RECoRD: A Multi-Agent LLM Framework for Reverse Engineering Codebase to Causal Relational Diagram

Yuan Xue[1,2], Xiaoyu Lu[2], Yunfei Bai[2], Yunan Liu[2], Hoiyi Ng[2]

[1]The Ohio State University
[2]Amazon

Abstract

Understanding the behavior and logical structure of complex algorithms is a fundamental challenge in industrial systems. Recent advancements in large language models (LLMs) have demonstrated remarkable code understanding capabilities. However, their potential for reverse engineering algorithms into interpretable causal structures remains unexplored. In this work, we develop a multi-agent framework, RECoRD, that leverages LLMs to *Reverse Engineering Codebase to Causal Relational Diagram*. RECoRD uses reinforcement fine-tuning (RFT) to enhance the reasoning accuracy of the relation extraction agent. Fine-tuning on expert-curated causal graphs allows smaller specialized models to outperform larger foundation models on domain-specific tasks. Experiments on three real-world use cases - News Vendor, MiniSCOT, and Black-Scholes - demonstrate the effectiveness of our approach. The RFT-trained models significantly outperformed their foundation counterparts, improving F1 score from 0.69 to 0.97 on MiniSCOT. RECoRD also exhibited strong generalization, with models fine-tuned on one use case improving performance on others. We further show how the extracted causal graphs can be leveraged to build a deep-dive assistant that reasons like domain experts, enabling rapid root cause analysis in complex software systems. By automating the construction of interpretable causal models from code, RECoRD has wide-ranging applications in areas such as software debugging, operational optimization, and risk management.

Introduction

Understanding the behavior and logical structure of complex algorithms in large industrial systems is critical for debugging, performance optimization, and security assurance. However, modern software systems are often developed and maintained by multiple teams over extended periods, making it challenging to retain a clear and interpretable representation of the underlying logic.

A prime example of such complex software systems is Amazon's automated planning systems, which are powered by advanced machine learning and optimization algorithms to control billions of dollars in inventory decisions. As these systems grow increasingly sophisticated, with new features supporting faster delivery speeds and capacity management, understanding the intricate relationships between system components becomes both critical and challenging.

While traditional program analysis techniques provide some level of insight, they often fail to capture the causal relationships governing software behavior. In contrast, causal graph-based methods offer a principled approach to address this complexity and answer "why" (attribution) and "what-if" (counterfactual) questions at scale. However, constructing these causal graphs remains a perennial challenge.

A thoughtful approach to software development often starts with the creation of causal relational diagrams (causal graphs), frequently in the form of high-level flowcharts and dependency diagrams, prior to implementing the actual code. These visual models serve as essential blueprints, clearly illustrating the logical structure and relationships between inputs and outputs within the program. Unfortunately, in reality, due to time constraints and rapidly evolving business requirements, developers often update the source code without maintaining the corresponding documentation and graphical models. Consequently, understanding software behavior, especially for non-technical stakeholders, becomes challenging, leading to inefficiencies in debugging, auditing, and overall system comprehension. Constructing causal graphs that accurately represent the underlying code is remarkably difficult. This process typically requires weeks to months of domain expert time, and the resulting graphs rapidly become outdated as the codebase evolves. Furthermore, these models are prone to omissions and inconsistencies across different documentation sources.

In this paper, we aim to address the aforementioned challenges by developing a novel framework called *Reverse Engineering Codebase into Causal Relational Diagram* (RECoRD). Leveraging recent advances in large-language models (LLMs), RECoRD automates the extraction of causal dependencies from source code. RECoRD consists of two key agents: the *entity extraction agent*, which identifies key variables used in the code, and the *relation extraction agent*, which leverages reinforcement fine-tuned LLM to identify the relationships among the code entities. Our method systematically extracts causal relationships among input, intermediate and output variables, providing structured and interpretable representations of complex algorithms while substantially reducing human efforts required to produce and maintain causal graphs. Our key contributions are summarized below:

- We develop a scalable LLM-driven dual-agent framework, RECoRD, that fuses deterministic program analysis with LLM reasoning to automate the extraction of causal graphs from complex computer code, thereby reducing the manual effort required to produce and maintain such representations.
- We propose a novel approach for generating curated training datasets of causal graphs, enhancing pre-trained language models through *reinforcement fine-tuning* (RFT), and employing the fine-tuned LLM to extract causal relationships from source code.
- We conduct empirical evaluations on several public codebases to validate the accuracy and efficiency of the RECoRD framework in producing interpretable causal structures from complex algorithms.
- We present a multi-agent framework that leverages RECoRD to provide scalable root cause analysis for complex systems.

