. *For every sufficiently sparse β^* s.t. Equation A.13, and every $\delta > 0, \varepsilon \geq \mathcal{E}(\lfloor M_{max}\rfloor)$, $N \geq 3$, there exists some constant $C_7(B-A,\varepsilon,\delta)$ such that when the sample size $M = \max(\lceil M_{min}\rceil, \min(\lfloor M_{max}\rfloor, \lceil C_7(B-A,\varepsilon,\delta)k\log N\rceil))$, with a probability at least $1-\delta$, $\|\hat{\beta}_{lasso} - \beta^*\|_2 \leq \varepsilon$.*

Proof. Let $\mathcal{E}(M) \leq \varepsilon$, we have $M \geq C_5(B-A,\delta)^2\|\xi\|_{L_q}^2 k\log N/\varepsilon^2$. Because of $\|\xi\|_{L_q}^q \leq e$ (a fact proved in the proof of Prop. A.1), we can further simplify it to $M \geq C_5(B-A,\delta)^2 ek\log N/\varepsilon^2$. Let $C_7(B-A,\varepsilon,\delta) = C_5(B-A,\delta)^2 e/\varepsilon^2$. Next, by clipping it with $\lfloor M_{max}\rfloor$, $\sigma_k(\beta^*) \leq C_4(\delta)\|\xi\|_{L_q} k\sqrt{\frac{\log(N)}{M}}$ is verified and $\varepsilon \geq \mathcal{E}(M)$ still holds due to $\varepsilon \geq \mathcal{E}(\lfloor M_{max}\rfloor)$; Equation A.13 further ensures that there exists at least a choice of M between $\lceil M_{min}\rceil$ and $\lfloor M_{max}\rfloor$. Finally, by further clipping with $\lceil M_{min}\rceil$, all preconditions are then satisfied and applying Equation A.11 concludes the proof. □

A.5.2 Extending the SV estimators

We first derive an L_2 error bound assuming monotone utility, and later discuss an L_∞ error bound when utility is not monotone.

As a reminder, we chose a distribution $\mathcal{P}$ such that the AME under $\mathcal{P}$ is close enough to the SV, and then apply LASSO to estimate the AME with a low error. The LASSO error bound requires

the sparsity of β^*, which utilizes the sparsity of the AME, which is derived from the sparsity of the SV. When SV is instead approximately sparse, we can still derive an approximate sparsity guarantee for the AME and consequently for β^*.

Lemma A.14. *When the utility is bounded in* $[0,1]$ *and monotone,* $\sigma_k(\beta^*) \le \sqrt{2}\sigma_k(AME) \le 3\sqrt{2}\sigma_k(SV)$ *holds for both truncated uniform* $Uni(\varepsilon', 1-\varepsilon')$ *and* $Beta(1+\varepsilon', 1+\varepsilon')$ *with* $\varepsilon' \in (0, 0.5)$.

Proof. Recall that $\beta^* = \text{AME}/\sqrt{v}$, we have $\sigma_k(\beta^*) = \frac{\sigma_k(\text{AME})}{\sqrt{v}} \le \sqrt{2}\sigma_k(\text{AME})$, where the inequality is due to $v \ge \frac{1}{2}$ for $Uni(\varepsilon', 1-\varepsilon')$ (see Corollary A.8) and $v \ge 4$ for $Beta(1+\varepsilon', 1+\varepsilon')$ (see (A.9)).

Next we connect $\sigma_k(\text{AME})$ and $\sigma_k(\text{SV})$. Monotonicity ensures that $\text{AME}_n > 0$ and $\text{SV}_n > 0, \forall n$. By (A.4), either $\text{AME}_n \le \text{SV}_n$ or $\text{AME}_n \le \text{SV}_n(1+\Delta)$ holds. Thus $\sigma_k(\text{AME}) \le (1+\Delta)\sigma_k(\text{SV})$. Further applying (A.5a) for $Uni(\varepsilon', 1-\varepsilon')$ and (A.8) for $Beta(1+\varepsilon', 1+\varepsilon')$ concludes the proof. □

With a similar application of the LASSO error bound as previously done in e.g., Corollary 2.8, we arrive at a similar (ε, δ)-approximation:

Corollary A.15. *For every constant* $\varepsilon > 0, \delta > 0, N \ge 3$, *there exists constants* $q > 2, \varepsilon'$, *a LASSO regularization parameter* λ, *and an* $M = O(k \log N)$, *such that with probability at least* $1-\delta$*:*

(1) $\|\sqrt{v}\hat{\beta}_{lasso} - SV\|_\infty \le \varepsilon$ *holds when the utility is bounded in* $[0,1]$ *and* $\sigma_k(AME) \le \frac{1}{\sqrt{2}} C_4(\delta) \|\xi\|_{L_q} k \sqrt{\frac{\log(N)}{\lceil M_{min} \rceil}} \approx \frac{1}{\sqrt{2}} C_4(\delta) \|\xi\|_{L_q} \sqrt{\frac{k \log(N)}{1+\log(N/k)}}$ *and* $\varepsilon \ge \mathcal{E}(\lfloor M_{max} \rfloor) \approx O(\sigma_k(AME)/\sqrt{k})$*;*

(2) $\|\sqrt{v}\hat{\beta}_{lasso} - SV\|_2 \le \varepsilon$ *holds when the utility is monotone and* $\sigma_k(SV) \le \frac{1}{3\sqrt{2}} C_4(\delta) \|\xi\|_{L_q} k \sqrt{\frac{\log(N)}{\lceil M_{min} \rceil}} \approx \frac{1}{3\sqrt{2}} C_4(\delta) \|\xi\|_{L_q} \sqrt{\frac{k \log(N)}{1+\log(N/k)}}$ *and* $\varepsilon \ge \mathcal{E}(\lfloor M_{max} \rfloor) \approx O(\sigma_k(SV)/\sqrt{k})$,

where the noise ξ *is defined as* $\langle \beta^*, X \rangle - Y$, *the AME is defined under* $\mathcal{P} = Uni(\varepsilon', 1-\varepsilon')$ *and* $v = \mathbb{E}_{p \sim \mathcal{P}}[\frac{1}{p(1-p)}]$.

The result of Beta distribution is similar.

A.6 Evaluation Details

A.6.1 Warm-starting Optimization and Hyperparameter Tuning

Given the high cost of training deep learning models, we support an optimization that uses warm-starting as a proxy for full model training. Specifically, instead of training each submodel from scratch, we fine-tune the main model on each subset S for a fixed number of iterations, usually the number of iterations in one main model training epoch. Although this results in a more noisy estimate of $U(S)$, our estimator is able to handle the noise, yielding an overall speedup in wall clock time.

With warm-starting, instead of learning a model from the subset S, we "unlearn" the signal from the points not in S. This has two implications on our choice of $\mathcal{P}$. First, changing the outcome of a given query usually requires removing all its contributors, as even a small number of them is sufficient to maintain the signal learned in the main model. Hence, we only consider lower inclusion probabilities ($p \leq 0.5$). Second, warm-starting does not collapse model even on very small data subsets, as opposed to learning the model from scratch. We can thus consider smaller values of p, and settle on the range $\mathcal{P} = Uni\{0.01, 0.1, 0.2, 0.3, 0.4, 0.5\}$.

Hyperparameters of model training for warm-starting. Warm-start training requires specifying the hyper-parameters (e.g., batch size, learning rate, training time, etc.). If the original learning rate changed adaptively during the course of training, e.g., when using a training approach such as Adam [99], we use the learning rate and batch size for the final epoch and run fine-tuning for one epoch on the data subset.

Otherwise we fine-tune every subset model for the same fixed number of iterations, and vary batch size proportionally to the number of training examples included, such that every datapoint is iterated through roughly only once (i.e., one epoch on the data subset). Moreover we observe that when batch size is below a certain number the models soon all collapse. Therefore, we

lower bound the batch size by that number, which is 100 for both CIFAR10 datasets and 20 for the ImageNet dataset. The reason of such a co-design is the following. To make the numbers comparable, the subset models should be all fine-tuned with the same number of steps. If we still use a constant batch size, every datapoint will be visited different number of times across different data subset sizes, making the impact of one source to be less comparable. Thus we decide to vary the batch size accordingly such that each datapoint is visited roughly once. In addition, we choose the learning rate such that the validation accuracy of most subset models drops by roughly 20%, a not-too-large but still significant number. The reason is that the learning rate should be large enough that when no or few poison is included, the subset model should have the poisoning mostly erased, and vice versa.

A.6.2 Datasets and Attacks

We provide more context and details on the datasets, models, and attacks from Table 2.1.

Datasets. We evaluate our approach using the following four data sets and inference tasks:

- *CIFAR10*, an image classification task with ten classes [106]. We consider each individual training sample as a single source, and use the `ResNet9` model and training procedure described in [6]. For one of the attacks (Poison Frogs [155]), we use `VGG-11` instead of `ResNet9`, and use transfer learning to specialize a model trained on the full CIFAR10 data using the training procedure described in [108]. Transfer learning is used to specialize the model for 10% of the CIFAR10 training data. We only re-train the last layer and freeze every other layers.

- *EMNIST*, a hand-written digit classification task with examples from thousands of users each with their own writing style [35]. Each user is a data source with multiple examples. We use models and training procedures from [150].

- *ImageNet*, a one-thousand class image classification task [154]. The training data includes

more images than CIFAR10 (over 1 million vs 50000), and each image has a higher resolution (average of 482x418 pixels vs 32x32 pixels), thus increasing the training overhead. For ImageNet, we group training data into sources using the URL where the image was collected. Specifically, we treat each URL path (excluding the item name) as a source, and then combine all paths contributing fewer than 10 images into one source. This results in a dataset with 5025 sources. We use a `ResNet50` [78] model trained using the procedure from [151].

- *NLP*, a sentiment analysis task on 1 million book reviews written by 307k Amazon users [142]. Most users contribute fewer than 1000 reviews: we select and combine multiple such users at random when producing sources. This results in 1000 sources, 7 of which contain reviews from a single user, and the rest group random users into a single source. We use the model and training procedure described in [175] to produce a binary classifier that uses the review text to predict whether or not the review has a positive score (i.e., greater than 3).

Attacks. We evaluate our approach using three types of poisoning attacks. (1) *Trigger attacks* [29] which rely on a human-visible trigger to poison models. On image detection tasks, we use a 5x5 red square added to the top of each image or a watermark as our trigger. Column k in Table 2.1 lists the number of poisoned sources. For the NLP task we use a neutral sentence as a trigger. (2) A *label-flipping attack* on EMNIST, where a poison source copies all the data from a benign user (in our case user 171) and associates a single label (in our case `6`) for all of this copied data. (3) The *Poison Frogs attack* [155] on CIFAR10, which is a clean-label attack that introduces imperceptible changes to training images that will poison a target model in a transfer learning setup, where the last few layers of an existing pre-trained model are refined using additional training data to improve inference performance. Figure 2.3 shows examples of trigger and poison frog attacks.

Hyperparameters. We evaluate the impact of hyperparameters using micro-benchmarks in §A.6.3. Unless otherwise stated, we use $q = 0$ for knockoffs. As we explain in §A.6.3, this is a con-

servative choice that seeks to minimize the false discovery rate. We also use $\mathcal{P} = Uni\{0.2, 0.4, 0.6, 0.8\}$ for training-from-scratch and $\mathcal{P} = Uni\{0.01, 0.1, 0.2, 0.3, 0.4, 0.5\}$ for warm-start training.

We use Glmnet [8, 75] for the LASSO implementation. For the LASSO regularization parameter λ, a correct choice is required by our error bound (Prop. 2.4), which unfortunately we have no information on. Bypassing this obstacle remains as an interesting future work. Indeed this has been studied in [113] and other slightly weaker LASSO error bounds (e.g., Theorem 3.5.1. from [145]) but with known λ exist. In practice we choose the λ using a common empirical procedure [75] that runs 20-fold cross-validation (CV) and chooses the largest λ (sparsest model) with errors within one standard deviation of the best CV error, which we denote as λ_{1se}. For the SV estimation, we use λ_{min} that gives the best CV error, as we are not seeking a sparsest model here.

A.6.3 Ablation Study for Different Parts of the Methodology and Parameters

Next we use microbenchmarks to evaluate the effect that different hyperparameters have on our methods. Then, we show a discussion of hyperparameters for warm starting, the benefits of using Knockoffs, and a comparison between LASSO and diff-in-means, a straight-forward AME estimator that uses empirical means to estimate expectations. Finally, we discuss the tradeoff between using training-from-scratch and warm-starting.

A.6.3.1 Effect of Hyperparameters

Effect of the Target FDR Level q. All results presented thus far were with $q = 0$. Here, we vary q, which allows us to trade-off precision to achieve higher recall. For this microbenchmark we use λ_{min}, which we define as the λ with the best cross-validation error, instead of λ_{1se} which we use in the rest of our evaluation. This is because the precision provided by LASSO imposes a lower-bound on the precision achieved using Knockoffs', and in our experiments we found that

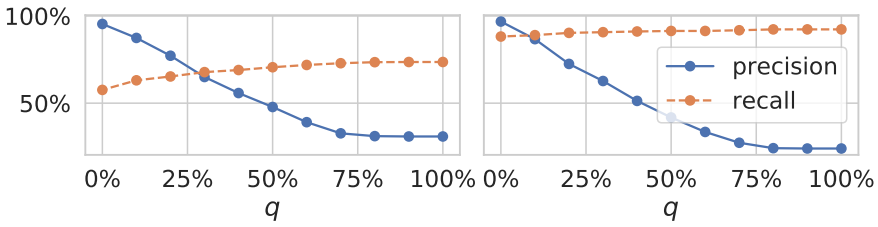
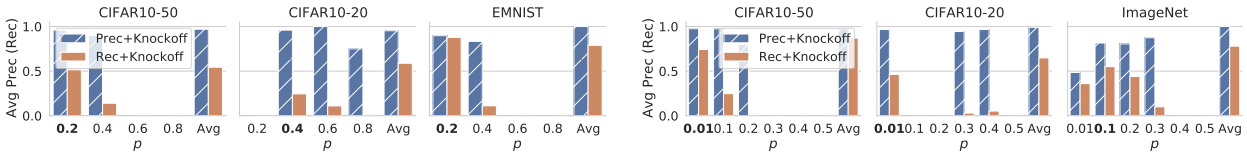


Figure A.1: Effect of q in Knockoffs on CIFAR10-50. Left: training-from-scratch; Right: warm-start training.



(a) Training-from-scratch **(b)** Warm-starting

Figure A.2: LASSO results run only on observations from a single p value, we highlight the p value providing the best result. Avg is the result from using the entire p value grid.

LASSO alone can achieve high-precision when λ_{1se} is used, making it harder to observe effects at lower values of q. Figure A.1 shows the results of varying q in the CIFAR10-50 setting. We can see that while increasing q does lead to a small increase in recall, it comes at significant cost to precision.

Effect of p. Next we address the question of how values of p affect our method. Figure A.2 shows our metrics on subset models drawn with one single value of p. We can see that no single value of p suffices across datasets and training algorithms. For instance, training-from-scratch $p = 0.2$ works well for CIFAR10-50 and EMNIST, but not for CIFAR10-20. This difference between CIFAR10-50 and CIFAR10-20 is likely due to the power of the attack: both use the same attack, but CIFAR10-50 uses a larger number of poisoned sources. This means that smaller values of p are more likely to select poisoned sources in CIFAR10-50, explaining our observations. For warm starting, we confirm that small values of p ($p = 0.01$) perform better than larger ones. These results thus show that (a) no single value of p suffices for all models, thus motivating our use of a grid, and (b) the grid $\mathcal{P}$ can be tuned when a prior is available.

Effect of adding less informative ps. One might wonder how including sub-optimal p values

Query	Prec_0.4	Rec_0.4	Prec_0.6	Rec_0.6
#3	**90.9**	**50.0**	nan	0.0
#7	**91.7**	**55.0**	nan	0.0
#9	100.0	10.0	**100.0**	**20.0**
#10	100.0	15.0	**100.0**	**25.0**
#14	**100.0**	**40.0**	nan	0.0
#18	75.0	15.0	**100.0**	**20.0**

Table A.1: Single-p LASSO results for $p = 0.4$ and $p = 0.6$ on selected queries, CIFAR10-20 training-from-scratch. $p = 0.4$ has better average results (see Fig. A.2), but $p = 0.6$ outperforms for some queries.

impacts our results? To answer this question, in Figure A.2, we compare the single p results to that of using all ps in the grid (shown as avg). We find that while very uninformative ps can hurt our results (e.g., with EMNIST), this is rare and the effect is small. On the other hand, in many cases the grid result is better than the result from just using the single best p.

Effect of different ps on different queries. As discussed above, no single value of p suffices across all datasets. Surprisingly, as shown in Table A.1, we found that even within the same dataset, the best p can differ *across queries*.

Comparison against fixing $p = 0.5$. We also compare with a simple distribution where each subset is sampled with equal probability. This corresponds to using fixed $p = 0.5$. The result is much worse than our default sampling over a grid as shown in Table A.2.

	precision	recall	c
CIFAR10-50-tfs	100	3.2	8
CIFAR10-50-ws	100	1.8	12
EMNIST	100	7.8	16

Table A.2: Experiment results of using fixed $p = 0.5$. "tfs" denotes training-from-scratch and "ws" denotes warm-starting.

A.6.3.2 Benefit of LASSO and Knockoffs

We evaluate the benefit of using LASSO over the more straight-forward diff-in-means that replaces the expectation with empirical mean in Eq. 2.1, and the power of Knockoffs in control-

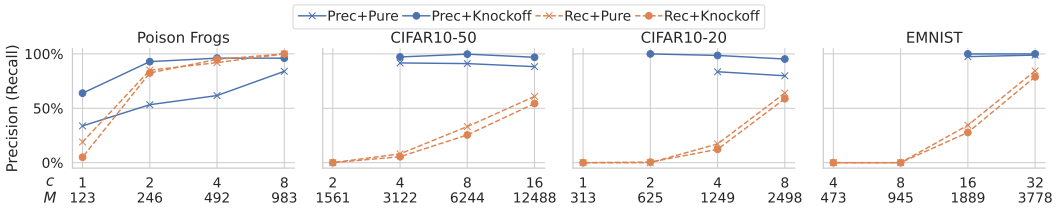
	Prec+ Pure	Rec+ Pure	Prec+ Diff	Rec+ Diff	c
Poison Frogs	84.0	100.0	**100.0**	100.0	8
CIFAR10-50-tfs	**88.3**	60.9	28.5	60.9	16
CIFAR10-20-tfs	**79.9**	64.0	76.7	64.0	8
EMNIST	**98.9**	84.4	62.9	84.4	16
NLP	**97.9**	98.2	5.6	98.2	24
CIFAR10-50-ws	**76.4**	89.6	41.6	89.6	24
CIFAR10-20-ws	**98.2**	66.0	9.2	66.0	48
ImageNet	**100.0**	77.0	94.2	77.0	12

Table A.3: Average precision and recall of *pure LASSO* and *diff-in-mean*. *tfs* and *ws* denote training-from-scratch and warm-starting respectively.

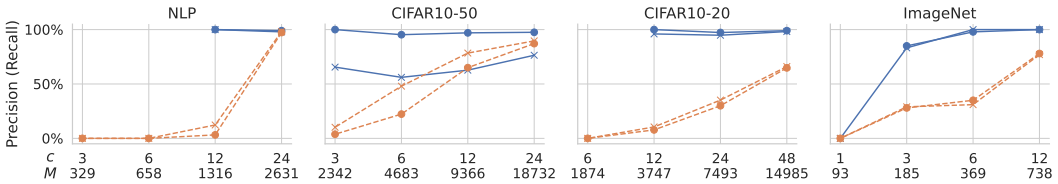
ling the FDR. First, to have a clean comparison against diff-in-means, we only run LASSO purely without adding the Knockoffs component. The selection is replaced with a procedure that selects all positive coefficients; For diff-in-means we select a threshold such that the recall matches that of pure LASSO for easy comparison. The result in Table A.3 shows that LASSO performs stably with valid precision while diff-in-means is fragile especially under high noise as in warm-starting. Next we compare this pure LASSO result against LASSO+Knockoff with varying number of observations in Figure A.3. We can see that LASSO+Knockoffs ensure that precision remains high even with a small number of samples. This is in contrast to LASSO which reduces precision when the number of samples decrease. Thus LASSO+Knockoffs allow our technique to be safely used even when an insufficient number of observations are available.

A.6.3.3 Training-from-scratch vs Warm-starting

Recall that warm-starting can greatly speed up the training procedure, but with a potential drawback of introducing higher noise for LASSO. In our evaluation, training CIFAR10-50 from scratch took 108.5 seconds, compared to 7.9 seconds when using warm-start training, representing a speedup of 13.8X. However, more observations might be required when using warm-start training. For example, in Table A.4 we find that for CIFAR10-20 the warm-start model achieves



(a) Training-from-scratch



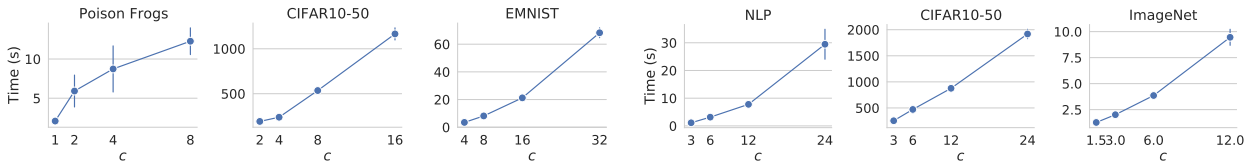
(b) Warm-starting

Figure A.3: Effect of growing c. "Prec+Pure" is the precision of LASSO without knockoffs; "Prec+Knockoff" is the precision of LASSO with knockoffs. "Rec+Pure" and "Rec+Knockoff" show the recall for both setups. Poison Frogs uses transfer learning and is not amenable to fine-tuning.

	Training-from-scratch		Warm-starting		c
	Prec	Rec	Prec	Rec	
CIFAR10-50	96.9	54.4	**97.0**	**77.6**	16
CIFAR10-20	**95.3**	**58.8**	100.0	0.8	8

Table A.4: Precision and recall comparison for training-from-scratch vs warm-starting under the same number of observations.

very low recall (0.8%), while training from scratch achieves reasonable recall (58.8%). We believe this is because of noise due to the number of poison sources required to trigger the attack. On the other hand, we see that CIFAR10-50 does not suffer from this problem, and in fact warm-start provides better results than training from scratch. This is because lower values of p work well to trigger the attack for CIFAR10-50, and warm starting over-samples this region. Adding more observations when using warm-start training with CIFAR10-20 improves this situation, and we find that when using $c = 48$ we can achieve a recall of 64.8%.



(a) Training-from-scratch **(b)** Warm-starting

Figure A.4: Average run times of LASSO on the queries as we grow c on training-from-scratch and warm-starting benchmarks. The error bar draws the standard deviation. CIFAR10-20 results are almost identical and thus omitted.

A.6.4 Runtime Evaluation

Run times of LASSO. The time to run LASSO grows linearly with the number of observations, as shown in Figure A.4. Together with Figure A.3 we can see that our technique can achieve good precision using a small number of observations, which translates to a small query time. Precision and recall improve with more observations, but this increases the time taken to execute a query.

Run times of inference. We report the total time of running ImageNet inference on a single RTX8000 GPU for a batch of 40 queries on 739 models, which corresponds to $c = 12$. We implement the inference of a model as three steps: allocating the model memory, loading the model from disk to GPU, and the actual GPU computation. They take 293.6, 447.5, and 584.9 seconds respectively for 739 models in total. The current throughput is 1.81 queries per minute, and the latency is 22.1 minutes. We note that both numbers could be further optimized (e.g., the model memory allocation can be done only once and reuse for every model; the model loading and actual computation can be pipelined, etc.), and one can also batch more queries together to further improve throughput. In this paper we focus on improving precision and recall. We plan to focus on further performance optimizations in the future.

	Ours		Repr.	
	Prec	Rec	Prec	Rec
ImageNet	100	78.0	85.9	78.0
CIFAR10-50	96.9	54.4	99.9	54.4
CIFAR10-20	95.3	58.8	68.9	58.8
EMNIST	100	78.9	100	78.9

Table A.5: Comparison with Representer Points at the same recall level. Representer Points use best tuned λ assuming knowledge of ground truth.

A.6.5 Additional Evaluation of Comparison with Existing Works

A.6.5.1 Additional Evaluation of SCAn

SCAn [163] is a state-of-the-art poison defense technique designed to identify whether or not a given model is poisoned, and find the poisoned class when it is. It requires access to some clean data (10% by default). SCAn computes an AI (Anomaly Index) score for each class, and reports that a class is poisoned if its AI> 7.3891. In their "online setting", it can also detect if a query input is poisoned by clustering on features extracted from the last hidden layer's activation. An input is marked as poisoned if it belongs to a poisoned class and is clustered in the group with fewer clean data. We modified SCAn to run on a training set to detect poisoned training data.

Modification of SCAn. Originally in the online setting, SCAn is designed to detect poison in every incoming test datapoint. It trains an untangling model on a clean set of data, and on receiving a test datapoint, it first fine-tunes the model and then use it to cluster with all the data it has to decide if the given test datapoint is poisoned. To modify it to detect poison data in the training set, instead of on the test data we train the untangling model on the training data. For efficiency, we assume the whole training set is available in the beginning, so that we only need to train the model once on the whole training set. This gives advantage to SCAn because originally in the beginning the model is poorly fit and gives many false positives because of insufficient data. SCAn also only gives a coarse-grained (poisoned v.s. clean) outcome to each datapoint, preventing us from exploring its precision-recall tradeoff in evaluation. Thus we modify SCAn

to provide a score for each datapoint, using the distance to the hyper-plane ($v^T r$ in Eq 6 of the paper).

The result. Outlier detection approaches like SCAn target a somewhat different problem than data attribution. Outliers can represent not just attacks but also benign but rare data, and thus outlier-detection cannot differentiate between attacks and benign data, nor between different attacks on the same data. We evaluate this effect using a mixed attack setup on CIFAR10, where we simultaneously use 3 different attacks, each of which poisons 20 images. The attacks used are trigger attacks (similar to CIFAR10-20), except each attack places the trigger in a different location. We found that SCAn clusters nearly all poisoned images (along with many clean images), regardless of the attack used, into the same group, showing that it cannot differentiate between attacks. In contrast, when using our approach with the same hyperparameters as CIFAR10-20 from warm-starting, ours can select the correct attack for each query, and achieves an average precision (recall) of 96.3% (65.5%), 97.4 (89.5%) and 97.1% (71.3%), respectively for 20 random queries from each attack.

Furthermore, even when a single type of attack is used, SCAn cannot detect all of the poisoned data. We evaluate this by using SCAn on CIFAR10-50, CIFAR10-20, EMNIST and ImageNet. To make SCAn work for EMNIST and ImageNet, which are source-based datasets, we adapt it to assign a score to a source equal to the number of selected datapoints within the source. The results in Fig. 2.8 shows that SCAn falsely selects many clean data for both CIFAR10 benchmarks, yielding very low precision. This is consistent with the results reported by SCAn authors, who state that at least 50 poison datapoints are needed for SCAn to work robustly. On the other hand, SCAn achieves perfect precision and recall on the other 2 benchmarks, since they are source-based datasets that contain hundreds of poison datapoints. Further, we found that SCAn's AI fails to identify the poisoned class for all datasets other than EMNIST. Thus SCAn cannot be used unless we are guaranteed to have a sufficiently large set of datapoints.

Additional evaluation for other variants of SCAn.

We also study if it helps SCAn to add more poison datapoints. We observe that SCAn starts to work well when we run SCAn on 104 and 300 poison datapoints in CIFAR10-50 and CIFAR10-20, respectively.[2] The full result is shown in Table A.6, where the we show both the precision and recall of the original SCAn and that of the variant after our distance modification. In the distance-based variant, we compute the precision and recall with a threshold that makes the recall match that of ours as closely as possible. This leads to an ad-hoc solution to this insufficient data problem: because query inputs are likely to be poisoned data, the operator will wait for an enough number of them to supplement the dataset before running SCAn. However, it is unclear how many queries should s/he wait for since (a) the number of poison data needed can vary across attacks and we do not know it in advance, and (b) even if we know it, we cannot know if this query is an attack. The problem is even worse when multiple attacks coexist, in which case the queries may spread across different attacks/classes, increasing the time needed for each attack/classes to have accumulated sufficient poisons.

On the mixed attack, with more poison datapoints SCAn still clusters the three attacks together, though it stops falsely clustering clean data into the same group as poisoned data.

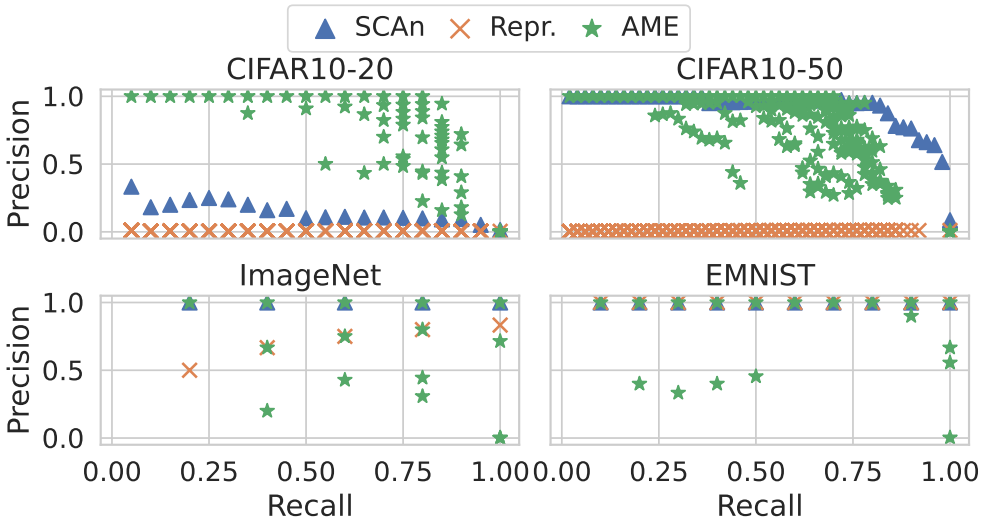


Figure A.5: The counterpart of Fig. 2.8 with AME using the regularization parameter choice of λ_{min}.

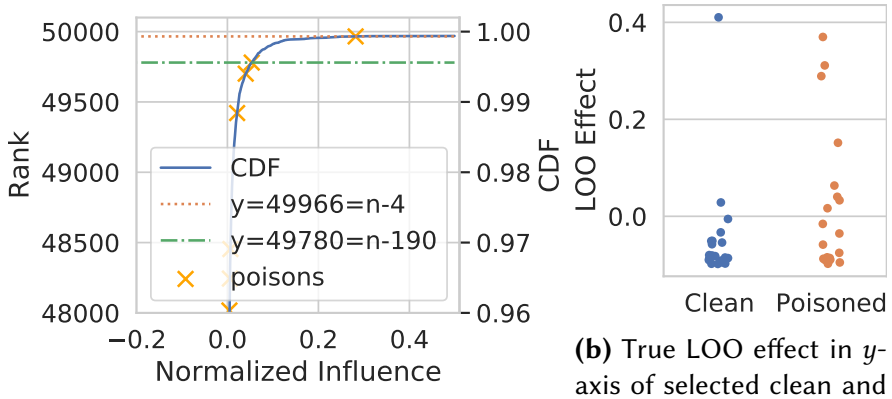
[2]Note that the main model is still trained on the original dataset without additional poison datapoints added.

A.6.5.2 Additional Evaluation of Rrepresenter Point

Representer Points [205] provide an approach to quantify the contribution each training datapoint has on a given query prediction. Given a query t, this approach assigns a score $\alpha_{i,Q_L} f_i^T f_t$ to each training datapoint i, where f_i is the feature extracted from the activation of the model's last hidden layer, f_t is the query input's feature, and α_{i,Q_L} is a scalar computed from the model's gradients. The query effect $f_i^T f_t$ is interpreted as the similarity between i and t, while the model effect α_{i,Q_L} represents i's importance to the model. There is a hyperparameter λ to trade off between the computation time and the result: smaller λs provide better results but are slower.

We compute a source-based score by summing up scores for all datapoints within a source, and pick the threshold such that the recall matches AME's recall as closely as possible. In our evaluation, we used $\lambda = 3e-3$; Figure 2.8 shows the results. While this approach performs better than ours on EMNIST, it is worse on ImageNet and does not work on either of the CIFAR10 datasets. We note that λ is crucial to the result, but the paper's suggestion of using a small λ like $3e-3$ does not always lead to good results because it tends to weight model effects over query effects. The paper does not document this problem, so it is unclear how λ can be tuned to avoid the problem. Indeed, finding a good λ requires prior knowledge of what sources are poisoned, as we show next.

We discover that λ has a potential effect crucial to result: it balances between model effect v.s. query effect, which is undocumented in the paper. The default λ gives bad result, so we manually tune it per attack with a grid search assuming knowing the ground truth and report the best result, as shown in Table A.5. It works better than ours on EMNIST and CIFAR10-50 but worse on ImageNet and CIFAR10-20. We note that the best λ can vary across datasets: as shown in Table A.7 the λ that works the best on CIFAR10-50 can be 30, which gives poor result on CIFAR10-20. Therefore, it is unclear how to find the right λ that works against unknown attacks even on the same dataset.



(a) Part of CDF of the influence scores.

(b) True LOO effect in y-axis of selected clean and poisoned datapoints.

Figure A.6: Experiment of the influence function on CIFAR10-20. The loss function is defined as the average loss on 20 query inputs. We use the parameter configuration of $r = 10, t = 1000$, damping term $\lambda = 0.01$, and scale term=25.

To investigate why the default λ falls apart, we look at what are selected and find that regardless of the query, it tends to select many the same images: the selection results between two random clean queries of the same predicted label often share $> 70\%$ of selections. This is likely because small λ tends to overweight the model effect.

A.6.5.3 Additional Evaluation of Influence Functions

Influence functions provide an approach to estimating the leave-one-out (LOO) effect, defined as $L_{excluded} - L_{original}$ where L denote the model's loss. We evaluated their efficacy for identifying poison datapoint by computing the influence function for each training point in the CIFAR10-20 setup. We expect proponent points will have higher influence scores, and thus poisoned data will have a higher influence score. In Figure A.6(a) we show a CDF of influence scores for each training datapoint, as well as their rank by influence value. We observe that only 2 (out of 20) of the poisoned data points show up in the top 190 elements. We can thus see that influence functions do not suffice for detecting poisoned datapoints in this case.

We next examine the efficacy of directly using leave-out-one training to detect poisoning. We do so by training 39 models where we leave out the 10 datapoints with the highest influence

scores, the 10 datapoints with the lowest influence scores, and any of the 20 poison datapoints not included in these sets. In Figure A.6(b) we show the LOO effect for all of these models. We can observe from this that many poisoned datapoints have LOO effects that are nearly as low as those observed for correct datapoints. This shows that a more precise technique for estimating LOO would still be insufficient for detecting poisoned sources.

A.6.5.4 Additional Evaluation of Shapley Value

Due to prohibitive costs of computation for running existing Shapley value estimators on our large experiments, we instead provide an evaluation on a subset MNIST with data poisoning (see details of the dataset in Appendix A.6.8.2). Unlike previous experiments in which each query explains a single prediction on a test example, we now look at one query to explain the attack success rate on an entire poisoned test set. The utility measurement is hence much less noisy than in our main experiments (we expect the AME to be even better comparatively in a noisy setting, except maybe for Compressive Sensing). The baselines are KernelSHAP [102], Monte Carlo [65], Truncated Monte Carlo [65], and two sparsity-aware approaches "KernelSHAP (L1)" [102] and Compressive Sensing [92] (implementation details in Appendix A.6.8). AME uses the training-from-scratch setting with $\mathcal{P} = \{0.2, 0.4, 0.6, 0.8\}$. The sample sizes (i.e., number of utility evaluations) are all fixed to 1024, a number that is small for consistency with the large experiment setup, and still exceeds $N = 1000$ so that permutation (Monte Carlo) and non-regularized regression (KernelSHAP) based approaches are applicable.

We again compare the precision of each method at different recall levels, by varying internal decision thresholds. The result in Figure A.7 shows that sparsity-aware approaches achieve better performance than others. In particular, AME and "KernelSHAP (L1)" achieve the same performance and outperform other approaches. We also compare them for SV estimation in §2.5.3, and show that KernelSHAP (L1) is less suited in that case.

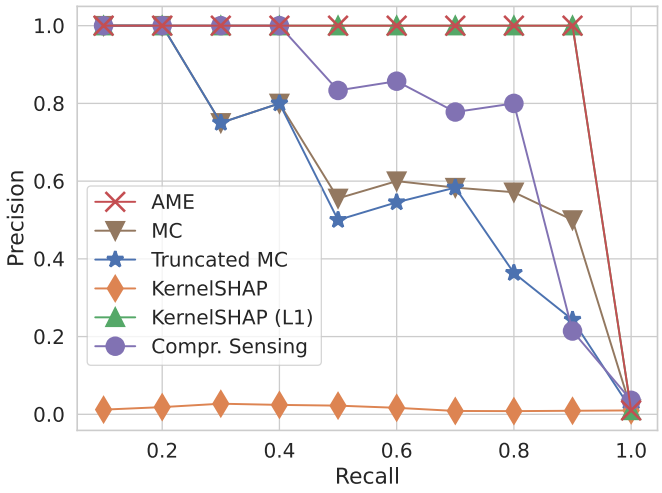


Figure A.7: Precision vs Recall curve comparison.

A.6.6 Details of the Hierarchical Design

Our hierarchical design is as follows: we partition users into groups based on when they first posted a review, which we treat as a proxy for when the user account was created. We group users into partitions of 1000 users each (so groups span different time periods: we found that fixed group sizes performed better than fixed time periods). Unlike the previous evaluation with the NLP data set (§A.6.2), in this experiment we do not combine tail users, i.e., users with a small number of review, into a single user. We use this process to partition the NLP dataset into $n_1 = 308$ top-level sources, each of which contains (up to) 1000 second-level sources. This partition covers all 307,159 users in the NLP dataset (i.e., $n_2 = 307,159$). We poison 15 second-level sources belonging to 2 randomly chosen top-level sources: putting 10 poisoned sources in one top-level source, and 5 in the other. We compare the results of applying our hierarchical approach in this setting, to the results of using the general approach on 307, 159 sources.

We use different ps when selecting top-level and second-level sources in the hierarchical approach, because top-level sources might contain multiple proponents and thus have a larger influence on inference results. In our experiment, we used $\mathcal{P} = Uni\{0.2, 0.4, 0.6, 0.8\}$) for the first-level

(p_1) and trained models from scratch, and we used $\mathcal{P} = Uni\{0.1, 0.2, 0.3, 0.4, 0.5\}$ for the second level (p_2) and used warm-start training.

Figure 2.5 shows the precision and recall achieved by the hierarchical approach on 20 different queries, as we vary the number of subset models. We find that (a) the hierarchical approach achieves good precision and recall in identifying top-level sources that contribute to data poisoning, even when fewer subset models are used than would allow second-level sources to be detected; and (b) recall when identifying second-level sources degrades gracefully as fewer observations are used. The hierarchical approach is also more efficient: it can achieve near-perfect precision and 60% recall with 1,588 observations. In comparison, the general method could not identify any of the poisoned sources when using the same number of observations.

Finally, the hierarchical approach also reduces the time spent running LASSO+Knockoffs, i.e., it reduces query runtime. To demonstrate this, we compare the running times for executing a query with 1,588 observations, using 5 threads on a server with 256GB of memory. The non-hierarchical approach takes 52.5s to construct the design matrix and 688.58s to run LASSO+Knockoffs. The hierarchical approach takes 37.343s to construct two design matrices (0.123s for the top level, and 37.220s for the second level) and 13.829s to run LASSO+Knockoffs (2.369s for the top level, 11.460s for the second level). This corresponds to an overall query time of 48.68s for the hierarchical approach versus 741.08s for the general approach, a reduction of 15×.

A.6.7 Details and Additional Evaluation of Data Attribution for Non-poisoned Predictions

We randomly select 10 classes and their images from the ImageNet dataset and use the same ResNet-9 model and training procedure. The main model achieves a top-1 accuracy of 89.2% on the validation set, from which we randomly select 40 with correct predictions and 40 with

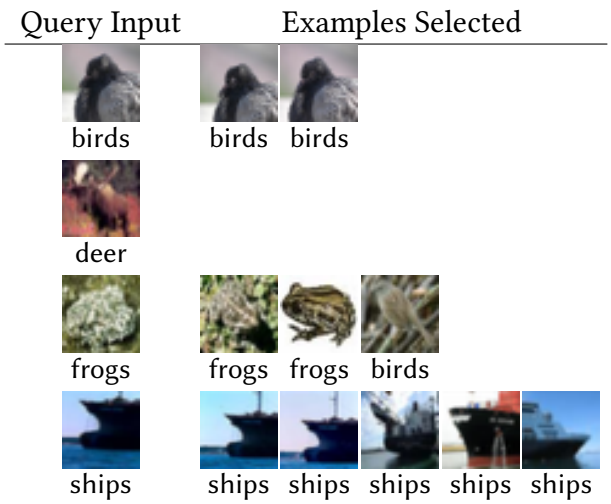


Figure A.8: Queries for correct predictions. The last row only shows a subset since they are too long to fit in.

incorrect predictions, as the queries. We train roughly 2700 subset models (calculated from $c = 10, k = 20, N \approx 10000$) from scratch. We switch to using λ_{min} (with $q = 0$) since empirically λ_{1se} is too conservative and selects few images, if any.

The results in Figure 2.6 and 2.7 (due to space constraints we show at most 3 selected images for each query) show that many selected images share similar visual characteristics with the query input, either in color, blur effect or texture. Some even are photos taken on the same place from different time/angle. These results suggest that our approach can be used to understand model behavior beyond the use case of data poisoning. Additional results for different queries, and a comparison to a naive baseline, can be found at `https://enola2022.github.io/`.

We also evaluate model explanation on CIFAR10-50 dataset. We reuse subset models from the CIFAR10-50 training-from-scratch experiments[3], and randomly select 39 images from the vanilla, non-poisoned test set as query inputs. Some queries with correct main model predictions (4 out of 39 in our evaluation) still fail because LASSO does not converge in 3600 seconds, possibly because no sparse solution exists. As a result we return an empty result set. Selected results are shown

[3]We did this to save time for model training. Though the training set includes poison, they should have little effect on the non-poisoned query inputs.

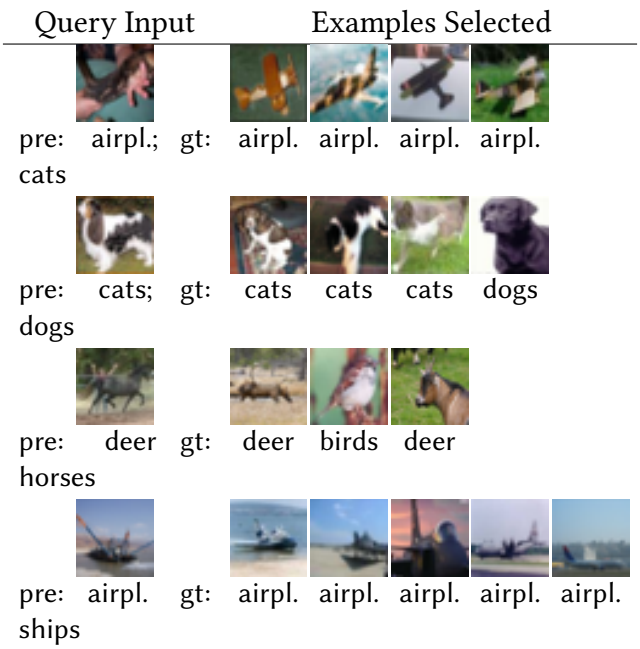


Figure A.9: Queries for wrong predictions. *Pre* is model's prediction and *gt* is the ground truth. The last row only shows a subset since they are too long to fit in.

in Figure A.8 and A.9, and we also draw all 39 queries we have ran and their result in grids, as shown in Fig. A.15 and Fig. A.16, where each row consists of two queries, each starting with the query input followed by images selected.