Related Work

Causal discovery and root cause analysis. Causal discovery is a fundamental problem in many domains, including supply chain management, biology, and medicine. Structural causal models, which provide a visual and intuitive approach to understanding complex systems and their underlying causal mechanisms, have been widely studied for causal inference from observational data (Pearl 2009; Hernán and Robins 2020). These methods aim to uncover the underlying causal structure, which can then support root cause analysis (Singal, Michailidis, and Ng 2021; Budhathoki et al. 2021) and counterfactual reasoning. While observational data-driven causal discovery algorithms, such as the Peter-Clark test and kernel conditional independent test (Glymour, Zhang, and Spirtes 2019) have seen progress (Zhang et al. 2011; Shimizu et al. 2006), a fundamental limitation remains: these approaches often require extensive domain knowledge and manually curated data, limiting their scalability and applicability to complex, evolving systems.

Generative knowledge graph construction. While the construction of causal graphs still largely relies on manual efforts, a related and preliminary step has been explored in the form of generative knowledge graph construction. Recent progress in LLMs has enabled new approaches to knowledge graph construction directly from text. Melnyk et al. (Melnyk, Dognin, and Das 2023) proposed an end-to-end system that generates knowledge graphs from textual inputs. The emergence of powerful foundation models like GPT-4 and Claude 3 has transformed the field, allowing knowledge graph construction through zero/one/few-shot prompting. Jiralerspong et al. (Jiralerspong et al. 2024) present a framework that leverages LLMs combined with a breadth-first search approach for full causal graph discovery, achieving state-of-the-art results on real-world causal graphs. Furthermore, Darvariu et al. (Darvariu, Hailes, and Musolesi 2024) demonstrate that LLMs can serve as effective priors for causal graph discovery, integrating background knowledge to improve performance on common-sense benchmarks, particularly in assessing edge directionality. The most recent work (Carta et al. 2023) showed LLM's capability in constructing knowledge graph without additional human efforts. However, these LLM-based methods often struggle with accurately capturing the structural semantics present in source code.

Code understanding and integration of program analysis with LLMs. Code-centric LLMs such as Codex (Chen et al. 2021), CodeGen (Nijkamp et al. 2022), and StarCoder (Li et al. 2023) show promising results in generation and translation, however, their purely token-level view often yields syntactically correct yet semantically wrong code. Hybrid systems therefore inject program structure into the prompt: CodeLLM-Devkit supplies abstract syntax tree and control/data-flow features (Krishna et al. 2024), while LLMDFA decomposes analyses and checks intermediate results with symbolic tracing (Wang et al. 2024). Empirical evidence shows that semantics-preserving rewrites can still mislead leading models, underscoring the need for explicit structural grounding (Nguyen et al. 2024). Building on this line of work, our method combines abstract syntax tree parsing and data-flow tracking with an LLM reasoning agent that first selects salient variables and then infers their causal links. Unlike prior efforts that use structure mainly for summarization (Sun et al. 2024) or synthesis (Jiang et al. 2024), we leverage it to construct interpretable causal directed acyclic graphs (DAGs). This shift from syntax-aware generation to causal structure discovery raises fresh challenges in variable-role disambiguation and edge accuracy, which we address through RFT.