We also report a quantitative result on CIFAR10-50 by removing the images selected by ours and retraining the model 6 times and measure the change in the predictions. We find that the predicted labels are often not changed compared to the main model. This is expected because our approach is designed to achieve good precision and as a result only a small number of proponents from the strongest are removed, and other proponents in the class generically support the prediction. However, we still see the average confidence score is lower compare to the main model's when only removing the strongest proponents, as shown in Fig. A.10, while a baseline that randomly selects datapoints in the class to remove fails to do so.

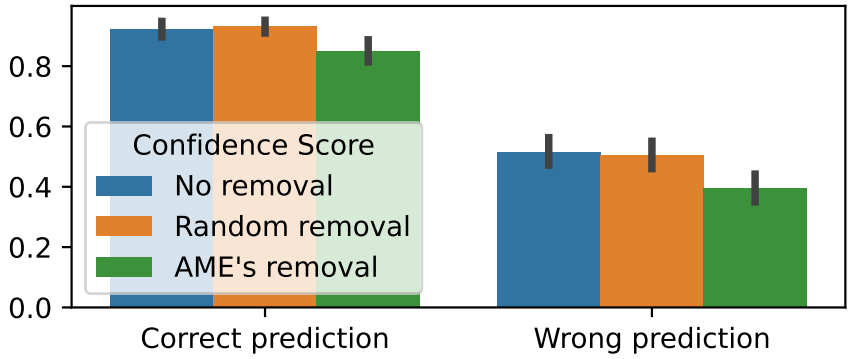


Figure A.10: Quantitative comparison of model explanation through the drop in the confidence scores. Y-axis shows the average confidence scores across all correct/wrong queries. The 95% confidence interval is drawn as the vertical bars.

A.6.8 Shapley Value Estimation Setting and Additional Evaluations

A.6.8.1 Simulated Dataset

Experimental Setup. We craft a two-valued threshold function as the utility, which evaluates to 1 when there are at least 2 in the first k sources are present. Formally, $U(S) = \mathbf{I}(|S \cap [k]| \geq 2)$. We pick $k = 3$ and $N = 1000$ for this experiment. This design allows us to compute the estimation error $\|\sqrt{v}\hat{\beta}_{lasso} - \text{SV}\|_2^2$ because the true value of Shapley values for the k sources can be easily known: by symmetry they are $U([N])/k = 1/k$ for the k sources and 0 otherwise.

We compared our AME based SV estimator to Monte Carlo [65, 92], Truncated Monte Carlo [65], Group Testing [92], Compressive Sensing [92], KernelSHAP [126], and Paired Sampling [37]. We run each approach 6 times with different random seeds. We plot the 95% confidence interval in the shaded area. The baseline is as follows:

- *Truncated Monte Carlo.* We adopt the implementation published by the author [65] and use the default hyperparameters (e.g., truncation tolerance is set to 0.01).
- *Group Testing.* We adopt the implementation provided by the author [91]. We use hyper-parameter $\epsilon = \frac{2}{\sqrt{N}}$ as in the script. We also tried $\epsilon = 0.01$ and 0.1 and the results remain the

same, thus omitted.

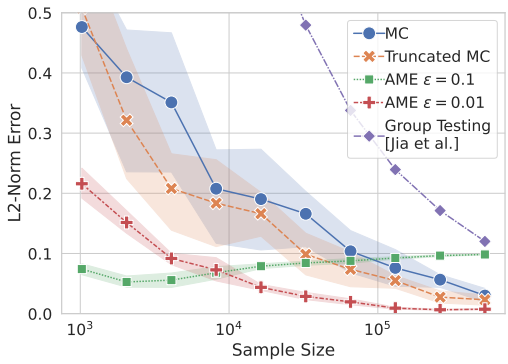
- *Compressive Sensing.* We implement the algorithm using CVXPY [2, 46]. The algorithm has a hyperparameter "M" that has no clear documentation for its choice. We therefore run one set of the 6-trial experiment for every $M \in \{2^7, 2^8, \ldots, 2^{16}\}$ and report for each sample size the best mean estimation error (the mean is taken on the 6 trials). This assesses its upper limit. We use $\epsilon = 0.01$ for the other hyperparameter "ϵ" when not specified. Unlike the original algorithm that shifts the SVs by the average utility $U([N])/N$ (see $\bar{s}$ in Algorithm 2 in their paper), we do not perform shifting because in a poisoning (thus sparse) setup most SVs are more likely to be zero rather than the average.

- *KernelSHAP.* We use their official implementation with the default hyperparameter setting. Note that by default KernelSHAP also uses L_1 regularization in a heuristic way. We also evaluate it with the regularization turned off to understand the effect.

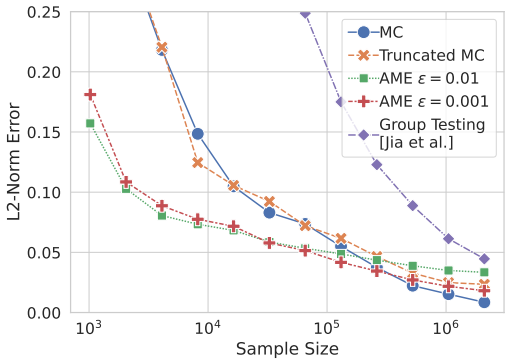
- *Paired Sampling.* We use their official implementation with batch size equal to 128, no convergence detection and non-stochastic cooperative game.

Additional evaluations and ablation study. In §2.5.3 in the main body we have shown a condensed result comparing against part of the baselines. Now we present the comparison against other baselines in Fig. A.11(a). High-level findings remain the same. We can also see that smaller ε leads to higher approximation precision at the cost of convergence speed. We also provide additional results of other choices of ϵ for Compressing Sensing in Figure A.12(a). It shows that $\epsilon = 0.1$ though has a marginal drop compared to $\epsilon = 0.01$ in estimation error when the sample size $\leq 2^{12}$ but with a (relatively big) sacrifice on the estimation error on large sample sizes, echoing a tradeoff role ε plays in AME (see §2.5.3), while $\epsilon = 0.001$ has almost identical result, suggesting a saturation on one end of the tradeoff space. This may also explain another interesting finding: the results are almost the same as Monte Carlo for the two smaller ϵ's. Recall that compressive sensing

utilizes ϵ to control the tradeoff between the sparsity and the accuracy (w.r.t. the measurements) of the recovered solution. Jia et al.'s design implicitly makes use of Monte Carlo for measurement, implying that the most accurate solution is Monte Carlo. The smaller the ϵ, the more accurate the recovery and hence the closer the result is to Monte Carlo.

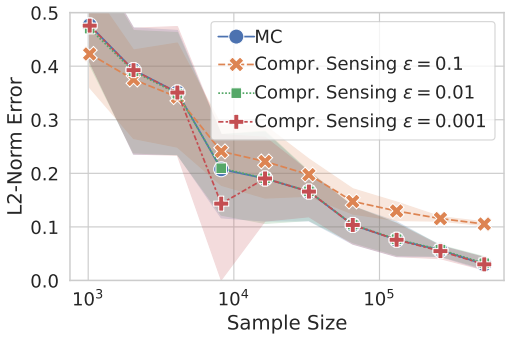


(a) Simulated dataset.

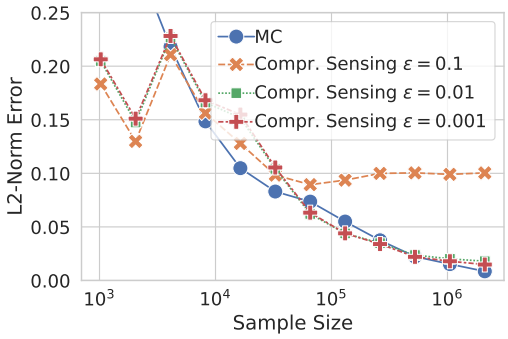


(b) MNIST dataset. We use as ground truth the Monte Carlo (MC)'s result on a sample size of 2^{22}.

Figure A.11: Additional comparison against other baselines on SV estimation.



(a) Simulated dataset.



(b) MNIST dataset.

Figure A.12: Compressive Sensing with different choices of ϵ.

We provide additional evaluations regarding the choice of featurization and distribution of our SV estimator from AME. In Fig. A.13 we compare the impact of featurization and fix the distribution to truncated uniform. It shows that they both work well and p-featurization performs slightly better (§A.2); In Fig. A.14 we compare beta distribution with truncated uniform $Uni(\varepsilon, 1-$

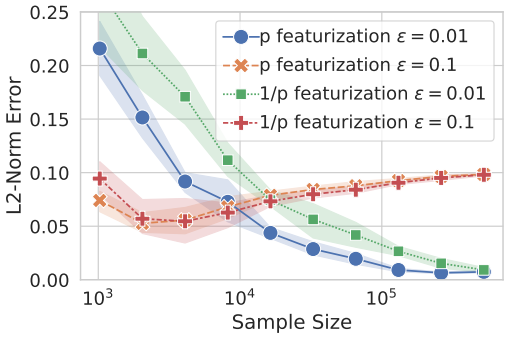


Figure A.13: p-featurization vs $1/p$-featurization (§A.2). Both use truncated uniform distribution $Uni(\varepsilon, 1-\varepsilon)$.

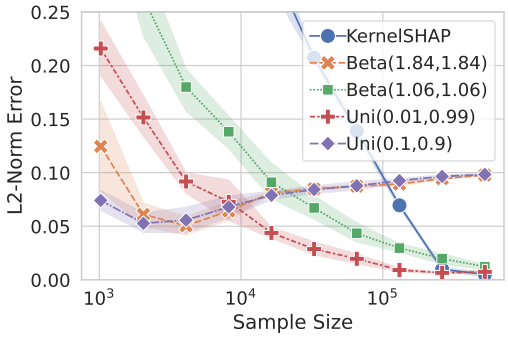


Figure A.14: Beta vs truncated uniform distribution. Our approaches (Beta* and Uni*) all use p-featurization (§A.2).

ε). The parameters for beta distribution are chosen to match that of truncated uniform such that their AMEs evaluate to the same value. In other words, they both converge to the same estimation error on an infinite sample size. It shows that they both perform well, but truncated uniform performs slightly better.

A.6.8.2 Poisoned MNIST dataset

Experimental Setup. For non-simulated case, given the prohibitive cost of computing the true SV, we craft a tiny MNIST dataset by randomly subsampling 1000 datapoints from both the original training set and test set. To create sparsity, we further poison 10 training datapoints randomly by imposing a white square on the top-left corner and overwriting their labels to 0. We also craft a poison test set by imposing the same trigger on the clean test set and use the attack success rate as the utility function. We use logistic regression (with regularizaiton 0.02) as the model for fast training. The resulting attack success rate of the full model is 65.7%, and the test accuracy is 89.4%. Different from the simulation case, it is computationally infeasible to compute the true SV. We instead use the estimation from the classic Monte Carlo method [65] as a proxy. We keep doubling the sample size until the estimation converges with a consecutive L_2 difference falling below 0.01, and use the final estimation as the proxy.

We compare the same set of approaches as in the simulated experiment (except Paired Sam-

Figure A.15: Correct-prediction queries and their results, where each row shows two queries and each query starts with the query input followed by images selected.

pling given the computational cost and its almost identical performance to KernelSHAP on the simulated dataset). The configurations for these approaches mostly remain the same, except we only run one trial to save computational cost.

Additional evaluations. Fig. A.11(b) shows the comparison against other baselines not shown in §2.5.3 in the main body, with Monte Carlo, Truncated Monte Carlo and Group Testing added. The qualitative findings hold.

A.7 Extended Related Work

The closest related works, **model explanation**, also aim to provide principled measures of training data impact on the performance of an ML model. These techniques include Shapley value, influence functions [102, 104] and Representer Point [205]. TracIn [149] and CosIn [73] are two other recent proposals for measuring the influence of a training sample based on how it

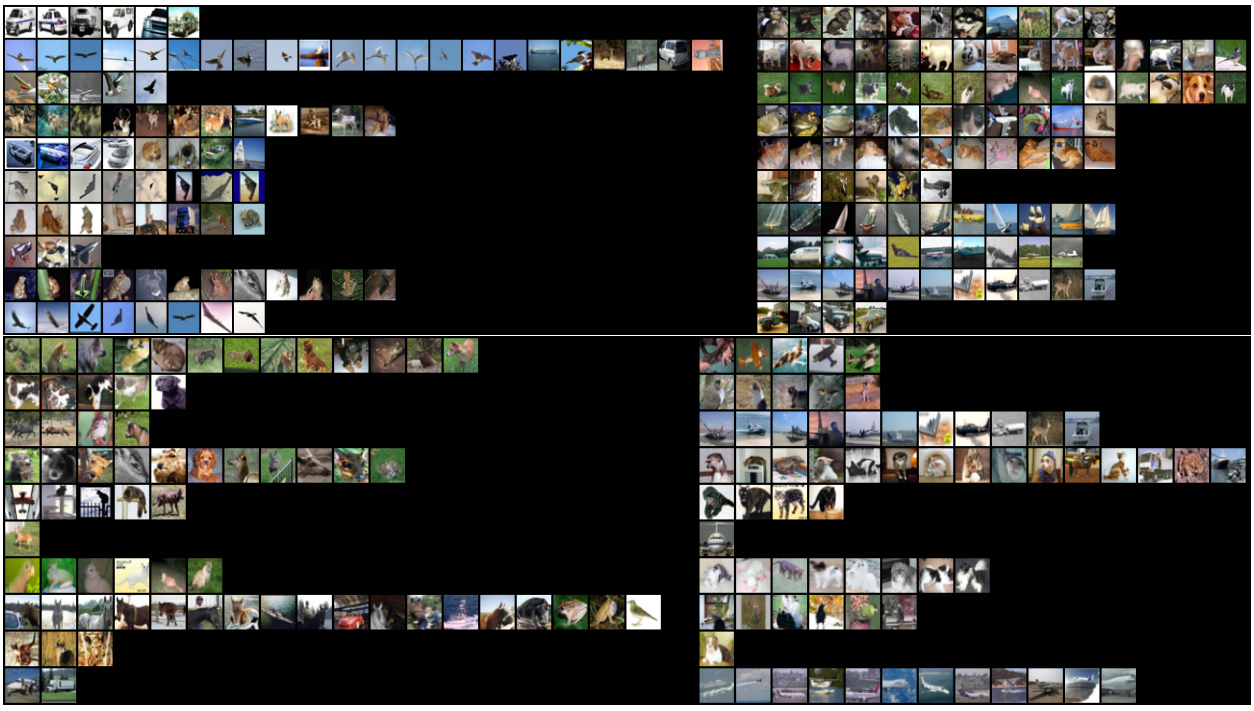


Figure A.16: Wrong-prediction queries and their results, where each row shows two queries and each query starts with the query input followed by images selected.

impacts the loss function during training. Existing model explanation approaches fall short. They either focus on marginal influence on the whole dataset [102, 104]; make strong assumptions (e.g., convex loss functions) that disallow their use with DNNs [11, 102, 104]; cannot reason about data sources or sets of training samples [23, 73, 149]; or subsample training data but focus on a single inclusion probability and thus cannot explain results in all scenarios[54, 87, 213].

Of those, the works focused on efficiently estimating the **Shapley value (SV)** are closest. SV which was proposed in game theory [157], has been widely studied in recent years, with applications to data valuation in ML [65, 92] and feature selection/understanding [37, 90, 123, 126], as well as extensions such as D-Shapley [64] which generalizes SV by modeling the dataset as a random variable. The SV is notoriously costly to measure, and efficient estimators are the focus of much recent work since the early permutation algorithm with L_2 error bounds in $O(N^2 \log(N))$ samples [131] under bounded utility. Recently proposed estimators also reduce SV estimation to regression problems, although different than ours [37, 90, 110, 126]. Beta-Shapley [110] is

closest, as it generalizes data Shapley to consider different weightings based on subset size, which coincides with our AME under p-induced distribution (e.g., Beta-Shapley is AME when $p \sim$ beta distribution). We also study this approach, and alternative distributions (including a truncated uniform which we found a bit better in practice). None of these works study the sparse setting, or provide efficient L_2 bounds for estimators in this setting. This may stem from their focus on explaining *features*, in smaller settings than we consider for training data and in which sparsity may be less natural. The most comparable work is that of Jia et.al [92], which provides multiple algorithms, including for the sparse, monotonous utility setting. Their approach uses compressive sensing, which is closely related to our LASSO based approach, and yields an $O(N \log\log N)$ rate. We significantly improve on this rate with an $O(k \log N)$ estimator, much more efficient in the sparse ($k \ll N$) regime. Other estimation strategies, use-cases, and relaxations have been recently proposed for the SV. [136] study efficient sampling of permutations. However, each "permutation sample" requires N utility evaluations, and is thus incompatible with our setting. [58] focus on feature SV's, and leverage the causal structure over features (on the data distribution) to decide value assignments. This is done by reweighing entire permutations of the features (e.g., giving more weight to permutations where a given feature appears early). In contrast, the AME reweighs the utility of different subsets of the data points, based on their size, which is orthogonal, and seems more amenable to sparse estimators. [74] extend the SV axioms to consider the joint effect of multiple players, prove the uniqueness of the solution, and derive its formulation. It measures the contribution of all subsets up to a specified subset size, which is however incompatible to our setup due to large N (e.g., there will be $O(N^2) \approx 2.5$ billion contribution terms just for subset size of two on our CIFAR-10 datasets). In contrast, the AME studies the contribution for individual players, as does the regular SV, and focuses on efficient estimation in the sparse regime. Our hierarchical setting considers fixed sources, and not all possible subsets.

Another related body of work is the work focused on defending against data poisoning attacks. Reject On Negative Impact (RONI) [9, 10] describes an algorithm that measures the LOO

effect of data points on subsets of the data. However, RONI is sample-inefficient and the paper does not prescribe a subset distribution to be used. As we explained previously, the choice of subset distribution can impact precision and recall. Another line of work [25, 76, 158, 163, 174] uses outlier detection to identify poisoned data. These are not query aware and thus can select benign outliers. Other approaches [47, 163, 178] assume the availability of clean data, or make strong assumptions about the model, e.g., assuming that linear models [89], or the attack, e.g., assuming that the attack is a source-agnostic trigger attacks [60], a trigger attack with small norm or size [31, 176, 183], or a clean-label attack [147]. None of these approaches can generalize across techniques.

Our work is also related to the existing literature on **data cleaning and management**. However, data cleaning approaches are not query driven, and must rely on other assumptions. As a result, many approaches depend on user provided integrity constraints [32, 33] or outlier detection [80, 132]. As a result these approaches cannot always identify poisoned data [103] and might also identify benign outliers. Recent approaches [48, 105], have also used another downstream DNN for data cleaning. However, these approaches assume that corrupt data has an influence on test set performance, an assumption that may not hold in scenarios such as data poisoning. Finally, Rain [196] is a recent query-driven proposal that proposes using influence function to explain SQL query results. While Rain shares similar goals, we focus on DNNs, a different use case.

Finally, our work leverages, and builds on, a large body of existing work from different fields. First, our work is related to the **causal inference** literature if we regard the inclusion of a data source as a treatment, from which AME is inspired. For instance, [88, 97] focuses on single treatment AME (i.e. only one source). Multiple treatments are introduced in [42, 50, 70], though their quantity uses different population distribution than ours, and different estimation techniques. In the computer science filed, Sunlight [114] uses a similar approach but with only one sampling probability and without knockoff procedure. Second, **sparse recovery** studies effi-

cient algorithms to recover a sparse signal from high dimensional observations. We leverage LASSO [112] –with properties related to those of compressive sensing [22]– and knockoffs [21]. Other approaches to the important factor selection problem exist, such as the analysis of variance (ANOVA) [16, 50, 148] used in [50], but we think LASSO is better suited to our use-case due to its scalability guarantees. Third, our goal is related to **group testing** [3, 193] as discussed in §2.1, and studying if and how group testing ideas could improve our technique is an interesting avenue for future work.

name	npoi	prec_dis	rec_dis	prec	rec	AI
ImageNet	5	100	80.0	100	100	5.500744
CIFAR10-50	50	96.4	54.0	2.1	100	0.788898
CIFAR10-50*	100	100	54.0	6.2	100	1.682893
CIFAR10-50*	101	100	54.5	7.9	100	1.744635
CIFAR10-50*	102	100	53.9	10.0	100	1.832493
CIFAR10-50*	103	100	54.4	10.4	100	1.826089
CIFAR10-50*	104	100	54.8	100	99.0	8.742186
CIFAR10-50*	105	100	54.3	100	99.0	8.916950
CIFAR10-50*	106	100	54.7	100	98.1	8.926177
CIFAR10-50*	107	100	54.2	100	98.1	9.054615
CIFAR10-50*	108	100	54.6	4.1	100	0.716676
CIFAR10-50*	109	100	54.1	100	98.2	9.374960
CIFAR10-50*	110	100	54.5	100	98.2	9.583294
CIFAR10-50*	120	100	54.2	100	97.5	11.441413
CIFAR10-50*	130	100	54.6	100	97.7	12.932239
CIFAR10-50*	140	100	54.3	100	98.6	13.600518
CIFAR10-50*	150	100	54.7	100	98.7	15.521987
CIFAR10-20	20	11.1	60.0	0.9	100	0.700744
CIFAR10-20*	40	27.9	60.0	1.7	100	0.793756
CIFAR10-20*	60	46.1	58.3	2.5	98.3	0.894113
CIFAR10-20*	80	52.8	58.8	3.4	98.8	0.970892
CIFAR10-20*	100	74.7	59.0	4.3	99.0	1.093217
CIFAR10-20*	120	80.5	58.3	5.2	99.2	1.253165
CIFAR10-20*	140	87.2	58.6	6.0	98.6	1.419778
CIFAR10-20*	160	89.5	58.8	7.0	98.8	1.527789
CIFAR10-20*	180	93.0	58.9	7.9	98.3	1.746322
CIFAR10-20*	200	97.5	59.0	9.2	98.5	1.884684
CIFAR10-20*	300	100	58.7	99.6	92.7	4.334019
CIFAR10-20*	400	100	58.8	100	93.0	7.996566
CIFAR10-20*	500	100	58.8	100	92.2	10.964163
CIFAR10-20*	600	100	58.7	100	92.3	14.168781
CIFAR10-20*	700	100	58.7	100	92.0	17.779293
CIFAR10-20*	800	100	58.8	100	91.6	20.268145
CIFAR10-20*	900	100	58.8	100	91.7	24.644904
CIFAR10-20*	1000	100	58.8	100	91.5	26.829417
EMNIST	10	100	80.0	100	100	84.367889

Table A.6: Full result of SCAn, where "*" indicates the dataset is supplemented, "_dis" indicates the distance-based score.

name	lambda	precision	recall
CIFAR10-20	3000.0	4.8	58.8
CIFAR10-20	30.0	18.0	58.8
CIFAR10-20	12.0	68.5	58.8
CIFAR10-20	11.9	68.9	58.8
CIFAR10-20	11.0	64.9	58.8
CIFAR10-20	3.0	1.0	58.8
CIFAR10-20	0.003	0.6	58.8
CIFAR10-50	3000.0	95.2	54.4
CIFAR10-50	30.0	99.9	54.4
CIFAR10-50	11.9	97.8	54.4
CIFAR10-50	3.0	0.7	54.4
CIFAR10-50	0.003	0.7	54.4
EMNIST	3000.0	100	78.9
EMNIST	3.0	100	78.9
EMNIST	0.003	100	78.9
ImageNet	3000.0	80.8	78.0
ImageNet	3.0	80.8	78.0
ImageNet	0.003	80.8	78.0
ImageNet	0.0001	84.9	78.0
ImageNet	4e-05	85.9	78.0
ImageNet	3e-05	85.9	78.0
ImageNet	2e-05	84.9	78.0

Table A.7: Full result of Representer Point

B | Appendix: Supplementary Materials for Chapter 3

B.1 Design details of MimicLDT and MetaLDT

In this section, we provide pseudocode for our proposed attacks and give a more detailed explanation of MimicLDT.

B.1.1 Pseudocode of MimicLDT

As described in Sec.3.4, to generate a poison graph $\mathcal{G}'$, we first inject fake nodes in $\mathcal{G}$ (Alg. 1) and then optimize the injected nodes' features using the gradient method (Alg. 2).

Algorithm 1 Inject Nodes – MimicLDT

Input: $\mathcal{G}$: benign graph; $\mathcal{V}_s$: list of attack points; Φ: upper bound of injected nodes per attack point

Output: Modified graph $\mathcal{G}'$ with injected nodes

$\mathcal{G}' \leftarrow \mathcal{G}$
foreach $v_s \in \mathcal{V}_s$ **do**
  $\mathcal{V}_{\text{subg\_s}} \leftarrow \{v_1, v_2, \ldots, v_\Phi\}$; `// Inject Φ nodes`
  $\mathcal{G}' \leftarrow \mathcal{G}' \cup \mathcal{V}_{\text{subg\_s}}$
  foreach *pair* (v_i, v_j) *where* $v_i, v_j \in \mathcal{V}_{subg\_s}$ **do**
    Add edge $e(v_i, v_j)$ to $\mathcal{G}'$ with probability $p = 0.5$
  repeat
    link_to_ap $\leftarrow$ random.binomial$(1, p, \Delta)$; `//` $p = 0.5$
  until *sum(link_to_ap)* > 0;
  for $i \leftarrow 1$ **to** Δ **do**
    if *link_to_ap[i] == 1* **then**
      Add edge $e(v_s, v_i)$ to $\mathcal{G}'$
  foreach $v_i \in \mathcal{V}_{subg\_s}$ **do**
    if *deg*(v_i) *== 0* **then**
      Remove v_i from $\mathcal{G}'$
return $\mathcal{G}'$

Algorithm 2 Injected Nodes Feature Optimization – MimicLDT

Input : Graph $\mathcal{G} = (\mathbf{A}, \mathbf{X})$; Injected nodes $\mathcal{V}_i$; Attack points $\mathcal{V}_s$; Target node $\mathcal{V}_t$; Pretrained GNN model $f_\theta(\cdot)$; Loss function L_{atk}

▷ *[r]Each injected node v_i is linked to its attack point $B(v_i)$

Parameter: β, MaxIters, learning rate α

Initialize $\mathbf{X}_{v_i} \leftarrow \mathbf{X}_{B(v_i)}, \quad \forall v_i \in \mathcal{V}_i$
Define $\mathcal{L}_{atk}$ as: $-\left(\frac{1}{|\mathcal{V}_s|}\sum_{v_s \in \mathcal{V}_s} Sim_f(\mathbf{h}_{v_s}^{(L)}, \mathbf{h}_{v_t}^{(L)}) + \beta \cdot \frac{1}{|\mathcal{V}_i|}\sum_{v_i \in \mathcal{V}_i} Sim_{in}(\mathbf{X}_{v_i}, \mathbf{X}_{B(v_i)})\right)$
for $t = 1$ **to** *MaxIters* **do**
  $\mathbf{h}_{v_s}^{(L)} \leftarrow f_\theta(v_s; \mathcal{G}^{(t-1)})$ $\mathbf{h}_{v_t}^{(L)} \leftarrow f_\theta(v_t; \mathcal{G}^{(t-1)})$ $\mathbf{X}_{\mathcal{V}_i}^{(t)} \leftarrow \mathbf{X}_{\mathcal{V}_i}^{(t-1)} - \alpha \nabla_{\mathbf{X}_{\mathcal{V}_i}} \mathcal{L}_{atk}(\mathbf{X}_{\mathcal{V}_i}^{(t-1)})$ ▷ *[r]For $v_j \notin \mathcal{V}_i$, $\mathbf{X}_{v_j}^{(t)} = \mathbf{X}_{v_j}^{(t-1)}$
return $\mathcal{G}^* = (\mathbf{A}, \mathbf{X}^*)$

B.1.2 Pseudocode of MetaLDT

MetaLDT performs its alternating optimization in rounds (Alg. 3). Within each round, it does one optimization step of the adjacency matrix optimization (Alg. 4) based on meta-gradient with constraints, and q optimization steps of the features matrix optimization (Alg. 5) based on the gradient descent across all injected nodes' all feature dimensions.

Algorithm 3 MetaLDT

Input: Graph $\mathcal{G} = (\mathbf{A}, \mathbf{X})$;
Labeled node labels C_L; Injected nodes $\mathcal{V}_{inj}$; Target node v_t;
Optimization rounds I; Feature optimization iterations q; Inner-training epochs T
Output: Optimized graph $\mathcal{G}' = (\mathbf{A}', \mathbf{X}')$

$\mathcal{G}' \leftarrow$ inject Δ nodes into $\mathcal{G}$ $\mathbf{X}'_{inj} \leftarrow \vec{0}$; `// Initialize features of injected nodes`
No linkages between $\mathcal{V}_{inj}$ and nodes in $\mathcal{G}$
for *rounds* $\leftarrow 1$ **to** I **do**
    Adjacency_Matrix_Optimization($\mathcal{G}', C_L, v_t, y_t, T$) **for** *iter* $\leftarrow 1$ **to** q **do**
        Feature_Matrix_Optimization($\mathcal{G}', C_L, v_t, y_t, T$)
return $\mathcal{G}' = (\mathbf{A}', \mathbf{X}')$

B.1.3 More Detailed Explanation for MimicLDT Attack

In MimicLDT attack, our feature optimization problem takes Eq.3.4 as the optimization formulation and uses a stochastic gradient descent based optimizer to compute feature vectors.

Injected nodes influence the *attack point*'s embedding through message passing. The first term of the loss function aims to make a attack point's embedding close to the target node's embedding. The optimization can only change features for injected nodes, because we assume that the attacker cannot modify others. But injected nodes are in the attack point's k-hop neighborhood, and can thus influence the attack point's final embedding due to GNN message passing. More formally, given a pre-trained GNN classifier f_θ and graph $\mathcal{G}'$, the embedding of nodes are $\mathbf{h}_{v_i}^{(L)} = f_\theta(v_i; \mathcal{G}'), \forall v_i \in \mathcal{V}(\mathcal{G}')$. Our goal is to ensure that the *target node*'s embedding v_t, $\mathbf{h}_{v_t} = f_\theta(v_t; \mathcal{G}')$, is close to the attack point v_s's embedding $\mathbf{h}_{v_s} = f_\theta(v_s; \mathcal{G}')$. Concretely, if we

Algorithm 4 Adjacency_Matrix_Optimization($\mathcal{G}', C_L, v_t, y_t, T$)

Data: $\mathcal{N}_k(v_t)$: k-hop neighbors of target node v_t

FLAG $\leftarrow$ True
for $t \leftarrow 1$ **to** T **do**
    $\theta_{t+1} \leftarrow \text{step}\big(\theta_t, \nabla_{\theta_t} \mathcal{L}_{train}(f_{\theta_t}(\mathcal{V}_L; \mathcal{G}'); C_L)\big)$
$\nabla^{meta}_{A'} \leftarrow \nabla_{A'} \mathcal{L}^h_{atk}(f_{\theta_T}(v_t; \mathcal{G}'); y_t)$
$S(u, v) \leftarrow \nabla^{meta}_{a_{u,v}}(-2 \cdot a_{u,v} + 1)$; `// Scoring each edge for update`
/* *Apply structural constraints* */ $S[\mathcal{V} \setminus \mathcal{V}_{inj},\ \mathcal{V} \setminus \mathcal{V}_{inj}] \leftarrow -\infty$ $S[\mathcal{V}_{inj},\ \mathcal{N}_k(v_t)] \leftarrow -\infty$ $S[\mathcal{N}_k(v_t),\ \mathcal{V}_{inj}] \leftarrow -\infty$
while *FLAG* **do**
    $\hat{e} \leftarrow \arg\max S(u, v)$; `// Edge proposal with highest score`
    if $v \in \mathcal{V}$ **then**
        if *exists* $e = (u, w) \in \mathcal{E}'$, *where* $w \in \mathcal{V}$ **then**
            $\hat{e} \leftarrow \arg\max S(u, v) \setminus \hat{e}$
        else
            FLAG $\leftarrow$ False
    else
        FLAG $\leftarrow$ False
$\mathbf{A}' \leftarrow$ Insert or remove $\hat{e}$ in $\mathbf{A}$

Algorithm 5 Feature_Matrix_Optimization($\mathcal{G}', C_L, v_t, y_t, T$)

for $t \leftarrow 1$ **to** T **do**
    $\theta_{t+1} \leftarrow \text{step}\big(\theta_t, \nabla_{\theta_t} \mathcal{L}_{train}(f_{\theta_t}(\mathcal{V}_L; \mathcal{G}'); C_L)\big)$
$\nabla^{meta}_{X'} \leftarrow \nabla_{X'} \mathcal{L}^h_{atk}(f_{\theta_T}(v_t; \mathcal{G}'); y_t)$
$\mathbf{X}'_{inj} \leftarrow \mathbf{X}'_{inj} - \alpha \nabla^{meta}_{X'_{inj}}$; `// Gradient update`
$\mathbf{X}' \leftarrow$ update $\mathbf{X}'_{inj}$, other nodes remain unchanged

consider a 2-layer GCN model, and attack point v_s, we compute $\mathbf{h}_{v_s}$ as:

$$\mathbf{h}_{v_s} = \sigma\left(\sum_{j \in \mathcal{N}_{v_s}} \frac{1}{c_{v_s,j}} \sigma\left(\sum_{m \in \mathcal{N}_j} \frac{1}{c_{j,m}} \mathbf{h}^{(0)}_{v_s} \theta^{(0)}\right) \theta^{(1)}\right) \tag{B.1}$$

where $c_{j,m} = \sqrt{\hat{d}_j \hat{d}_m}$, $\hat{d}_j = 1 + deg(j)$. When injected nodes are within the 2-hop neighborhood of the attack point v_s, they will pass their information to their neighbors and finally the attack point to influence the $\mathbf{h}_{v_s}$.

Attack stealthiness. The second term of equation 3.4 tries to ensure that injected nodes are

similar to other nodes in their neighborhood. We use feature vector similarity Sim_{in} as a proxy for homophily [30], though we also used node-centric homophily [30] in §??. To formulate in detail:

We use the following *CosineSimilarity* formulation:

$$Sim_{in} = \frac{\mathbf{X}_{v_i} \cdot \mathbf{X}_{B(v_i)}}{||\mathbf{X}_{v_i}||_2 ||\mathbf{X}_{B(v_i)}||_2} \tag{B.2}$$

We also did evaluation by using the following *node-centric homophily* formulation:

$$Sim_{in} = \frac{1}{|\mathcal{V}_i|} \sum_{v_i} sim(\sum_{j \in \mathcal{N}_{v_i}} \frac{1}{\sqrt{d_j}\sqrt{d_{v_i}}} \mathbf{X}_j,\ \mathbf{X}_{v_i}) \tag{B.3}$$

where $sim(\cdot)$ here is the *CosineSimilarity* and d_j represents the degree of node j.

Our empirical evaluation shows that both similarity metrics perform similarly.

B.2 The MetaLDT Attack via Optimization

MetaLDT is a meta-learning based long-distance node injection attack that we developed by extending meta-attack [227]. We provide a more complete description here.

We describe an optimization approach to solving the problem defined in section §3.3. Given information about how the model is trained as well as access to the original graph $\mathcal{G}$, a target node v_t, and a target label y_t, our optimization algorithm produces an attack graph $\mathcal{G}'$. To do this, we start with an initial attack graph $\mathcal{G}'_0$ and iteratively modify it to minimize the attacker's loss function $\mathcal{L}_{atk}$.

Inspired but different from traditional Meta-Attack [14, 227], we design an *iterative pipeline* that requires the optimization procedure to alternate between optimizing the adjacency matrix and optimizing node features. At the start of the process, MetaLDT produces an initial graph $\mathcal{G}'_0$ by injecting Δ new nodes into the input graph $\mathcal{G}$. These injected nodes have zeroed-out features,

and no edges connecting them to any other node. In each iteration i ($i \geq 0$), MetaLDT updates graph $\mathcal{G}'_i$ and produces the graph $\mathcal{G}'_{i+1}$, which the next iteration operates on. When producing $\mathcal{G}'_{i+1}$, we can either alter $\mathcal{G}'_i$'s adjacency matrix (thus adding or removing edges) or feature matrix (thus changing node features), and MetaLDT uses alternating minimization [86] to update both. Specifically, this means that our iterations alternate between changing the adjacency matrix and changing the feature matrix.

In each iteration i, we determine updates to the feature or adjacency matrix (as appropriate) using a computed meta-gradient $\nabla^{meta}_{\mathcal{G}'_i}$, which we compute by unrolling the model training loop for T epochs. Formally:

$$\begin{aligned}\nabla^{meta}_{\mathcal{G}'_i} &= \nabla_{\mathcal{G}'_i} \mathcal{L}_{atk}\left(f_{\theta_T}\left(\mathcal{G}'_i\right)\right) \\ &= \nabla_f \mathcal{L}_{atk}\left(f_{\theta_T}\left(\mathcal{G}'_i\right)\right) \cdot \left[\nabla_{\mathcal{G}'_i} f_{\theta_T}\left(\mathcal{G}'_i\right) + \nabla_{\theta_T} f_{\theta_T}\left(\mathcal{G}'_i\right) \cdot \nabla_{\mathcal{G}'_i} \theta_T\right]\end{aligned} \tag{B.4}$$

where the last term is recursively defined as $\nabla_{\mathcal{G}'_i} \theta_{t+1} = \nabla_{\mathcal{G}'_i} \theta_t - \alpha \nabla_{\mathcal{G}'_i} \nabla_{\theta_t} \mathcal{L}_{train}\left(f_{\theta_t}\left(\mathcal{G}'_i\right)\right)$, and α is the learning rate. Observe that computing this meta-gradient does not require access to the model used by the attack's victim, but requires running T training epochs, using the same training setting (i.e., the same algorithm and approach) as used by the victim. In the rest of the paper, we use the term *surrogate model* to refer to models trained by the attacker using the same process as the victim.

B.2.1 Changing Graph Structure

Iterations that alter the adjacency matrix assume that the node feature matrix is a constant. Consequently, we can treat $\nabla^{meta}_{\mathcal{G}'_i}$ as the meta-gradient for the graph $\mathcal{G}'_i$'s adjacency matrix A_i, and can compute a *meta-score* [227], $S(u,v) = \nabla^{meta}_{\mathcal{G}'_i}[a_{uv}] \cdot (-2 \cdot a_{uv} + 1)$ for each pair-of-nodes (u,v), where $[a_{uv}]$ indicates that we indexed the value at position (u,v) in $\nabla^{meta}_{\mathcal{G}'_i}$'s adjacency matrix. Our approach is predicated on the observation that altering the adjacency matrix for the

pair (u, v) with the highest computed meta-score $S(u, v)$ is likely to best decrease the attacker's loss $\mathcal{L}_{atk}$.

However, our assumptions (§3.3) limit what adjacency matrix modifications the attacker can perform, and so we only consider a subset of node pairs in this process. In particular, we impose the following constraints on the node-pairs we consider: (a) either u or v must be an injected node; (b) neither u nor v can be within v_t's k-hop neighborhood, thus ensuring that the attacks are long-distance; and (c) that an injected node u has no more than one-edge connecting it to a node in the original graph $\mathcal{G}$, a constraint we add to avoid cases where the optimization spends all of its time optimizing a single injected node. We evaluate the effect of the last optimization in Appendix B.4.3.

Thus, iterations that change the adjacency matrix compute a score $S(u, v)$ for any pair of nodes (u, v) that meet our constraints, identify the pair (u_m, v_m) with the largest score, and then adds edge (u_m, v_m) if none exists or removes it if it already exists.

B.2.2 Changing Node Features

Similarly, iterations where node-features are changed assume that the adjacency matrix is a constant, and therefore use $\nabla^{meta}_{\mathcal{G}'_i}$ as the meta-gradient for the feature matrix.

However, different from traditional meta-score based selection, we use a *gradient descent based feature optimizer*, which allow us to change the feature of all injected nodes in a single feature-optimization iteration (rather than requiring changes to a single feature dimension of a single node at a time). in this case, we do not use $\nabla^{meta}_{\mathcal{G}'_i}$ to compute a scoring function to select and then update node features for a single node. Instead, we use $\nabla^{meta}_{\mathcal{G}'_i}$ to compute feature gradients which we use to update $\mathcal{G}_i$'s feature matrix. Care must be taken when doing so, since we assume attacker cannot change features for any nodes already present in the input graph $\mathcal{G}$ (§??). We impose this constraint by zeroing out the corresponding elements in $\nabla^{meta}_{\mathcal{G}'_i}$'s feature matrix, and in what follows we refer to the resulting matrix as $X_{\nabla_{\mathcal{G}'_i}}$. Given this, we compute: $\hat{X}_{\mathcal{G}'_{i+1}} = X_{\mathcal{G}'_i} - \alpha X_{\nabla_{\mathcal{G}'_i}}$

Table B.1: Dataset statistics.

Dataset	Nodes($\|V\|$)	Edges($\|E\|$)	Classes($\|Y\|$)	Labeled nodes
Ogbn-ArXiv	169343	1157799	40	90941
SciBERT-embed-ArXiv	169343	1157799	40	90941
PubMed	19717	44338	3	18217
Cora	2708	5429	7	1708

where $X_{\mathcal{G}_i'}$ is $\mathcal{G}_i'$'s feature matrix, and α is the learning rate.

Empirically, we found that a single gradient update is often insufficient, so in practice each iteration repeats this process q times (and we compute a new $\nabla^{meta}_{\mathcal{G}_i'}$ after each update).

B.3 Details of experimental setup

Dataset statistics Table. B.1 shows detailed statistics for the datasets used in the evaluation.

Model settings By default, all GNNs used in the experiments have 3 layers (except 2 layers GNN models for Cora), a hidden dimension of 16 for Cora, PubMed, and a hidden dimension of 256 for the ArXiv dataset. We adopt dropout with dropout rate of 0.5 between each layer (i.e., 0.6 for GAT model). We use $5e-4$ weight decay for models except the training for ArXiv graph. By default, we set the maximum GNN models training epochs as 1000 and do the early stopping of 100 epochs by examining the validation accuracy.

Hyperparameter settings The hyperparameters for MimicLDT and MetaLDT are shown in Table B.2 and Table B.3, respectively.

Table B.2: Hyperparameters for MimicLDT.

GNN	Dropout	Weight Decay	LR	Max epochs	Patience	Others
GCN	0.5	0.0005	0.01	1000	100	
GraphSAGE	0.5	0.0005	0.01	1000	100	
GAT	0.6	0.0005	0.01	1000	100	
GNNGuard	0.5	0.0005	0.01	1000	50	$\epsilon = 1e-2$
SoftMedianGDC	0.5	0.0005	0.01	1000	50	$k = 64$, $\alpha = 0.15$, $T = 1.0$
JaccardGCN	0.5	0.0005	0.01	1000	50	$\epsilon = 0.01$
SVDGCN	0.5	0.0005	0.01	1000	50	Rank = 2000 (ArXiv) and 50 (others)
ProGNN	0.5	0.0005	0.01	1000	50	$\alpha = 5e-4$, $\beta = 1.5$, $\gamma = 1.0$, $\lambda\_ = 0$, $\phi = 0$, $\text{lr}_{adj} = 0.01$

Table B.3: Hyperparameters for MetaLDT.