Fine-tuning methodologies for LLMs. Fine-tuning LLMs is essential for aligning model behavior with human preferences and domain-specific requirements. Techniques like reinforcement learning from human feedback (RLHF) (Christiano et al. 2017) and direct preference optimization (DPO) (Rafailov et al. 2023a) have emerged as principled approaches for LLM alignment. Unlike RLHF which involves first training a reward model to predict rankings, and optimizing the language model to maximize expected rewards using algorithms such as proximal policy optimization (PPO) (Schulman et al. 2017a), DPO directly models the likelihood ratio between preferred and dis-preferred responses, offering better training stability and simplicity compared to RLHF (Rafailov et al. 2023b). Other methods, such as instruction tuning (Luo et al. 2024), test-time adaptation, and Nash Learning from Human Feedback (NLHF) (Munos et al. 2024), further enhance LLM generalization and robustness for specialized tasks. These diverse methodologies reflect an evolution in LLM fine-tuning—from supervised fine-tuning (SFT) to RFT that is a feedback-driven optimization framework capable of leveraging evolving instructions, adaptive policies, and human preference data. Our use of RFT in RECoRD builds upon this foundation, enabling efficient and scalable alignment of models to extract causal structure from code.

Methodology

Our proposed RECoRD framework automates the extraction of interpretable causal graphs directly from complex source code by integrating deterministic program analysis and advanced reasoning capabilities of RFTed LLMs. Our approach

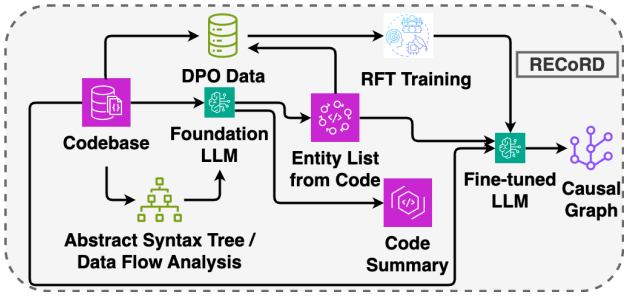

Figure 1: Three-stage causal graph generation flow. The process begins by ingesting and chunking code and documents (Codebase as entry point), proceeds through entity identification, and culminates in relationship extraction that yields a final causal graph.

consists of three sequential yet complementary stages (see Algorithm 1 and Figure 1 for an illustration). First, entities that represent causal nodes are systematically identified through a hybrid process combining *abstract syntax tree* (AST) parsing, data flow analysis, and LLM-guided refinement, optionally followed by manual filtering to mitigate potential inaccuracies. Subsequently, using these refined entities, we employ a carefully crafted prompt engineering strategy to guide foundation LLMs in extracting causal relationships. Finally, recognizing limitations in foundational model outputs, we implement a RFT strategy utilizing DPO informed by expert-annotated preferences, enhancing the precision and reliability of inferred causal relationships. In the rest of this paper, we use the terms causal relational diagram, causal graph, and DAG all interchangeably.

Entity Extraction Agent

The first critical component of RECoRD is automated entity extraction. This process systematically identifies relevant variables within the source code, including inputs, intermediate states, and outputs, which collectively constitute the nodes of our causal graph. We combine program analysis techniques, such as AST parsing and data flow analysis, with LLM-based refinement to achieve high accuracy. The source code is parsed to extract fundamental properties, such as function signatures, parameter lists, return variables, class hierarchies, and object attributes. These properties are then processed by an LLM agent. This agent evaluates the contextual significance of each variable, refining the initial candidate set to ensure comprehensive identification of causal nodes.

Relation Extraction Agent – Prompt Engineering

The relation extraction agent is responsible for inferring the causal relationships between the entities identified by the entity extraction agent. Crafting effective prompts for this task is crucial, as it directly impacts the quality of the inferred causal structures. We employ a multi-stage prompt engineering approach to guide the relation extraction agent:

1. Entity Framing: The prompt begins by clearly stating the list of entities extracted from the code (e.g., entities identified in Section), framing them as the key variables or components in the system.
2. Causal Relationship Elicitation: The agent is then instructed to analyze the relationships between these entities and identify the causal dependencies.
3. Directional Insights: To enrich the causal understanding, the prompt further requests the agent to specify the directionality of each relationship (e.g., what variable is an input to another variable).

We start with prompting large size foundation models (FMs) using code and its corresponding instructions. The following prompt instruction gives the best results. $< entity1 >, < entity2 >, ...$ represent the identified entity from Section . Although FMs can identify a set of key entities within code and infer causal relationships among them, their outputs often contain inaccuracies such as missing causal links and the inclusion of spurious relations not supported by the code. These limitations persist even in state-of-the-art models such as Qwen3-32B. To address these issues, we adopt an RFT approach to enhance the model's code comprehension and improve its reasoning accuracy.