GNN	Dropout	Weight Decay	Inner-train-epochs	Others
GCN	0.5	0.0005	200	
GraphSAGE	0.5	0.0005	200	
GAT	0.6	0.0005	200	
GNNGuard	0.5	0.0005	50	$\epsilon = 1e-6$
SoftMedianGDC	0.5	0.0005	50	$k = 64$, $\alpha = 0.15$, $T = 1.0$
JaccardGCN	0.5	0.0005	200	$\epsilon = 0.01$
SVDGCN	0.5	0.0005	200	Rank = 50
ProGNN	0.5	0.0005	50	$\alpha = 0.1$, $\beta = 10.0$, $\gamma = 1.0$, $\lambda\_ = 0$, $\phi = 0$, $\text{lr}_{adj} = 0.01$

B.4 Additional evaluations on MetaLDT

B.4.1 Alternating vs. combined optimization of adj. matrix and node feature

Apart from adding constraints to enforce long-distance, a key difference of MetaLDT over existing meta-learning based GNN attack [227] is that MetaLDT performs alternating optimization of the adjacent matrix and node features as opposed to combining both in one step. We also tried the combined optimization used in [227] and achieved poison success rate of 65.5% for GCN on Cora, which is significantly less than 96% achieved by the alternating optimization approach. We find that the combined optimization is heavily biased towards modifying the adjacency matrix as opposed to node feature, as the gain from changing a node feature dimension is much smaller than that of adding/deleting an edge.

B.4.2 Benefits of optimizing the adjacency matrix

Compared to MetaLDT, our faster attack MimicLDT does not optimize the connections between injected and existing nodes but simply connects injected nodes to each other and to randomly chosen attack points with the target label. To understand the additional benefits of optimizing the adjacency matrix, we run MetaLDT over a fixed adjacency matrix like that used by MimicLDT. MetaLDT is only allowed to optimize the fake nodes' features, like MimicLDT. This feature-only MetaLDT achieves poison success rate of 79.5% for GCN over Cora, compared to 96% achieved by the full MetaLDT. However, if we are to connect fake nodes to attack points with random labels instead of target labels, the poison success rate drops dramatically to 22%. This shows that MimicLDT's heuristic of forming connections is a good, albeit still imperfect, strategy.

Table B.4: Poison success rate with varying q for GCN over Cora. $I = 68$.

q	1	10	1000
Poison Success Rate	40.5%	52%	96%

B.4.3 Importance of constraining edges between injected and existing nodes

Apart from constraining each injected nodes to only connect to existing nodes outside of the target's k-hop neighborhood, MetaLDT also constrains each injected node to connect to at most one existing node. We run experiments without this constraint. Not only the resulting poison success rate is worse (71%), the generated poisoned graphs tend to connect only a very small number of fake nodes (i.e, ≤ 3) each of which have a large number of connections with different existing nodes. Thus, due to their high degree, the injected fake nodes are hardly inconspicuous.

B.4.4 Effects of hyperparameters

We evaluate the effects of q and I. Hyperparameter q refers to the number of optimization steps on node features for each optimization step of the adjacency matrix. Hyperparameter I refers to the total number of rounds where each round takes one optimization step on adjacency matrix and q steps on node feature. Table. B.4 shows the effects of varying q while keeping $I = 68$ unchanged. As we can see, the success rate improves with larger q. We believe this is because larger q allows feature optimization to converge better for each given adjacency matrix change. By default, our experiments use $q = 1000$.

Increasing the rounds of optimization could make MetaLDT convergence to a higher poison success rate at the cost of extra running time. As shown in Figure. B.1, MetaLDT's optimization is quite efficient: at $I = 68$, the poison success rate has mostly converged. Since the number of optimization rounds must match or exceed the number of allowed adjacency matrix modification

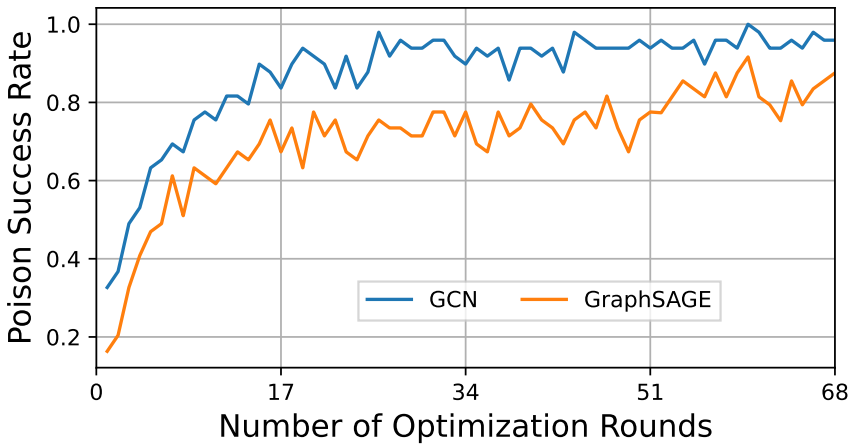


Figure B.1: Poison success rate as the rounds of optimization increases.

($I \geq \Delta = 68$), $I = 68$ is the smallest sensible I.

B.4.5 Importance of adapting MetaLDT to GNN defenses

In Section 3.5.2 (Table 3.1), we show MetaLDT's performance when its inner training adapts to the GNN defense mechanisms used. The implementation of the adaption follows the work [137]. We also experimented with non-adaptive MetaLDT its surrogate model is the vanilla GCN, no matter what defense mechanism is. As shown in Table. B.5, non-adaptive MetaLDT fares much worse than MetaLDT, e.g. lowering success rate to 14% from 62% for SVD. Our finding is consistent with that reported by [137].

B.5 Additional evaluations on MimicLDT

B.5.1 Effect of Φ

We evaluate how the value of Φ, which determines the number of nodes injected per attack point, affects poison success rate. We show the results in Figure B.2 show the poison success rate with varying Φ for GraphSAGE over ArXiv. We observe that the success rate increases as the

Table B.5: Poison success rate of *MetaLDT* on Cora. In the adaptive setting, MetaLDT's inner training loops takes into account the GNN defense used, and vice versa. To handle OOM cases, we also evaluate MetaLDT in a setting with fewer inner training epochs (50) instead of the default (200). This setting is referred to as (ET, early termination).

	Vanilla			Robust				
	GCN	GraphSAGE	GAT	GNNGuard	SoftMedianGDC	JaccardGCN	SVDGCN	ProGNN
MetaLDT (non-adaptive)	0.96	0.76	0.51	0.19	0.30	0.72	0.14	0.31
MetaLDT-Adaptive.	0.96	0.87	0.84	OOM	OOM	0.91	0.83	OOM
MetaLDT-Adaptive (ET)	0.69	0.73	0.68	0.53	0.58	0.86	0.62	0.55

Table B.6: The average graph degree distribution changes for different datasets according to the *EMD* metric.

Dataset	Cora	PubMed	ArXiv
EMD (degree)	0.0393 ± 0.0021	0.0550 ± 0.0008	0.1189 ± 0.0007

value of Φ increases, but the additional benefit is small beyond $\Phi = 4$, which is our default value.

B.5.2 More on attack graph's node degree distribution

Table. B.6 shows the node degree distribution change as a result of the MimicLDT attack for different datasets. The change is measured in the Earth Mover's Distance (EMD) metric.

B.5.3 More on attack graph homophily

In MimicLDT, we use feature vector similarity as a proxy for homophily. We also evaluate a variant of MimicLDT, called MimicLDT-homo, that uses the original node homophily metric [30] in the loss function. Figure B.3 shows that both MimicLDT and MimicLDT-homo can preserve graph homophily well.

We additionally measure changes in node homophily distribution after the attack using the

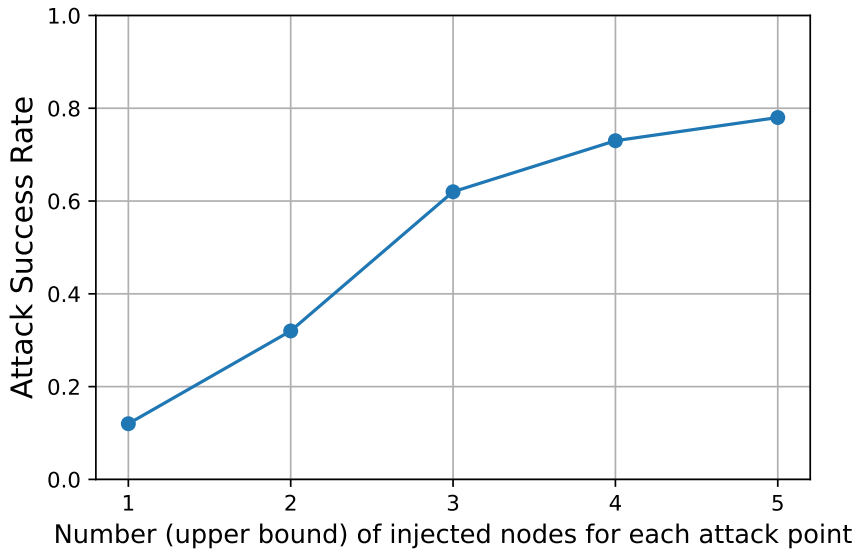


Figure B.2: Attack success rate of varying Φ, which is the upper bound number of injected nodes for each attack point. keep other settings and hyperparameters to be the same.

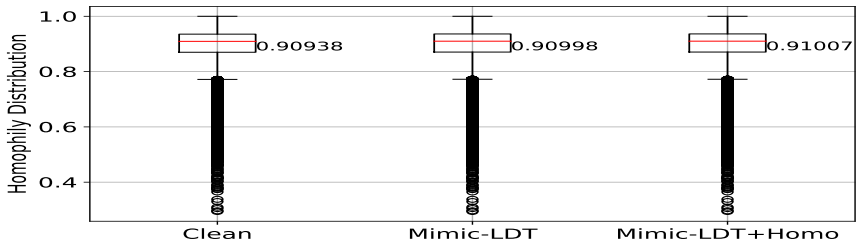


Figure B.3: Homophily distribution for clean and poisoned graph over ArXiv. MimicLDT-homo refers to a variant of MimicLDT that uses the node-centric homophily metric for its loss function.

EMD metric for all datasets, shown in Table B.7. As can be seen, the attacks only result in slight changes on graph homophily distribution.

B.5.4 Non-Adapting MimicLDT to GNN defenses

In Section 3.5.2 (Table 3.1), we show MimicLDT's performance when the surrogate model knows what the GNN defense mechanisms used. We also experimented with non-adaptive MimicLDT, where the attacker who uses a GCN as the surrogate model either did not know what model was being used (e.g. GAT) or was unaware of what defenses were in use, and measured

Table B.7: The average graph homophily distribution changes for different datasets according to the *EMD* metric.

Dataset	Cora	PubMed	ArXiv
EMD (homophily)	0.0205 ± 0.0010	0.0123 ± 0.0004	0.0008 ± 0.0003

Table B.8: Poison success rate of *MimicLDT* on Cora and ArXiv. Experiments about either the surrogate model account for what GNN defenses are used, or does not account for GNN defenses and instead use vanilla GCN as the surrogate model.

	Vanilla			Robust				
	GCN	GraphSAGE	GAT	GNNGuard	SoftMedianGDC	JaccardGCN	SVDGCN	ProGNN
Cora (Aware the defense)	0.67	0.63	0.60	0.70	0.55	0.66	0.74	0.59
Cora (Vanilla GCN)	0.67	0.63	0.59	0.68	0.29	0.65	0.35	0.53
ArXiv (Aware the defense)	0.74	0.73	0.70	0.64	0.59	0.63	0.62	0.58
ArXiv (Vanilla GCN)	0.74	0.72	0.67	0.64	0.42	0.61	0.49	0.55

the efficacy of MimicLDT. We show results of Cora and ArXiv datasets in the Table. B.8 (the rows corresponding to "vanilla GCN"). Basically, our proposed attack assumes knowledge of the victim model, and we do not believe it can be generalized to unknown victim models in theory. However, empirically, the situation is a bit more complex. As shown in Table. B.8, the results show that in many cases (e.g. GAT) using a vanilla GCN based surrogate model can poison as successfully as using a surrogate based on the correct victim model. However, we also see a significant drop in poison success rate in other cases (e.g., SoftMedian, SVDGCN).

B.6 Comparison to short-distance baselines

We consider two different types of the short-distance attacks: short-distance modification attacks, and the node injection poisoning attacks.

B.6.1 Comparing with short-distance modification attacks

We compare against three existing short-distance attacks in the targeted poisoning setting: **Nettack** [226], **FGA** [26] and **IG-FGSM** [195].

We use the implementation from DeepRobust's[1], but modify the loss function so that the attacker's goal is the same as ours, which is to flip the target node's label to a specific label of the attacker's choosing instead of *any* arbitrary incorrect label.

The baselines are modification attacks, aka they can either perturb the graph structure or existing nodes' features. By contrast, MimicLDT and MetaLDT are injection attacks. Thus, it is impossible to directly compare them. Therefore, to make these baseline attacks somewhat comparable to ours, we limit the existing attacks to perturb the graph structure only. From a practical perspective, it is much harder (or even impossible) for an attacker to modify an existing node's features than to create a link to it.

Baseline attacks setups. In order to compare with MimicLDT attack, we did experiments on baseline attacks (i.e., Nettack, FGA, IG-FGSM) according to the Table. B.9. Because for each graph in MimicLDT attack, in expectation, the new edges created to link with the real nodes is $p * \Delta = p * \Phi * r|\mathcal{V}_L|$. In our experiment, $\Phi = 4$, $p = 0.5$, therefore the average newly created edges is $2r|\mathcal{V}_L|$. Thus, correspondingly, we allow the number of perturbations in each baseline attack to be $2r|\mathcal{V}_L|$ (i.e., $r = 1\%$ in Cora; 0.5% for ArXiv and PubMed). As for the indirect attack (i.e., Nettack-indirect), the number of influencers (i.e., the attack perturbs the edges that connect to up to how many of the target's neighbors) should be at least greater than average degree. As

[1]Open-sourced DeepRobust library: https://github.com/DSE-MSU/DeepRobust

Table B.9: Hyperparameter setups for short-distance modification attacks. Direct attacks includes Nettack-direct, FGA and IG-FGSM; Indirect attack includes Nettack-indirect.

		Mimic-LDT	Direct attacks	Indirect attacks	
Dataset	Avg. Degree	r	perturbations	perturbations	influencers
Cora	3.84	0.01	34	34	10
PubMed	4.50	0.005	182	182	20
ArXiv	13.67	0.005	N/A	N/A	N/A

mentioned in [226], increasing the number of influencers will greatly increase the running time for attacks. We set the number of influencers to be 10 for Cora and 20 for PubMed.

B.6.2 Comparing with short-distance modification attacks with varying edge perturbation budgets

Here, we show a comparison by varying the edge perturbation budgets in baseline attacks and MimicLDT attack for Cora dataset and GCN model. We used three settings (i.e., add one more setting) for Nettack here: direct where the target node's edges are perturbed; indirect with 2 influencers, where the attack perturbs the edges that connect to up to 2 of the target's direct neighbors; and indirect with 10 (which is more than Cora's average node degree) influencers.

Figure B.4 shows baseline attacks' poison success rate as we vary their edge perturbation budget, while Figure B.5 shows MimicLDT's success rate as we vary the number of edges between attack points and injected nodes. As we probabilistically connect fake nodes to their associated attack points, the number of edges between the two, as shown as the x-axis in Figure B.5, is in expectation. We can see that all baseline attacks, especially direct attacks (Nettack-direct, FGA, IG-FGSM), are much more efficient than our attack in terms of the number of edge budget required. Indirect (aka influence) attacks are much less efficient than direct ones, but they can still be more efficient than our attack under some settings (e.g., Nettack indirect with 10 influencers when budget is 17). However, we note that an indirect attack is not a long-distance attack, since it

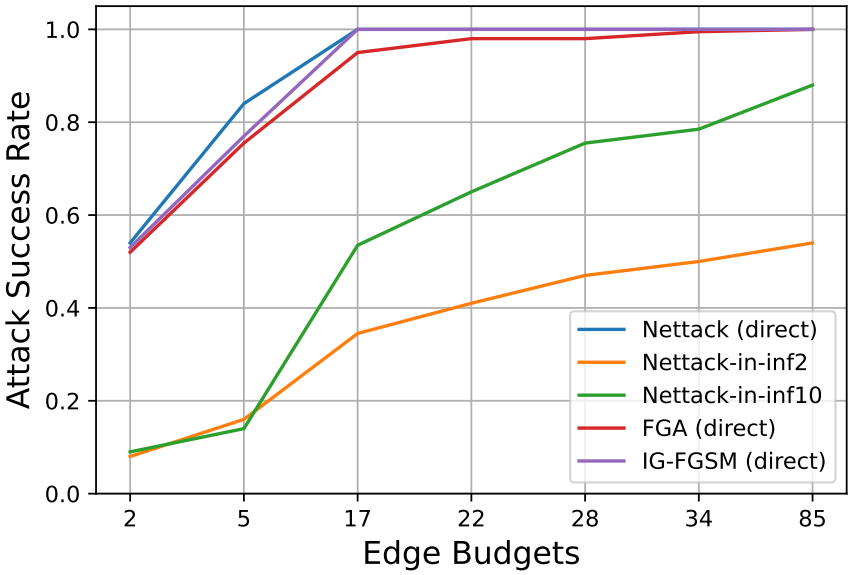


Figure B.4: Attack success rate when varying the edge perturbation budget for direct Nettack, indirect Nettack with 2 or 10 influencers, FGA and IG-FGSM.

must modify the target's neighbors. This shows that long-distance attacks, which cannot change the target neighborhood, carry an efficiency cost.

B.6.3 Comparing with node-injection poisoning attack

We also evaluated the efficacy of **AFGSM** [185], a well cited recent node-injection poisoning attack. And we'd also like to note that most recent injection attacks, including, TDGIA [225], GIA-HAO [30], CANA [164], G^{2}A2C [96], are test-time evasion attacks, so we cannot directly compare against them.

As comparison with [185], we used the author's implementation, but modified it to perform targeted label-flipping poisoning instead of misclassifying the target to be any arbitrary class. We perform the results of direct attacks (injected nodes can be direct neighbors of the target node) as well as indirect attacks (injected nodes cannot be direct neighbors but can be k-hop neighbors).

Baseline attacks setups. As for the experiments settings, the number of injected nodes and the number of injected edges are two critical hyperparameters, however the paper itself does

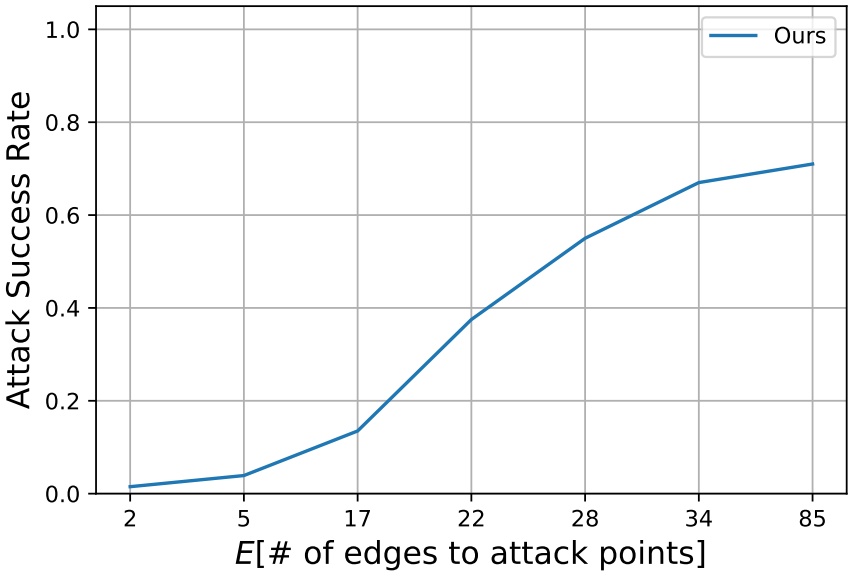


Figure B.5: Attack success rate as a function of the number of edges connecting injected nodes and attack points.

not provide clear guidance on parameter settings. For direct attacks, in order to avoid obvious damage to the graph structure distribution, we make it could inject 2 nodes directly with each target, and the budget of injected edges equal to the number of injected nodes times the average node degree of the dataset. For indirect attacks, although AFGSM does not make it long-distance attack, we use the same setting as that in our work to compare (i.e., $\Phi * r|\mathcal{V}_L|$ injected nodes and $p * \Phi * r|\mathcal{V}_L|$ injected edges).

Comparing the attacks performance. Detailed results are in §B.7 (Table B.10 and B.11). We can see that the direct-attacks of AFGSM on vanilla GCN have poison success rate 74% for Cora and 83% for PubMed, which is higher than what MimicLDT achieved (67% and 71%). However this direct-attack shows more easily to be destroyed by defenses, and have the poison success rate over robust models ranging from 3-70% for Cora and 10-72% for PubMed, which are lower than what we achieved (55-74% for Cora, 56-70% for PubMed). For comparing with indirect-attacks, our MimicLDT outperforms AFGSM in both vanilla and robust models. The indirect-attacks have a poison success rate ranging from 30-64% for Cora and 15-45% for PubMed, which are lower than

what we achieved (55-74% for Cora, 56-71% for PubMed).

B.6.4 Diagnosing baseline attacks using GNN explainability tools

By modifying the target node's k-hop neighborhood, baseline attacks are vulnerable to existing GNN explainability tools. We evaluate this by measuring the likelihood that GNNExplainer [206] can detect perturbations from the baseline attacks.

GNNExplainer takes as input a target node, graph and GNN model, and returns the subgraph that has the largest influence on the target node's prediction. Our experiments use the GNNExplainer implementation included in the PyTorch-Geometric library[2] to find this subgraph for the targeted node, and we use this returned subgraph to compute precision and recall. The baseline attacks can both add and remove edges. However, GNNExplainer cannot indicate removed edges in its explanation. Therefore, we only consider edges added by the attack when computing precision and recall: *recall* is the fraction of edges added by the attack contained in the GNNExplainer subgraph, while *precision* is the fraction of edges in the subgraph that were added by the attack:

$$recall = \frac{\text{Number of attack-added edges exist in GNNExplainer subgraph}}{\text{Number of total attack-added edges}} \tag{B.5}$$

$$precision = \frac{\text{Number of attack-added edges exist in GNNExplainer subgraph}}{\text{Number of total edges in GNNExplainer subgraph}} \tag{B.6}$$

Then F1 score is calculated based on the precision and recall.

Effect of varying threshold. GNNExplainer allows users to specify a threshold, and selects only those edges whose importance score is above the specified threshold. We show results for three thresholds: 0.0 (the default value), 0.6 and 0.8. By runnig GNNExplainer on various short-distance attacks setups over Cora dataset, we find that GNNExplainer can find a significant fraction of the perturbed edges, and that at higher thresholds (like 0.8) it has a recall that is around 0.5 (so it finds basically half the added edges), while also having a precision ranging from $0.52 - 0.93$ thus making it a feasible tool for inspection and defense against such attacks. When using lower

[2] https://pytorch-geometric.readthedocs.io/en/latest/modules/explain.html

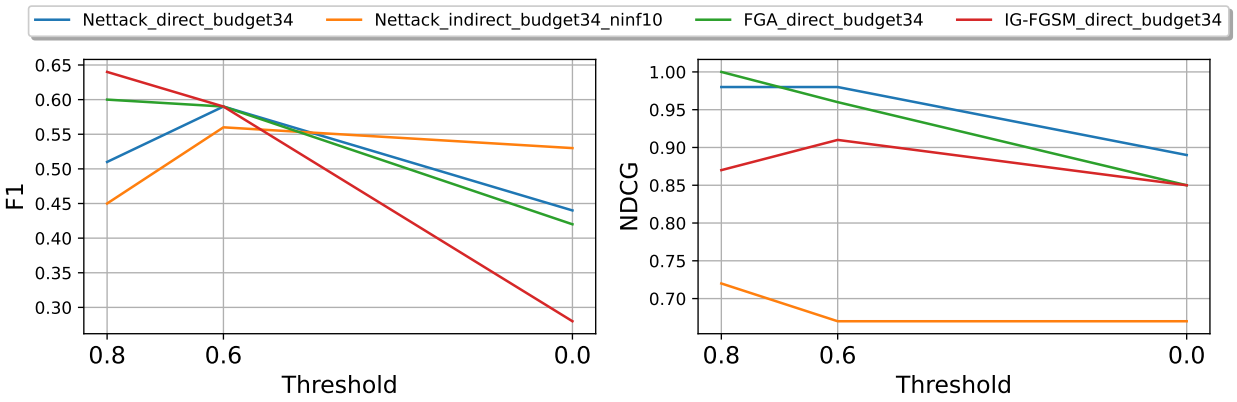


Figure B.6: Effect of GNNExplainer on various short-distance attacks over Cora dataset under the metrics of NDCG/F1 scores.

threshold (like 0.0), almost all added adversarial edges can be accurately selected (see Figure. 3.5 in §3.5.2).

Another important metrics people usually care about is **NDCG** scores, which we only check the rank position (of truely perturbed edges) through the selected out edges (i.e., the subgraph GNNExplainer returned after threshold). Because higher values of NDCG indicates that the adversarial edges present at top ranks and could be more noticeable by people (such as system inspectors or designers). So here in Figure. B.6, we show the effect of varying threshold under the NDCG scores and F1 scores. We can see that at higher thresholds (like 0.8), the NDCG scores are 0.87-1.0 for directed attacks and 0.72 for indirect attack, which shows the selected adversarial perturbations are very noticeable.

Remove the detected edges and retrain. To even complete the pipeline, we examine the poison success rate by removing these detected malicious edges and retraining the models for the target node's prediction. In Figure. 3.5 in §3.5.2, for different short-distance attacks settings, we show the attack success rate(ASR) before detection and the ASR after the *remove and retrain*. The poison success rate decreased from 100% to 4-7% for direct attacks, and decreased from 79% to 3% for indirect attacks, which shows the short-distance attacks are defended against.

Table B.10: Detailed Results on Cora.

		Long-distance		Short-distance (targeted, modification)				Node-injection (AFGSM)	
		MimicLDT	MetaLDT	Nettack-direct	Nettack-indirect	FGA	IG-FGSM	Direct	Indirect
Vanilla	GCN	0.67	0.96	1.00	0.79	0.98	1.00	0.74	0.64
	GraphSAGE	0.63	0.87	0.96	0.42	0.58	0.70	0.68	0.53
	GAT	0.60	0.84	0.97	0.53	0.72	0.86	0.65	0.55
Robust	GNNGuard	0.70	(0.53)	1.00	0.98	0.94	0.96	0.69	0.61
	SoftMedianGDC	0.55	(0.58)	1.00	0.46	0.88	0.94	0.49	0.52
	JaccardGCN	0.66	0.91	1.00	0.47	0.90	0.96	0.60	0.33
	SVDGCN	0.74	0.83	1.00	0.18	0.96	1.00	0.03	0.30
	ProGNN	0.59	(0.55)	1.00	0.60	0.90	0.97	0.57	0.58

Table B.11: Detailed Results on PubMed. *OOM* means out-of-memory under the GPU limitation.

		Long-distance	Short-distance (targeted, modification)				Node-injection (AFGSM)	
		MimicLDT	Nettack-direct	Nettack-indirect	FGA	IG-FGSM	Direct	Indirect
Vanilla	GCN	0.71	1.00	1.00	1.00	1.00	0.83	0.28
	GraphSAGE	0.69	1.00	0.84	1.00	OOM	0.72	0.45
	GAT	0.69	1.00	0.82	1.00	OOM	0.71	0.42
Robust	GNNGuard	0.70	1.00	1.00	1.00	OOM	0.79	0.45
	SoftMedianGDC	0.56	1.00	0.45	1.00	OOM	0.52	0.38
	JaccardGCN	0.67	1.00	1.00	1.00	1.00	0.10	0.15
	SVDGCN	0.60	1.00	0.09	1.00	0.97	0.58	0.32
	ProGNN	0.57	1.00	0.58	1.00	OOM	0.63	0.44

B.7 Detailed results over different datasets and short-distance baselines

In this section, we give the detailed results of MetaLDT over Cora and MimicLDT over all datasets, as well as the detailed results for short-distance baselines. We report the results by datasets: Table B.10 (Cora), Table B.11 (PubMed), and Table B.12 (ArXiv). These tables correspond to the same experiments as the summary table presented in §3.5 (Table 3.1).

Table B.12: Detailed Results on ArXiv. *OOM* means out-of-memory under the GPU limitation.

		Long-distance	Short-distance (targeted, modification)				Node-injection (AFGSM)	
		MimicLDT	Nettack-direct	Nettack-indirect	FGA	IG-FGSM	Direct	Indirect
Vanilla	GCN	0.74	OOM	OOM	OOM	OOM	OOM	OOM
	GraphSAGE	0.73	OOM	OOM	OOM	OOM	OOM	OOM
	GAT	0.70	OOM	OOM	OOM	OOM	OOM	OOM
Robust	GNNGuard	0.64	OOM	OOM	OOM	OOM	OOM	OOM
	SoftMedianGDC	0.59	OOM	OOM	OOM	OOM	OOM	OOM
	JaccardGCN	0.63	OOM	OOM	OOM	OOM	OOM	OOM
	SVDGCN	0.62	OOM	OOM	OOM	OOM	OOM	OOM
	ProGNN	0.58	OOM	OOM	OOM	OOM	OOM	OOM

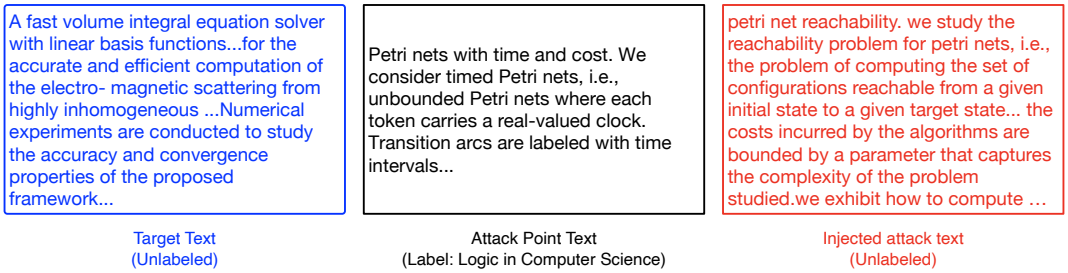


Figure B.7: An example text generated for a fake node. The attack manages to flip the target node's label from *Numerical Analysis* to *Logic in Computer Science*.

B.8 Design and evaluation of end-to-end attacks

We describe our extension to MimicLDT that allows it to generate textual features for fake nodes to attack GNN models over citation graphs.

B.8.1 End-to-end attack design

We assume that the attacker has access to the encoder model weights used by the GNN to embed textual node features. This assumption is reasonable as pretrained language models, such as SciBERT [13], are widely available for embedding purpose.

Our design requires the attacker to use the encoder and some corpus of text to train a *decoder model* that can be used to generate fake texts from arbitrary embedding. The *decoder* model is modified from an existing encoder-decode architecture such as SciBERT. We change the decoder

to take a single embedding vector as input instead of an embedding matrix containing a vector for each token. We also modify the overall encoder-decoder so that the encoder's output is the final embedding of the classification token (`[CLS]`) which is then used as the decoder's input.

To train the decoder, we fine-tune a pre-trained encoder-decoder model. During the fine-tuning process, we use the victim's encoder weights (which are provided as an input) and freeze them. The fine-tuning process can thus only update decoder weights, and must do so to ensure that the output sequence is close to the input sequence.

Once we have trained the decoder model, we can use it for end-to-end attack as follows. First, we generate the fake nodes using MimicLDT (or MetaLDT) and attach them to the existing graph accordingly. Then, we take the continuous feature vectors of each injected node and pass them as input to the decoder model, which generates text that the attacker can use to create the corresponding fake nodes with raw textual features.

B.8.2 End-to-end attack evaluation

We evaluate the end-to-end attack for GraphSAGE over arXiv with SciBERT embedding. We refer to this dataset as SciBERT-embed-arXiv which is generated by passing the title and abstract of each paper in the Ogbn-arXiv dataset through SciBERT, and using the encoder's output (i.e., the [CLS] token embedding after the final pooling layer) as the node's feature in the graph. As we previously stated (§3.4), across 50 experiments, we observed a poison success rate of 84%. Furthermore, our manual inspection showed that the generated title and abstract are readable, and we show examples in Table B.13 and Table B.14.

Finally, we also found that the text generated by the decoder lead to node features that are close to what was generated by the attack optimization formulation: Figure B.8 shows a CDF of node feature similarity between nodes injected by the end-to-end attack (after they have been passed through an encoder) and injected node features generated by the attack optimization, and over 80% of the nodes have a cosine distance smaller than 0.1.

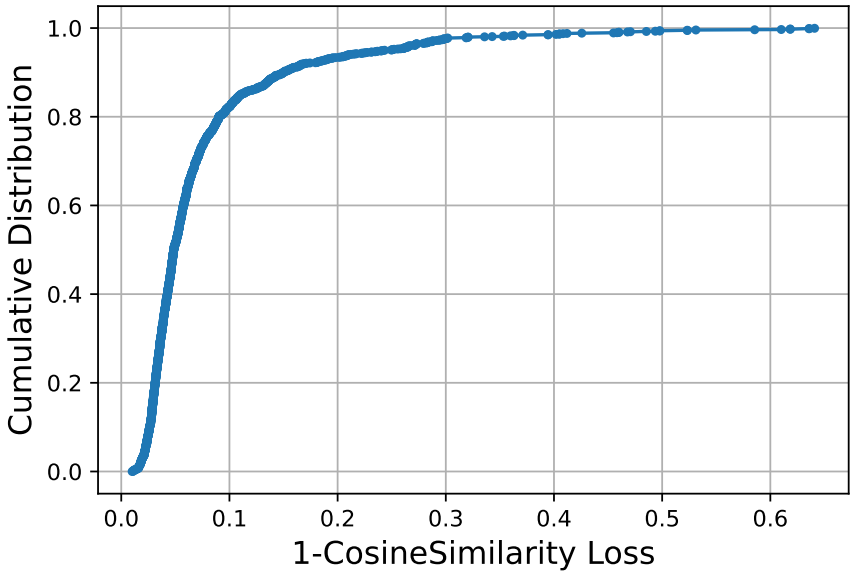


Figure B.8: An example of successfully poisoned end-to-end attack: the CDF of the 1-CosineSimilarity loss of f_{opt} and f'_{opt} for all injected nodes.

Case study. In Fig B.7, we show an example text generated for an unlabeled fake node to flip the target node's label from "Numerical Analysis" to "Logic in Computer Science". Addtional examples can be found in Table B.13 and Table B.14.

Table B.13: More examples (Part-1) of target node, one base node, and one of the generated fake-text link to this base. The category (i.e., labels) of the base/target node shown in first column, generated injected node is unlabelled. Due to space limit, only show partial of the contents.

	Generated fake-text *(link to base)*	*petri net reachability. we study the reachability problem for petri nets, i.e., the problem of computing the set of configurations reachable from a given initial state to a given target state... the costs incurred by the algorithms are bounded by a parameter that captures the complexity of the problem studied.we exhibit how to compute optimal solutions...*
Logic in Computer Science	Base: Title and Abstract	Petri nets with time and cost. We consider timed Petri nets, i.e., unbounded Petri nets where each token carries a real-valued clock. Transition arcs are labeled with time intervals...
Numerical Analysis	Target: Title and Abstract	A fast volume integral equation solver with linear basis functions...for the accurate and efficient computation of the electromagnetic scattering from highly inhomogeneous ...Numerical experiments are conducted to study the accuracy and convergence properties of the proposed framework...
	Generated fake-text *(link to base)*	*on the performance of real time text and speech enhancement in mobile communication systems. in this paper, we present a comprehensive study on the effectiveness of a real - time text - and - speech enhancement technique for automatic speech recognition (asr) applications. we consider a scenario in which a mobile phone communicates with a user through a background noise - free background noise channel, while the phone continuously monitors the speech signal and provides feedback to the user about the quality of its speech enhancement...as a complement to an extensive performance evaluation in real - life applications...*
Information Theory	Base: Title and Abstract	On the performance of selection cooperation with imperfect channel estimation. In this paper, we investigate the performance of selection cooperation in the presence of imperfect channel estimation. In particular, we consider a cooperative scenario with multiple relays and amplifyand-forward protocol over frequency flat fading channels...outage probability and average capacity per bandwidth of the received signal in the presence of channel estimation errors. A simulation study...
Multimedia	Target: Title and Abstract	High quality low delay music coding in the opus codec. The IETF recently standardized the Opus codec as RFC6716. Opus targets a wide range of real-time Internet applications by combining a linear prediction coder with a transform coder. We describe the transform coder, with particular attention to the psychoacoustic knowledge built into the format. The result out-performs existing audio codecs that do not operate under real-time constraints.

Table B.14: More examples (Part-2) of target node, one base node, and one of the generated fake-text link to this base. The category (i.e., labels) of the base/target node shown in first column, generated injected node is unlabelled. Due to space limit, only show partial of the contents.

	Generated fake-text *(link to base)*	*uncertainty quantification in automated planning and verification of automated control systems. this paper presents an uncertainty quantification framework for automated control system verification. uncertainty quantification is carried out in three steps ... verification of the generated probabilistic model against real - world constraints. the uncertainty quantification problem is formulated as a mixed integer linear program (milp) and solved using a branch - and - bound approach. the results are applied to a case study of the automated control of a heating, ventilation and air conditioning (hvac) system in a residential building...*
Computational Engineering	Base: Title and Abstract	Heuristic optimization for automated distribution system planning in network integration studies. Network integration studies try to assess the impact of future developments, such as the increase of Renewable Energy Sources or the introduction of Smart Grid Technologies... This allows the estimation of the expected cost in massive probabilistic simulations of large numbers of real networks...
Numerical Analysis	Target: Title and Abstract	Numerical verification of affine systems with up to a billion dimensions. Affine systems reachability is the basis of many verification methods. With further computation, methods exist to reason about richer models with inputs, nonlinear differential equations, and hybrid dynamics... In this paper, we improve the scalability of affine systems verification... this direct approach requires an intractable amount of memory while using an intractable amount of computation time...
	Generated fake-text *(link to base)*	*reactive planning and control of autonomous vehicles in uncertain environments. this paper deals with the problem of reactively driving an autonomous vehicle in a uncertain environment. in particular, we assume that the vehicle is equipped with a collision avoidance system, and that the environment is uncertain... the proposed reactive strategy is able to perform reactively and reliably, while taking into consideration the uncertainties of both the environment and the vehicle dynamics.*
Machine Learning	Base: Title and Abstract	Neural networks trained with wifi traces to predict airport passenger behavior. The use of neural networks to predict airport passenger activity choices inside the terminal is presented in this paper. Three network architectures are proposed...A real-world case study exemplifies the application of these models, using anonymous WiFi traces collected at Bologna Airport to train the networks...
Systems and Control	Target: Title and Abstract	An unconditionally stable first order constraint solver for multibody systems. This article describes an absolutely stable, first-order constraint solverfor multi-rigid body systems that calculates (predicts) constraint forces for typical bilateral and unilateral constraints, contact constraints with friction, and many other constraint types...I assess the approach on some fundamental multibody dynamics problems.

C | Appendix: Supplementary Materials for Chapter 4

C.1 Additional details

C.1.1 Data collection details

Table C.1 shows the keywords we use to identify beginning of new reasoning paths to help segmenting reasoning trace into chunks.

Keywords for chunk segmentation
"wait", "double-check", "alternatively", "make sure", "another way", "verify", "to confirm"

Table C.1: Keywords we use for identifying reasoning path switch and segmenting reasoning trace into chunks.

For AIME, we use AIME_1983_2024 [1] for training and AIME_2025 [2] for testing. For MATH, we use the original training set and the 500-example test set released by HuggingFace [3]. For KnowLogic dataset, we randomly split the dataset into a training and test set by 80% and 20%, and collect probing data separately.

Table C.2 shows the statistics of collected chunks for each dataset. We use vLLM [109] for inference and set maximum output length to 30K. Examples whose model completion goes over

[1] https://huggingface.co/datasets/di-zhang-fdu/AIME_1983_2024
[2] https://huggingface.co/datasets/yentinglin/aime_2025
[3] https://huggingface.co/datasets/HuggingFaceH4/MATH-500

the maximum output length are discarded.

Reasoning Model	#Train Examples	#Test Examples	#Train Chunks	#Test Chunks	Avg. Chunk Len.	Positive Chunks (%)
	GSM8K					
R1-Distill-Llama-8B	1000	1317	7379	11228	328.0	70.97
R1-Distill-Llama-70B	998	1318	9030	6116	272.4	84.36
R1-Distill-Qwen-1.5B	995	1308	8599	11730	379.0	63.57
R1-Distill-Qwen-7B	1000	1316	5615	7568	302.8	75.87
R1-Distill-Qwen-32B	996	1317	4393	6381	293.1	84.25
	MATH					
R1-Distill-Llama-8B	1000	491	6259	3380	615.1	76.91
R1-Distill-Llama-70B	996	499	4865	2559	701.5	82.88
R1-Distill-Qwen-1.5B	988	495	7388	4089	996.7	68.46
R1-Distill-Qwen-7B	983	494	5062	2764	713.3	79.77
R1-Distill-Qwen-32B	991	497	4732	2460	678.5	84.40
	AIME					
R1-Distill-Llama-8B	922	30	7158	323	1652.0	35.24
R1-Distill-Llama-70B	923	30	5443	318	1528.0	50.78
R1-Distill-Qwen-1.5B	892	29	8358	314	1809.4	26.33
R1-Distill-Qwen-7B	922	29	5501	179	1841.8	42.50
R1-Distill-Qwen-32B	868	25	4181	104	1244.1	55.03
	KnowLogic					
R1-Distill-Llama-8B	986	320	7620	2596	1079.6	44.27
R1-Distill-Llama-70B	996	297	6529	2000	639.7	57.71
R1-Distill-Qwen-1.5B	762	245	6879	2036	1070.0	20.56
R1-Distill-Qwen-7B	938	306	7169	2430	1072.7	42.25
R1-Distill-Qwen-32B	979	315	6131	1827	818.8	57.40

Table C.2: Statistics for obtained probing dataset across task datasets and reasoning models. The inconsistency in training examples and test examples number comes from discard of examples with truncated model completion. The average chunk length is calculated by sampling 1000 chunks from each training dataset and measured by number of tokens. The positive chunk ratio is calculated based on the training set.

Figure C.1 is a visualization of chunk representations obtained for different datasets with R1-Distill-Llama-8B [44]. The domain difference between logical reasoning and mathematical problems is evident.

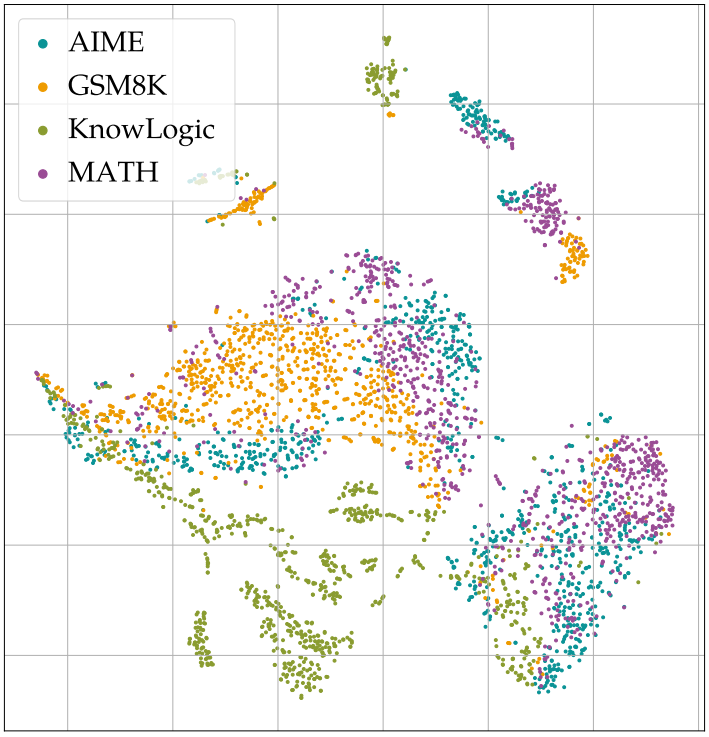


Figure C.1: T-SNE visualization of chunk representations for different datasets. 1000 chunks are randomly sampled from each training set and R1-Distill-Llama-8B is used to obtain the representation.