Prompt Instruction

You are a causal relationship extractor and your task is to extract causal relationships from a code snippet and a technical document. Let's extract it step by step:

(1) Identify the entities in the text from the code snippet and the document. An entity must be a noun or a noun phrase that refers to a real-world object or an abstract concept.
(2) Identify the relationships between the entities. A relationship must describe as "is an input to" from the head entity to the tail entity.
(3) List the relationship in triplet format: ('head entity', 'is an input to', 'tail entity').

The code snippet is provided in $< code > < /code >$ tags. The technical document is provided in $< doc > < /doc >$ tags. Only identify the causal relationship from the following nodes: $< entity1 >, < entity2 >, ...$
DO NOT put any preambles in the response. If you don't know the answer, just say I don't know.
DO NOT make up the answer.

Relation Extraction Agent – RFT

To enhance the model's reasoning accuracy, we adopt an RFT approach on smaller-scale models with 7B to 14B parameters. Among various RFT methods, we specifically choose DPO (Rafailov et al. 2023a) to address several fundamental challenges in training causal graph extraction models. First, for large systems, there typically exists only one correct full relational graph, resulting in extremely sparse ground truth samples for training. While correct full graphs are rare, correct subgraphs are more readily available and verifiable, particularly in industrial settings where different teams possess deep domain knowledge of specific system components rather than the entire system. By leveraging these numerous

validated subgraphs for fine-tuning, we can effectively overcome the limitation of sparse full-graph training data while capitalizing on distributed domain expertise. Second, we observe a significant data imbalance: while incorrect subgraphs are abundant (as they include all possible incorrect edge combinations), correct subgraphs are limited in number. Third, our empirical observations show that human experts find it more natural and reliable to express preferences between alternative subgraphs rather than directly validating complete graphs. DPO is particularly well-suited for these challenges as it directly optimizes model behavior through pairwise preference feedback, offering better training stability compared to traditional RLHF approaches like PPO (Schulman et al. 2017b).

We generate various subsets of the full causal graph, assign preference scores to each, and use them to fine-tune the model (see details in Section). This process allows the model to learn causal relationships between smaller sets of entities with a large number of examples, and generalize the reasoning to the full graph. For RFT, we employ DPO in combination of the *Parameter-Efficient Fine-Tuning* (PEFT) (Houlsby et al. 2019; Xu et al. 2023) method with *Low-Rank Adaptation* (LoRA) (Hu et al. 2021) adapters. DPO directly updates the foundation model's parameters based on relative preferences rather than absolute labels, leading to a more stable and efficient learning process. LoRA further enhances efficiency and generalization ability cross a wide range of codebases by fine-tunes only a small subset rather than updating all parameters of the pre-trained model. In our framework, each adapter is specialized for a specific codebase, allowing us to deploy a single foundation model equipped with multiple codebase specific LoRA adapters. This design not only support diverse code understanding task, but also mitigates catastrophic forgetting by isolating task-specific knowledge within separate adapters.

RFT Dataset Generation To generate sub-graphs for training, we set an upper threshold ϵ_{upper} on the proportion of nodes included in the fine-tuning training examples given n number of entities (nodes). For each subsampled list of entities $\{N_1, N_2, \cdots, N_m\}$ where $m \sim Uniform([0, [\epsilon_{upper} \cdot n])$, we generate a pair of the correct sub-graph and an incorrect sub-graph, where the incorrect sub-graph is generated through sampling a DAG with the m entities where the DAG is not the same as the ground truth sub-graph.

$$s(\mathcal{G}_I, \beta) = \max\left(0, \frac{\sum (A_I - \hat{A}_I) \cdot \mathbf{1}_{\{A_I - \hat{A}_I > 0\}}}{|I|^2} + \beta \frac{\sum -(A_I - \hat{A}_I) \cdot \mathbf{1}_{\{A_I - \hat{A}_I < 0\}}}{|I|^2}\right) \quad (1)$$

The DPO preference score is calculated by comparing the edges between the generated sub-graph and the ground truth sub-graph. To allow for asymmetric penalty for missing edges versus wrong edges, we use a weighted sum of the two different penalties controlled by the parameter β as the score:

where I is the set of the nodes in the sub-graph, $\mathcal{G}_I$ refers to the sub-graph with the node list I, A_I and $\hat{A}_I$ are the adjacency matrices of the ground truth sub-graph and the generated sub-graph respectively. A larger value of β imposes a higher penalty on incorrect edges compared to missing edges. In our experimental setup, we consistently use $\beta = 1.3$ to maintain this balance between penalizing false positives and false negatives in edge detection.