C.1.2 Prompts

Table C.3 shows the prompt we used to elicit reasoning trace from all reasoning models. Note that for Qwen models, the prompt we use is slightly different from its original prompt. We observe the performance on the benchmark does degrade a little but within a reasonable range. To ensure the extracted feature is on-policy, we also keep the same prompt when extracting representations for each reasoning chunk.

Table C.4 is the evaluation prompt we use for Gemini 2.0 Flash [63] for answer extraction and evaluation based on given reasoning chunks.

Inference Prompt
<BOS_TOKEN> <|User|> `{instruction}` Please reason step by step, and put your final answer within \boxed{}. <|Assistant|>

Table C.3: Prompt used for inference with reasoning models.

Evaluation Prompt

Given several chunks of a reasoning trace, along with a ground-truth answer, independently evaluate each chunk. If a chunk reaches a result at the end, return the intermediate result; otherwise, return None if the chunk does not contain an intermediate result (e.g., pure reflections).
Then, if an intermediate answer exists, compare it to the ground-truth answer. If the intermediate result in the chunk equals the ground-truth answer, return True; if the intermediate result in the chunk does not equal the ground-truth answer, return False; if no intermediate answer exists, return None.
Output in JSON format:
`[`
`  {"id": "1", "result": "6 + 9i" / None, "correctness": True / False / None},`
`  ...`
`]`
Input chunks: `{reasoning_trace}`
Ground-truth answer: `{answer}`

Table C.4: Prompt used for answer extraction and evaluation with Gemini 2.0 Flash.

C.1.3 Grid search

We perform grid search over hyparameters include learning rate, loss weight scaling factor α, weight decay for optimizer, and classifier hidden size d. The specific search range for each hyperparameter can be found in Table C.5, and the resulting optimal hyperparameter settings for each probing dataset are shown in Table C.6.

Hyperparameter	Search Space
Learning rate	1e-3, 1e-4, 1e-5
Scaling factor α	0.3, 0.5, 0.7, 0.9, 1.0, 1.5, 2.0, 3.0
Weight decay	0.001, 0.01, 0.1
Hidden size d	0, 16, 32

Table C.5: Hyperparameter search space for classifier training.

Model	Dataset	Learning rate	Loss weight α	Weight decay	Hidden size d
R1-Distill -Llama-8B	GSM8K	1e-4	3.0	0.1	16
	MATH	1e-5	2.0	0.001	0
	AIME	1e-5	0.3	0.1	0
	KnowLogic	1e-5	0.7	0.1	0
R1-Distill -Qwen-1.5B	GSM8K	1e-5	2.0	0.1	16
	MATH	1e-3	2.0	0.01	16
	AIME	1e-5	0.5	0.01	16
	KnowLogic	1e-4	0.3	0.001	0
R1-Distill -Qwen-7B	GSM8K	1e-4	3.0	0.1	0
	MATH	1e-4	3.0	0.1	0
	AIME	1e-3	0.9	0.1	0
	KnowLogic	1e-5	0.9	0.1	0
R1-Distill -Qwen-32B	GSM8K	1e-3	3.0	0.001	16
	MATH	1e-4	2.0	0.1	0
	AIME	1e-5	1.0	0.01	16
	KnowLogic	1e-5	0.9	0.1	0
R1-Distill -Llama-70B	GSM8K	1e-4	2.0	0.001	0
	MATH	1e-4	3.0	0.001	0
	AIME	1e-4	2.0	0.001	0
	KnowLogic	1e-3	1.0	0.01	32
QwQ-32B	GSM8K	1e-4	3.0	0.1	0
	MATH	1e-3	2.0	0.001	16
	AIME	1e-3	3.0	0.01	16
	KnowLogic	1e-4	1.5	0.1	0

Table C.6: Results of grid search across reasoning models and datasets.

C.1.4 Further results

Table C.7 and Table C.8 show in-distribution probing performance measured by accuracy, precision, recall, and macro F1 across reasoning models and datasets.

Table C.9 to Table C.13 show out-of-distribution probing performance trained and test on rep-

resentations from R1-Distill-Qwen-1.5B, R1-Distill-Qwen-7B, R1-Distill-Qwen-32B, and QwQ-32B, respectively.

Reasoning Model	GSM8K				MATH			
	Accuracy	Precision	Recall	Macro F1	Accuracy	Precision	Recall	Macro F1
R1-Distill-Llama-8B	0.77	0.85	0.82	0.73	0.80	0.84	0.88	0.75
R1-Distill-Llama-70B	0.91	0.92	0.97	0.82	0.89	0.92	0.93	0.83
R1-Distill-Qwen-1.5B	0.76	0.81	0.81	0.74	0.84	0.84	0.88	0.83
R1-Distill-Qwen-7B	0.84	0.88	0.92	0.77	0.87	0.89	0.94	0.82
R1-Distill-Qwen-32B	0.89	0.91	0.95	0.79	0.89	0.94	0.92	0.85
QwQ-32B	0.83	0.83	0.99	0.49	0.87	0.95	0.89	0.79

Table C.7: Accuracy, precision, recall, and macro F1 score for probes trained and test on GSM8K and MATH datasets in in-distribution setting.

Reasoning Model	AIME				KnowLogic			
	Accuracy	Precision	Recall	Macro F1	Accuracy	Precision	Recall	Macro F1
R1-Distill-Llama-8B	0.85	0.37	0.38	0.64	0.62	0.62	0.41	0.60
R1-Distill-Llama-70B	0.75	0.80	0.54	0.73	0.67	0.79	0.62	0.67
R1-Distill-Qwen-1.5B	0.83	0.45	0.62	0.71	0.72	0.23	0.53	0.42
R1-Distill-Qwen-7B	0.78	0.65	0.64	0.74	0.69	0.60	0.58	0.67
R1-Distill-Qwen-32B	0.91	0.88	0.96	0.91	0.70	0.80	0.67	0.69
QwQ-32B	0.82	0.84	0.85	0.82	0.78	0.82	0.88	0.74

Table C.8: Accuracy, precision, recall, and macro F1 score for probes trained and test on AIME and KnowLogic datasets in in-distribution setting.

Training Data	GSM8K		MATH		AIME		KnowLogic	
	AUC ↑	ECE ↓	AUC ↑	ECE ↓	AUC ↑	ECE ↓	AUC ↑	ECE ↓
GSM8K	0.82	0.04	0.90 (+0.06)	0.07 (+0.04)	0.75 (-0.05)	0.14 (+0.04)	0.62 (-0.05)	0.08 (+0.01)
MATH	0.82 (-0.01)	0.10 (+0.06)	0.84	0.03	0.84 (+0.04)	0.18 (+0.08)	0.63 (-0.04)	0.14 (+0.08)
KnowLogic	0.67 (-0.16)	0.36 (+0.32)	0.73 (-0.11)	0.34 (+0.32)	0.68 (-0.12)	0.05 (-0.05)	0.67	0.07

Table C.9: ROC-AUC scores and ECE of trained probes on out-of-distribution test set. The numbers in red and green denote performance decrease and increase relative to the probe trained on in-distribution training set, respectively. R1-Distill-Qwen-1.5B is used as the reasoning model.

Training Data	GSM8K		MATH		AIME		KnowLogic	
	AUC ↑	ECE ↓	AUC ↑	ECE ↓	AUC ↑	ECE ↓	AUC ↑	ECE ↓
GSM8K	0.82	0.04	0.86 (+0.02)	0.06 (+0.03)	0.76 (-0.04)	0.15 (+0.05)	0.60 (-0.07)	0.17 (+0.10)
MATH	0.86 (+0.04)	0.06 (+0.02)	0.84	0.03	0.73 (-0.07)	0.18 (+0.08)	0.68 (+0.02)	0.17 (+0.10)
KnowLogic	0.81 (-0.02)	0.07 (+0.03)	0.83 (-0.01)	0.10 (+0.07)	0.72 (-0.08)	0.16 (+0.06)	0.67	0.07

Table C.10: ROC-AUC scores and ECE of trained probes on out-of-distribution test set. The numbers in red and green denote performance decrease and increase relative to the probe trained on in-distribution training set, respectively. R1-Distill-Qwen-7B is used as the reasoning model.

Training Data	GSM8K		MATH		AIME		KnowLogic	
	AUC ↑	ECE ↓	AUC ↑	ECE ↓	AUC ↑	ECE ↓	AUC ↑	ECE ↓
GSM8K	0.82	0.04	0.87 (+0.03)	0.04 (+0.01)	0.98 (+0.17)	0.17 (+0.07)	0.73 (+0.06)	0.06 (-0.01)
MATH	0.89 (+0.06)	0.03 (-0.01)	0.84	0.03	0.97 (+0.16)	0.10 (+0.00)	0.72 (+0.05)	0.15 (+0.08)
KnowLogic	0.83 (+0.00)	0.09 (+0.05)	0.89 (+0.05)	0.10 (+0.07)	0.91 (+0.10)	0.22 (+0.12)	0.67	0.07

Table C.11: ROC-AUC scores and ECE of trained probes on out-of-distribution test set. The numbers in red and green denote performance decrease and increase relative to the probe trained on in-distribution training set, respectively. R1-Distill-Qwen-32B is used as the reasoning model.

Training Data	GSM8K		MATH		AIME		KnowLogic	
	AUC ↑	ECE ↓	AUC ↑	ECE ↓	AUC ↑	ECE ↓	AUC ↑	ECE ↓
GSM8K	0.82	0.04	0.88 (+0.04)	0.09 (+0.06)	0.71 (-0.09)	0.17 (+0.07)	0.62 (-0.04)	0.25 (+0.18)
MATH	0.87 (+0.05)	0.06 (+0.02)	0.84	0.03	0.75 (-0.05)	0.16 (+0.06)	0.73 (+0.06)	0.20 (+0.13)
KnowLogic	0.84 (+0.01)	0.10 (+0.06)	0.87 (+0.03)	0.13 (+0.10)	0.70 (-0.10)	0.12 (+0.02)	0.67	0.07

Table C.12: ROC-AUC scores and ECE of trained probes on out-of-distribution test set. The numbers in red and green denote performance decrease and increase relative to the probe trained on in-distribution training set, respectively. R1-Distill-Llama-70B is used as the reasoning model.

Training Data	GSM8K		MATH		AIME		KnowLogic	
	AUC ↑	ECE ↓	AUC ↑	ECE ↓	AUC ↑	ECE ↓	AUC ↑	ECE ↓
GSM8K	0.82	0.04	0.74 (-0.10)	0.14 (+0.12)	0.73 (-0.07)	0.23 (+0.13)	0.61 (-0.06)	0.29 (+0.23)
MATH	0.55 (-0.27)	0.22 (+0.18)	0.84	0.03	0.87 (+0.07)	0.07 (-0.03)	0.76 (+0.09)	0.11 (+0.04)
KnowLogic	0.61 (-0.22)	0.14 (+0.11)	0.81 (-0.03)	0.05 (+0.02)	0.84 (+0.04)	0.07 (-0.03)	0.67	0.07

Table C.13: ROC-AUC scores and ECE of trained probes on out-of-distribution test set. The numbers in red and green denote performance decrease and increase relative to the probe trained on in-distribution training set, respectively. QwQ-32B is used as the reasoning model.

Bibliography

[1] Ossama Abdel-Hamid et al. "Convolutional neural networks for speech recognition". In: *IEEE/ACM Transactions on audio, speech, and language processing* 22.10 (2014), pp. 1533–1545.

[2] Akshay Agrawal et al. "A rewriting system for convex optimization problems". In: *Journal of Control and Decision* 5.1 (2018), pp. 42–60.

[3] Matthew Aldridge, Oliver Johnson, and Jonathan Scarlett. "Group Testing: An Information Theory Perspective". In: *Foundations and Trends® in Communications and Information Theory* 15.3-4 (2019), pp. 196–392. ISSN: 1567-2190. DOI: 10.1561/0100000099.

[4] Joshua D Angrist and Jörn-Steffen Pischke. *Mostly harmless econometrics: An empiricist's companion*. Princeton university press, 2008.

[5] Amos Azaria and Tom Mitchell. *The Internal State of an LLM Knows When It's Lying*. 2023.

[6] Woonhyuk Baek. *wbaek/torchskeleton*. Online; accessed 2021-02-07.

[7] Yuntao Bai et al. *Constitutional AI: Harmlessness from AI Feedback*. 2022.

[8] B. J. Balakumar. *glmnet_python*.

[9] Nathalie Baracaldo et al. "Mitigating poisoning attacks on machine learning models: A data provenance based approach". In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. 2017, pp. 103–110.

[10] Marco Barreno et al. “The security of machine learning”. In: *Machine Learning* 81.2 (2010), pp. 121–148.

[11] Samyadeep Basu, Philip Pope, and Soheil Feizi. “Influence functions in deep learning are fragile”. In: *arXiv preprint arXiv:2006.14651* (2020).

[12] Yonatan Belinkov. *Probing Classifiers: Promises, Shortcomings, and Advances*. 2021.

[13] Iz Beltagy, Kyle Lo, and Arman Cohan. “SciBERT: A Pretrained Language Model for Scientific Text”. In: *arXiv 1903.10676* (2019).

[14] Yoshua Bengio. “Gradient-Based Optimization of Hyperparameters”. In: *Neural Computation* 12.8 (2000), pp. 1889–1900. DOI: 10.1162/089976600300015187.

[15] Aleksandar Bojchevski and Stephan Günnemann. “Adversarial attacks on node embeddings via graph poisoning”. In: *International Conference on Machine Learning (ICML)*. 2019.

[16] Howard D Bondell and Brian J Reich. “Simultaneous factor selection and collapsing levels in ANOVA”. In: *Biometrics* 65.1 (2009), pp. 169–177.

[17] Leo Breiman. “Random Forests”. In: *Mach. Learn.* 45.1 (Oct. 2001), pp. 5–32. ISSN: 0885-6125. DOI: 10.1023/A:1010933404324.

[18] Glenn W Brier. “Verification of forecasts expressed in terms of probability”. In: *Monthly weather review* 78.1 (1950), pp. 1–3.

[19] Tom Brown et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[20] Collin Burns et al. *Discovering Latent Knowledge in Language Models Without Supervision*. 2024.

[21] Emmanuel Candes et al. “Panning for gold: Model-X knockoffs for high-dimensional controlled variable selection”. In: *arXiv preprint arXiv:1610.02351* (2016).

[22] Emmanuel J Candès et al. “Compressive sampling”. In: *Proceedings of the international congress of mathematicians*. 2006.

[23] Aleksandar Chakarov et al. “Debugging machine learning tasks”. In: *arXiv:1603.07292* (2016).

[24] Heng Chang et al. “A restricted black-box adversarial framework towards attacking graph embedding models”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 3389–3396.

[25] Bryant Chen et al. “Detecting backdoor attacks on deep neural networks by activation clustering”. In: *arXiv preprint arXiv:1811.03728* (2018).

[26] Jinyin Chen et al. “Fast gradient attack on network embedding”. In: *arXiv:1809.02797* (2018).

[27] Jinyin Chen et al. “MGA: momentum gradient attack on network”. In: *IEEE Transactions on Computational Social Systems* 8.1 (2020), pp. 99–109.

[28] Xingyu Chen et al. *Do NOT Think That Much for 2+3=? On the Overthinking of o1-Like LLMs*. 2025.

[29] Xinyun Chen et al. “Targeted backdoor attacks on deep learning systems using data poisoning”. In: *arXiv preprint arXiv:1712.05526* (2017).

[30] Yongqiang Chen et al. “Understanding and improving graph injection attack by promoting unnoticeability”. In: *International Conference on Learning Representations (ICLR)*. 2022.

[31] Edward Chou et al. “Sentinet: Detecting physical attacks against deep learning systems”. In: (2018).

[32] Xu Chu, Ihab F Ilyas, and Paolo Papotti. “Discovering denial constraints”. In: *Proceedings of the VLDB Endowment* 6.13 (2013), pp. 1498–1509.

[33] Xu Chu, Ihab F Ilyas, and Paolo Papotti. "Holistic data cleaning: Putting violations into context". In: *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE. 2013, pp. 458–469.

[34] Karl Cobbe et al. *Training Verifiers to Solve Math Word Problems*. 2021.

[35] Gregory Cohen et al. "EMNIST: Extending MNIST to handwritten letters". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 2921–2926.

[36] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. "Certified Adversarial Robustness via Randomized Smoothing". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 1310–1320.

[37] Ian Covert and Su-In Lee. "Improving KernelSHAP: Practical Shapley Value Estimation Using Linear Regression". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2021, pp. 3457–3465.

[38] Antonia Creswell and Murray Shanahan. *Faithful Reasoning Using Large Language Models*. 2022.

[39] Enyan Dai et al. "Unnoticeable Backdoor Attacks on Graph Neural Networks". In: (2023).

[40] Hanjun Dai et al. "Adversarial attack on graph structured data". In: *International conference on machine learning (ICML)*. 2018.

[41] Mehul Damani et al. *Learning How Hard to Think: Input-Adaptive Allocation of LM Computation*. 2024.

[42] Tirthankar Dasgupta, Natesh S Pillai, and Donald B Rubin. "Causal inference from 2 K factorial designs by using potential outcomes". In: *Journal of the Royal Statistical Society: Series B: Statistical Methodology* (2015), pp. 727–753.

[43] DeepMind. *AI solves IMO problems at silver medal level*. Accessed: 2025-03-24. 2024.

[44] DeepSeek-AI. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. 2025.

[45] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019.

[46] Steven Diamond and Stephen Boyd. "CVXPY: A Python-embedded modeling language for convex optimization". In: *Journal of Machine Learning Research* 17.83 (2016), pp. 1–5.

[47] Bao Gia Doan, Ehsan Abbasnejad, and Damith C Ranasinghe. "Februus: Input purification defense against trojan attacks on deep neural network systems". In: *Annual Computer Security Applications Conference*. 2020, pp. 897–912.

[48] Mohamad Dolatshah et al. "Cleaning Crowdsourced Labels Using Oracles for Statistical Classification". In: *Proc. VLDB Endow.* 12.4 (Dec. 2018), pp. 376–389. ISSN: 2150-8097. DOI: 10.14778/3297753.3297758.

[49] Alexandre Duval and Fragkiskos D Malliaros. "Graphsvx: Shapley value explanations for graph neural networks". In: *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part II 21*. Springer. 2021, pp. 302–318.

[50] Naoki Egami and Kosuke Imai. "Causal interaction in factorial experiments: Application to conjoint analysis". In: *Journal of the American Statistical Association* (2018).

[51] Negin Entezari et al. "All you need is low (rank) defending against adversarial attacks on graphs". In: *Proceedings of the 13th International Conference on Web Search and Data Mining*. 2020, pp. 169–177.

[52] Wenqi Fan et al. "Jointly Attacking Graph Neural Network and its Explanations". In: *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. 2023, pp. 654–667. DOI: 10.1109/ICDE55515.2023.00056.

[53] Junfeng Fang et al. “Evaluating Post-hoc Explanations for Graph Neural Networks via Robustness Analysis”. In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023.

[54] Vitaly Feldman and Chiyuan Zhang. “What neural networks memorize and why: Discovering the long tail via influence estimation”. In: *arXiv preprint arXiv:2008.03703* (2020).

[55] Jiazhan Feng et al. *Language Models can be Logical Solvers*. 2023.

[56] Mohamed Amine Ferrag, Norbert Tihanyi, and Merouane Debbah. *From LLM Reasoning to Autonomous AI Agents: A Comprehensive Review*. 2025.

[57] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. 2017.

[58] Christopher Frye, Colin Rowat, and Ilya Feige. “Asymmetric Shapley values: incorporating causal knowledge into model-agnostic explainability”. In: *Advances in Neural Information Processing Systems* (2020).

[59] Yichao Fu et al. *Efficiently Serving LLM Reasoning Programs with Certaindex*. 2024.

[60] Yansong Gao et al. “Strip: A defence against trojan attacks on deep neural networks”. In: *Proceedings of the 35th Annual Computer Security Applications Conference*. 2019, pp. 113–125.

[61] Simon Geisler et al. “Robustness of Graph Neural Networks at Scale”. In: *Neural Information Processing Systems (NeurIPS 2021)*. 2021.

[62] Simon Geisler et al. “Robustness of graph neural networks at scale”. In: *Advances in Neural Information Processing Systems*. 2021.

[63] Gemini-Team. *Gemini: A Family of Highly Capable Multimodal Models*. 2024.

[64] Amirata Ghorbani, Michael Kim, and James Zou. "A distributional framework for data valuation". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3535–3544.

[65] Amirata Ghorbani and James Zou. "Data shapley: Equitable valuation of data for machine learning". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2242–2251.

[66] Ben Goertzel. "Artificial general intelligence: concept, state of the art, and future prospects". In: *Journal of Artificial General Intelligence* 5.1 (2014), p. 1.

[67] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *ICLR*. 2015.

[68] Aaron Grattafiori et al. *The Llama 3 Herd of Models*. 2024.

[69] Google Research Group. "PaLM: scaling language modeling with pathways". In: *J. Mach. Learn. Res.* 24.1 (Jan. 2023). ISSN: 1532-4435.

[70] Jens Hainmueller, Daniel J Hopkins, and Teppei Yamamoto. "Causal inference in conjoint analysis: Understanding multidimensional choices via stated preference experiments". In: *Political analysis* 22.1 (2014), pp. 1–30.

[71] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs". In: *Advances in neural information processing systems* 30 (2017).

[72] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *Neural Information Processing Systems (NIPS 2017)*. 2017.

[73] Zayd Hammoudeh and Daniel Lowd. "Simple, Attack-Agnostic Defense Against Targeted Training Set Attacks Using Cosine Similarity". In: ().

[74] Chris Harris, Richard Pymar, and Colin Rowat. "Joint Shapley values: a measure of joint feature importance". In: *International Conference on Learning Representations*. 2022.

[75] Trevor Hastie, Junyang Qian, and Kenneth Tay. *An Introduction to glmnet*. 2016.

[76] Jonathan Hayase et al. “SPECTRE: Defending Against Backdoor Attacks Using Robust Statistics”. In: *arXiv preprint arXiv:2104.11315* (2021).