In practice, when the full ground truth graph is unknown, we will generate pairs of sub-graphs for human domain experts to rank their preferences, which will then be used for DPO fine-tuning. The larger the threshold ϵ_{upper}, more nodes are involved in the sub-graphs and therefore more domain knowledge is required to label the human preference.

For each prompt, in addition to the code/document, we modify the instruction to let the agent extract causal graph based on the node list involved in each sub-graph only. The prompt, in combination with the positive/negative examples with their scores, constitute the DPO training dataset.

Algorithm 1: RECoRD: Automated Causal Graph Extraction

Require: Source code $\mathcal{C}$, pre-trained LLM $\mathcal{M}$, parameters $\epsilon_{\text{upper}}, \beta, p$
Ensure: Causal graph $G = (V, E)$, adjacency matrix A
Stage 1: Entity Extraction (Sec 3.1)
Generate Abstract Syntax Tree from source code $\mathcal{C}$
Perform data flow analysis on AST to identify initial candidate nodes
Use LLM-based node-list generation agent (Claude 3.7 Sonnet) to refine candidate nodes
Optional: Manually filter nodes to remove obviously irrelevant entities
Finalize refined node list $V \leftarrow \{v_1, v_2, \ldots, v_n\}$
Stage 2: Relation Extraction via Prompt Engineering (Sec 3.2)
Construct prompt using refined nodes V and source code $\mathcal{C}$
Infer preliminary causal relationships $E' \subseteq V \times V$ using prompt-engineered inference with $\mathcal{M}$
Stage 3: Reinforcement Fine-Tuning via DPO (Sec 3.3)
for $t \leftarrow 1$ **to** T **do**
    Sample $m \sim \text{Uniform}(1, \lfloor \epsilon_{\text{upper}} \cdot n \rfloor)$ nodes
    Generate subgraph node set $I \subseteq V$, $|I| = m$
    Generate correct subgraph G_I based on ground truth edges E_I
    Generate incorrect subgraph $\hat{G}_I$ by randomly removing or adding edges proportional to $|E_I| \cdot p$
    Compute preference score $s(G_I, \beta)$ using Eq. (1)
    Collect pair $(G_I, \hat{G}_I)$ with preference score $s(G_I, \beta)$
end for
Fine-tune $\mathcal{M}$ using collected training pairs with DPO and LoRA adapters to obtain fine-tuned model $\mathcal{M}_{\text{RFT}}$
Final Graph Extraction
Use fine-tuned LLM $\mathcal{M}_{\text{RFT}}$ with node set V to infer final causal relationships E
Construct the final causal graph $G = (V, E)$ and derive adjacency matrix A
return G, A

Experiments

We demonstrate RECoRD's ability to efficiently and accurately extract causal DAG given code snippets using three real-world use cases that have causal interpretations (physical

mechanisms) supporting the implementation: News Vendor, MiniSCOT, and Black-Scholes, each with a known ground truth DAG annotated by multiple domain experts with a consensus and readily available open-sourced code base:

- **News vendor:** The news vendor problem (Arrow, Harris, and Marschak 1951) is a classic operations research problem which determines how much of a perishable product to order under stochastic demand, aiming to minimize total expected cost. This cost comes from ordering too much (overage cost) or too little (underage cost). The optimal ordering quantity is a function of the demand distribution, whole sale price, retail price, and salvage cost (https://github.com/amzn/supply-chain-simulation-environmentcode).
- **MiniSCOT** is a simplified simulation platform developed by Amazon. It simulates multiple supply chain metrics evolution over time based on physics of the supply chain operation. In particular, we focus on its strategic buying decision which is one of the most critical and computationally expensive components in real-world supply chain systems (Maggiar, Song, and Muharremoglu 2022). The MiniSCOT buying module optimizes for the optimal purchase order quantities and inbound routing based on demand forecast, inventory positions, planning period and sales patterns across different retail locations (https://github.com/amzn/supply-chain-simulation-environmentcode).
- **Black-Scholes:** The Black-Scholes model (Oksendal 1992) is a classic problem in quantitative finance that determines the optimal price of an European option of certain asset (e.g., stock) which is assumed to evolve according to a geometric Brownian motion. The optimal option price is a function of key input parameters including the initial stock price, drift and volatility parameters, interest rate, strike price, and maturity time (https://github.com/CarloLepelaars/blackscholes/tree/main/src/blackscholescode).