[77] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015.

[78] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[79] M.A. Hearst et al. “Support vector machines”. In: *IEEE Intelligent Systems and their Applications* 13.4 (1998), pp. 18–28. DOI: 10.1109/5254.708428.

[80] Joseph M Hellerstein. “Quantitative data cleaning for large databases”. In: *United Nations Economic Commission for Europe (UNECE)* 25 (2008).

[81] Dan Hendrycks et al. *Measuring Mathematical Problem Solving With the MATH Dataset*. 2021.

[82] Weihua Hu et al. “Open Graph Benchmark: Datasets for Machine Learning on Graphs”. In: *arXiv 2005.00687* (2021).

[83] Weihua Hu et al. “Open graph benchmark: datasets for machine learning on graphs”. In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS ’20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546.

[84] Qiang Huang et al. “Graphlime: Local interpretable model explanations for graph neural networks”. In: *IEEE Transactions on Knowledge and Data Engineering* (2022).

[85] W Ronny Huang et al. “Metapoison: Practical general-purpose clean-label data poisoning”. In: *Advances in Neural Information Processing Systems* (2020).

[86] G.Tusnady I. Csiszar. “Information Geometry and Alternating Minimization Procedures”. In: (1984).

[87] Andrew Ilyas et al. “Datamodels: Predicting predictions from training data”. In: *arXiv preprint arXiv:2202.00622* (2022).

[88] Guido W Imbens and Donald B Rubin. *Causal inference in statistics, social, and biomedical sciences*. Cambridge University Press, 2015.

[89] Matthew Jagielski et al. "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning". In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, pp. 19–35.

[90] Neil Jethani et al. "FastSHAP: Real-Time Shapley Value Estimation". In: *arXiv e-prints* (2021), arXiv–2107.

[91] Ruoxi Jia. *Group Testing Implementation*. Oct. 2021.

[92] Ruoxi Jia et al. "Towards efficient data valuation based on the shapley value". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR. 2019, pp. 1167–1176.

[93] Chao Jiang et al. "Camouflaged poisoning attack on graph neural networks". In: *Proceedings of the 2022 International Conference on Multimedia Retrieval*. 2022, pp. 451–461.

[94] Wei Jin et al. "Adversarial Attacks and Defenses on Graphs: A Review and Empirical Study". In: *CoRR* abs/2003.00653 (2020).

[95] Wei Jin et al. "Graph Structure Learning for Robust Graph Neural Networks". In: *Knowledge Discovery and Data Mining (KDD)*. 2020.

[96] Mingxuan Ju et al. "Let Graph be the Go Board: Gradient-free Node Injection Attack for Graph Neural Networks via Reinforcement Learning". In: *Thirty-Seventh AAAI Conference on Artificial Intelligence*. 2023.

[97] Joseph DY Kang, Joseph L Schafer, et al. "Demystifying double robustness: A comparison of alternative strategies for estimating a population mean from incomplete data". In: *Statistical science* 22.4 (2007), pp. 523–539.

[98] Sanyam Kapoor et al. *Large Language Models Must Be Taught to Know What They Don't Know*. 2024.

[99] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[100] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017.

[101] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *International Conference on Learning Representations*. 2017.

[102] Pang Wei Koh and Percy Liang. “Understanding black-box predictions via influence functions”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1885–1894.

[103] Pang Wei Koh, Jacob Steinhardt, and Percy Liang. “Stronger data poisoning attacks break data sanitization defenses”. In: *arXiv preprint arXiv:1811.00741* (2018).

[104] Pang Wei Koh et al. “On the accuracy of influence functions for measuring group effects”. In: *arXiv preprint arXiv:1905.13289* (2019).

[105] Sanjay Krishnan et al. “Boostclean: Automated error detection and repair for machine learning”. In: *arXiv preprint arXiv:1711.01299* (2017).

[106] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).

[107] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet classification with deep convolutional neural networks”. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*. NIPS’12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.

[108] Kuangliu. *kuangliu/pytorch-cifar*. Online; accessed 2021-02-07.

[109] Woosuk Kwon et al. *Efficient Memory Management for Large Language Model Serving with PagedAttention*. 2023.

[110] Yongchan Kwon and James Zou. "Beta Shapley: a Unified and Noise-reduced Data Valuation Framework for Machine Learning". In: *International Conference on Artificial Intelligence and Statistics*. 2022.

[111] Long Hei Matthew Lam, Ramya Keerthy Thatikonda, and Ehsan Shareghi. *A Closer Look at Logical Reasoning with LLMs: The Choice of Tool Matters*. 2024.

[112] Guillaume Lecué, Shahar Mendelson, et al. "Regularization and the small-ball method i: sparse recovery". In: *The Annals of Statistics* 46.2 (2018), pp. 611–641.

[113] Guillaume Lecué and Shahar Mendelson. "Regularization and the small-ball method II: complexity dependent error rates". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 5356–5403.

[114] Mathias Lecuyer et al. "Sunlight: Fine-Grained Targeting Detection at Scale with Statistical Confidence". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. 2015.

[115] Patrick Lewis et al. "Retrieval-augmented generation for knowledge-intensive nlp tasks". In: *Advances in neural information processing systems* 33 (2020), pp. 9459–9474.

[116] Bai Li et al. "Certified adversarial robustness with additive noise". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[117] Xuhong Li et al. "Interpretable deep learning: Interpretation, interpretability, trustworthiness, and beyond". In: *Knowledge and Information Systems* 64.12 (2022), pp. 3197–3234.

[118] Yiwei Li et al. *Escape Sky-high Cost: Early-stopping Self-Consistency for Multi-step Reasoning*. 2024.

[119] Hunter Lightman et al. *Let's Verify Step by Step*. 2023.

[120] Stephanie Lin, Jacob Hilton, and Owain Evans. *Teaching Models to Express Their Uncertainty in Words*. 2022.

[121] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. "Explainable ai: A review of machine learning interpretability methods". In: *Entropy* 23.1 (2020), p. 18.

[122] Zhan Ling et al. *Deductive Verification of Chain-of-Thought Reasoning*. 2023.

[123] Stan Lipovetsky and Michael Conklin. "Analysis of regression in game theory approach". In: *Applied Stochastic Models in Business and Industry* (2001).

[124] Hanmeng Liu et al. *Logical Reasoning in Large Language Models: A Survey*. 2025.

[125] Xuanqing Liu et al. "A unified framework for data poisoning attack to graph-based semi-supervised learning". In: (2019).

[126] Scott M Lundberg and Su-In Lee. "A unified approach to interpreting model predictions". In: *Proceedings of the 31st international conference on neural information processing systems*. 2017, pp. 4768–4777.

[127] Dongsheng Luo et al. "Parameterized explainer for graph neural network". In: *Advances in neural information processing systems* 33 (2020), pp. 19620–19631.

[128] Jiaqi Ma, Shuangrui Ding, and Qiaozhu Mei. "Towards more practical adversarial attacks on graph neural networks". In: *Advances in neural information processing systems (NeurIPS)*. 2020.

[129] Aman Madaan et al. *Self-Refine: Iterative Refinement with Self-Feedback*. 2023.

[130] Aleksander Madry et al. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *ICLR*. 2018.

[131] Sasan Maleki et al. *Bounding the Estimation Error of Sampling-based Shapley Value Approximation*. 2014.

[132] Jonathan I Maletic and Andrian Marcus. “Data Cleansing: Beyond Integrity Analysis.” In: *Iq*. Citeseer. 2000, pp. 200–209.

[133] Rohin Manvi, Anikait Singh, and Stefano Ermon. *Adaptive Inference-Time Compute: LLMs Can Predict if They Can Do Better, Even Mid-Generation*. 2024.

[134] Guilherme Brandão Martins, João Paulo Papa, and Hojjat Adeli. “Deep learning techniques for recommender systems based on collaborative filtering”. In: *Expert Systems* 37.6 (2020), e12647.

[135] Sabrina J. Mielke et al. “Reducing Conversational Agents’ Overconfidence Through Linguistic Calibration”. In: *Transactions of the Association for Computational Linguistics* 10 (2022). Ed. by Brian Roark and Ani Nenkova, pp. 857–872. DOI: 10.1162/tacl_a_00494.

[136] Rory Mitchell et al. “Sampling permutations for Shapley value estimation”. In: (2022).

[137] Felix Mujkanovic et al. “Are Defenses for Graph Neural Networks Robust?” In: *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*. 2022.

[138] Tergel Munkhbat et al. *Self-Training Elicits Concise Reasoning in Large Language Models*. 2025.

[139] Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. “Obtaining well calibrated probabilities using bayesian binning”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 29. 1. 2015.

[140] Ali Bou Nassif et al. “Speech recognition using deep neural networks: A systematic review”. In: *IEEE access* 7 (2019), pp. 19143–19165.

[141] Toan Nguyen Thanh et al. “Poisoning GNN-based recommender systems with generative surrogate-based attacks”. In: *ACM Transactions on Information Systems* 41.3 (2023), pp. 1–24.

[142] Jianmo Ni, Jiacheng Li, and Julian McAuley. "Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 188–197. DOI: 10.18653/v1/D19-1018.

[143] OpenAI. *OpenAI o1 System Card*. 2024.

[144] Debjit Paul et al. *REFINER: Reasoning Feedback on Intermediate Representations*. 2024.

[145] Edouard Pauwels. *Lecture notes: Statistics, optimization and algorithms in high dimension*. 2020.

[146] Bezalel Peleg and Peter Sudhölter. *Introduction to the theory of cooperative games*. Vol. 34. Springer Science & Business Media, 2007.

[147] Neehar Peri et al. "Deep k-NN defense against clean-label data poisoning attacks". In: *European Conference on Computer Vision*. Springer. 2020, pp. 55–70.

[148] Justin B. Post and Howard D. Bondell. "Factor Selection and Structural Identification in the Interaction ANOVA Model". en. In: *Biometrics* 69.1 (2013), pp. 70–79. ISSN: 1541-0420. DOI: https://doi.org/10.1111/j.1541-0420.2012.01810.x.

[149] Garima Pruthi et al. "Estimating Training Data Influence by Tracing Gradient Descent". In: *Advances in Neural Information Processing Systems* 33 (2020).

[150] PyTorch. *pytorch/examples*. en. Online; accessed 2021-02-07.

[151] PyTorch. *pytorch/torchvision/resnet*. en. Online; accessed 2021-03-30.

[152] Holger Rauhut. "Compressive sensing and structured random matrices". In: *Theoretical foundations and numerical methods for sparse recovery* 9.1 (2010), p. 92.

[153] Omar Rivasplata. "Subgaussian random variables: An expository note". en. In: (), p. 11.

[154] Olga Russakovsky et al. “Imagenet large scale visual recognition challenge”. In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[155] Ali Shafahi et al. “Poison frogs! targeted clean-label poisoning attacks on neural networks”. In: *arXiv preprint arXiv:1804.00792* (2018).

[156] Ali Shafahi et al. “Poison frogs! targeted clean-label poisoning attacks on neural networks”. In: *Advances in neural information processing systems* 31 (2018).

[157] L. S Shapley. “A value for n-person games”. In: *Contributions to the Theory of Games* 2.28 (1953), pp. 307–317.

[158] Shiqi Shen, Shruti Tople, and Prateek Saxena. “Auror: Defending against poisoning attacks in collaborative deep learning systems”. In: *Proceedings of the 32nd Annual Conference on Computer Security Applications*. 2016, pp. 508–519.

[159] Noah Shinn et al. *Reflexion: Language Agents with Verbal Reinforcement Learning*. 2023.

[160] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015.

[161] Yang Sui et al. *Stop Overthinking: A Survey on Efficient Reasoning for Large Language Models*. 2025.

[162] Yiwei Sun et al. “Adversarial attacks on graph neural networks via node injections: A hierarchical reinforcement learning approach”. In: *Proceedings of the Web Conference (WWW)*. 2020.

[163] Di Tang et al. “Demon in the variant: Statistical analysis of dnns for robust backdoor contamination detection”. In: *30th {USENIX} Security Symposium ({USENIX} Security 21)*. 2021.

[164] Shuchang Tao et al. *Adversarial Camouflage for Node Injection Attack on Graphs*. 2023.

[165] Shuchang Tao et al. “Adversarial Immunization for Certifiable Robustness on Graphs”. In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. WSDM ’21. Virtual Event, Israel: Association for Computing Machinery, 2021, pp. 698–706. ISBN: 9781450382977. DOI: 10.1145/3437963.3441782.

[166] Shuchang Tao et al. “Graph Adversarial Immunization for Certifiable Robustness”. In: *IEEE Transactions on Knowledge and Data Engineering* (2023).

[167] Shuchang Tao et al. “Rethinking the Imperceptibility of Node Injection Attack on Graphs”. In: *arXiv 2208.01819* ().

[168] Shuchang Tao et al. “Single node injection attack against graph neural networks”. In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2021, pp. 1794–1803.

[169] Qwen Team. *QwQ-32B: Embracing the Power of Reinforcement Learning*. Mar. 2025.

[170] Qiaoying Teng et al. “A survey on the interpretability of deep learning in medical diagnosis”. In: *Multimedia Systems* 28.6 (2022), pp. 2335–2355.

[171] Katherine Tian et al. *Just Ask for Calibration: Strategies for Eliciting Calibrated Confidence Scores from Language Models Fine-Tuned with Human Feedback*. 2023.

[172] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023.

[173] Hugo Touvron et al. “Resmlp: Feedforward networks for image classification with data-efficient training”. In: *IEEE transactions on pattern analysis and machine intelligence* 45.4 (2022), pp. 5314–5321.

[174] Brandon Tran, Jerry Li, and Aleksander Madry. “Spectral signatures in backdoor attacks”. In: *arXiv preprint arXiv:1811.00636* (2018).

[175] Ben Trevett. *bentrevett/pytorch-sentiment-analysis/*. en. Online; accessed 2021-03-10.

[176] Sakshi Udeshi et al. “Model agnostic defence against backdoor attacks in machine learning”. In: *arXiv preprint arXiv:1908.02203* (2019).

[177] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[178] Akshaj Kumar Veldanda et al. “NNoculation: Broad spectrum and targeted treatment of backdoored DNNs”. In: *arXiv preprint arXiv:2002.08313* (2020).

[179] Petar Veličković et al. “Graph Attention Networks”. In: *International Conference on Learning Representations*. 2018.

[180] Binghui Wang and Neil Zhenqiang Gong. “Attacking graph-based classification via manipulating the graph structure”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019.

[181] Binghui Wang, Youqi Li, and Pan Zhou. “Bandits for Structure Perturbation-based Black-box Attacks to Graph Neural Networks with Theoretical Guarantees”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 13379–13387.

[182] Binghui Wang et al. “Certified Robustness of Graph Neural Networks against Adversarial Structural Perturbation”. In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. KDD ’21. Virtual Event, Singapore: Association for Computing Machinery, 2021, pp. 1645–1653. ISBN: 9781450383325. DOI: 10.1145/3447548.3467295.

[183] Bolun Wang et al. “Neural cleanse: Identifying and mitigating backdoor attacks in neural networks”. In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 707–723.

[184] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. “Collaborative deep learning for recommender systems”. In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2015, pp. 1235–1244.

[185] Jihong Wang et al. “Scalable attack on graph data by injecting vicious nodes”. In: *Data Mining and Knowledge Discovery* 34 (2020), pp. 1363–1389.

[186] Jihong Wang et al. “Scalable attack on graph data by injecting vicious nodes”. In: *Data Min. Knowl. Discov.* 34.5 (Sept. 2020), pp. 1363–1389. ISSN: 1384-5810. DOI: 10.1007/s10618-020-00696-7.

[187] Qineng Wang et al. “Rethinking the Bounds of LLM Reasoning: Are Multi-Agent Discussions the Key?” In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Lun-Wei Ku, Andre Martins, and Vivek Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024. DOI: 10.18653/v1/2024.acl-long.331.

[188] Xiaoyun Wang et al. “Attack graph convolutional networks by adding fake nodes”. In: *arXiv preprint arXiv:1810.10751* (2018).

[189] Xinglin Wang et al. *Make Every Penny Count: Difficulty-Adaptive Self-Consistency for Cost-Efficient Reasoning*. 2025.

[190] M. Waniek et al. “Hiding individuals and communities in a social network”. In: *Nature Human Behaviour* 2 (2018).

[191] Jason Wei et al. “Chain-of-thought prompting elicits reasoning in large language models”. In: *Advances in neural information processing systems* 35 (2022), pp. 24824–24837.

[192] Yixuan Weng et al. *Large Language Models are Better Reasoners with Self-Verification*. 2023.

[193] WikipediaContributors. *Group testing*. en. Online; accessed 2021-02-19. Jan. 2021.

[194] Brian Williamson and Jean Feng. “Efficient nonparametric statistical inference on population feature importance using Shapley values”. In: *International Conference on Machine Learning*. 2020.

[195] Huijun Wu et al. “Adversarial examples on graph data: Deep insights into attack and defense”. In: *International Joint Conference on Artificial Intelligence*. 2019.

[196] Weiyuan Wu et al. “Complaint-driven training data debugging for query 2.0”. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 1317–1334.

[197] Zhaohan Xi et al. “Graph Backdoor.” In: *USENIX Security Symposium*. 2021, pp. 1523–1540.

[198] zaishuo xia et al. “GNNCert: Deterministic Certification of Graph Neural Networks against Adversarial Perturbations”. In: *The Twelfth International Conference on Learning Representations*. 2024.

[199] Miao Xiong et al. *Can LLMs Express Their Uncertainty? An Empirical Evaluation of Confidence Elicitation in LLMs*. 2024.

[200] Kaidi Xu et al. “Topology attack and defense for graph neural networks: An optimization perspective”. In: *International Joint Conference on Artificial Intelligence*. 2019.

[201] Keyulu Xu et al. “How Powerful are Graph Neural Networks?” In: *International Conference on Learning Representations*. 2019.

[202] Mayi Xu et al. “Adaption-of-Thought: Learning Question Difficulty Improves Large Language Models for Reasoning”. In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 5468–5495. DOI: 10.18653/v1/2024.emnlp-main.313.

[203] An Yang et al. “Qwen2.5 Technical Report”. In: *arXiv preprint arXiv:2412.15115* (2024).

[204] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. “Revisiting Semi-Supervised Learning with Graph Embeddings”. In: (2016).

[205] Chih-Kuan Yeh et al. “Representer Point Selection for Explaining Deep Neural Networks”. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS’18. Montréal, Canada: Curran Associates Inc., 2018, pp. 9311–9321.

[206] Rex Ying et al. “GNNExplainer: A tool for post-hoc explanation of graph neural networks”. In: *Advances in neural Information Processing Systems (NeurIPS)*. 2019.

[207] Zhaoning Yu and Hongyang Gao. “Motifexplainer: a motif-based graph neural network explainer”. In: *arXiv preprint arXiv:2202.00519* (2022).

[208] Hao Yuan et al. “On explainability of graph neural networks via subgraph explorations”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 12241–12252.

[209] Seongjun Yun et al. “Graph transformer networks”. In: *Advances in neural information processing systems* 32 (2019).

[210] Xiao Zang et al. “Graph universal adversarial attacks: A few bad actors ruin graph learning models”. In: 2020.

[211] Weidong Zhan et al. *KnowLogic: A Benchmark for Commonsense Reasoning via Knowledge-Driven Data Synthesis*. 2025.

[212] Binchi Zhang et al. “Adversarial Attacks on Fairness of Graph Neural Networks”. In: *The Twelfth International Conference on Learning Representations*. 2024.

[213] Chiyuan Zhang et al. “Counterfactual Memorization in Neural Language Models”. In: *arXiv preprint arXiv:2112.12938* (2021).

[214] Jiaxing Zhang, Dongsheng Luo, and Hua Wei. “MixupExplainer: Generalizing Explanations for Graph Neural Networks with Data Augmentation”. In: *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. KDD ’23. Long Beach, CA, USA: Association for Computing Machinery, 2023, pp. 3286–3296. ISBN: 9798400701030. DOI: 10.1145/3580305.3599435.

[215] Lunjun Zhang et al. *Generative Verifiers: Reward Modeling as Next-Token Prediction*. 2025.

[216] Xiang Zhang and Marinka Zitnik. "Gnnguard: Defending graph neural networks against adversarial attacks". In: *Advances in neural information processing systems* 33 (2020).

[217] Yu Zhang et al. "A survey on neural network interpretability". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 5.5 (2021), pp. 726–742.

[218] Yunxiang Zhang et al. *Small Language Models Need Strong Verifiers to Self-Correct Reasoning*. 2024.

[219] Zirui Zhao et al. *Automatic Curriculum Expert Iteration for Reliable LLM Reasoning*. 2025.

[220] Lianmin Zheng et al. "Judging LLM-as-a-judge with MT-bench and Chatbot Arena". In: *Proceedings of the 37th International Conference on Neural Information Processing Systems*. 2023.

[221] Qinkai Zheng et al. "Graph Robustness Benchmark: Benchmarking the Adversarial Robustness of Graph Machine Learning". In: *CoRR* abs/2111.04314 (2021).

[222] Aojun Zhou et al. *Solving Challenging Math Word Problems Using GPT-4 Code Interpreter with Code-based Self-Verification*. 2023.

[223] Yulin Zhu et al. "FocusedCleaner: Sanitizing Poisoned Graphs for Robust GNN-Based Node Classification". In: *IEEE Transactions on Knowledge and Data Engineering* (2024).

[224] Yulin Zhu et al. "FocusedCleaner: Sanitizing Poisoned Graphs for Robust GNN-Based Node Classification". In: *IEEE Transactions on Knowledge and Data Engineering* 36.6 (2024), pp. 2476–2489. DOI: 10.1109/TKDE.2023.3322129.

[225] Xu Zou et al. "Tdgia: Effective injection attacks on graph neural networks". In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2021, pp. 2461–2471.

[226] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. "Adversarial attacks on neural networks for graph data". In: *ACM SIGKDD international conference on knowledge discovery & data mining*. 2018.

[227] Daniel Zügner and Stephan Günnemann. "Adversarial Attacks on Graph Neural Networks via Meta Learning". In: *International Conference on Learning Representations (ICLR)*. 2019.