Entity Extraction

Across all three use cases we observe a clear precision–recall trade-off among the entity-extraction pipelines (Table 1). We first apply a lightweight human-filtering step (H): a brief, non-expert review which typically takes 1–2 minutes for graphs with fewer than 100 candidate nodes that deletes clearly irrelevant tokens such as loop counters, logging handles, framework stubs, or generic placeholders. Since it relies only on simple syntactic cues, this step requires no domain knowledge and offers a pragmatic middle ground between fully automatic extraction and full manual annotation, trimming the potential hallucinated entities produced by the extraction agent. This inexpensive sanity check lets the code-only + H pipeline reach perfect precision on the Newsvendor case and slightly higher recall than AST-assisted variants on MiniSCOT, confirming its effectiveness for compact, business-rule–driven code. In structurally richer settings such as Black-Scholes, however, many relevant variables are buried in deeper abstractions; here AST guidance, despite adding noise, captures subtle but critical terms (e.g., higher-order Greeks) and yields substantially better recall. This recall gain is particularly valuable because relation extraction performance is more sensitive to missing nodes than spurious ones, as missing variables preclude the generation of edges entirely. Overall, the AST +LLM +H pipeline strikes the best balance between coverage and precision, and its recall advantage translates to higher downstream edge F1 in Table 2.

Use Case	AST + Code		AST + Code + H		Code-only + H	
	P	R	P	R	P	R
Newsvendor	0.89	0.73	0.89	0.73	**1.00**	0.73
MiniSCOT	0.56	0.63	**0.83**	0.63	0.61	**0.69**
Black–Scholes	0.34	0.58	**0.37**	0.58	0.27	0.21

Table 1: Entity-level precision (P) and recall (R) under different inputs.

Relation Extraction

We experiment with RECoRD's Relation Generation Agent on the above three use cases using the following RFT models: Qwen2.5-7B-Instruct-RFT, Qwen2.5-14B-Instruct-RFT, and Phi-4-RFT. We infer each model with three types of entity list as input: 1) **Full entity list**: ground truth entity list; 2) **LLM+H+AST**: AST+LLM generated entity list with human filtering; 3) **LLM+H**: LLM generated entity list with human filtering. The model performance is measured by the F1-score between the model generated DAG entities and relations and the ground truth annotated by human experts. We benchmark the results from RFT-LLMs using Qwen3-32B foundation model as the baseline.

The results in Table 2 demonstrate that RECoRD can effectively extract interpretable causal graphs from complex codebases in an automated manner. For the MiniSCOT use case, the RFT-based approach was able to accurately recover the causal structure without any wrongly identified nodes or edges (see Figure 2 for the ground truth and learned causal graphs). Even with partial entity lists which is closer to practical settings, the RFT-trained models significantly boosted performance compared to the foundation models. Qualitatively, the remaining errors contain ambiguous or overloaded variables (e.g., higher-order Greeks in Black–Scholes), and we find that incorporating associated documentation and type hints further reduces such failure modes. The full results of causal graph node and edge generation for all the three cases can be found in Appendix.

To demonstrate the generalization capability of RECoRD to a new codebase without human-labeled data, we use a RFT model trained using the News Vendor codebase to extract a causal graph from the MiniSCOT codebase. Figure 3 and Table 3 show the corresponding causal graph and F1 score.

Details on RFT

When conducting RFT with LoRA, we carefully tune the hyperparameters to streamline the efficient learning process and minimize the training loss. Specifically, we grid search over the learning rate, batch size, LoRA rank, and LoRA alpha, to select the optimal size of LoRA training parameter

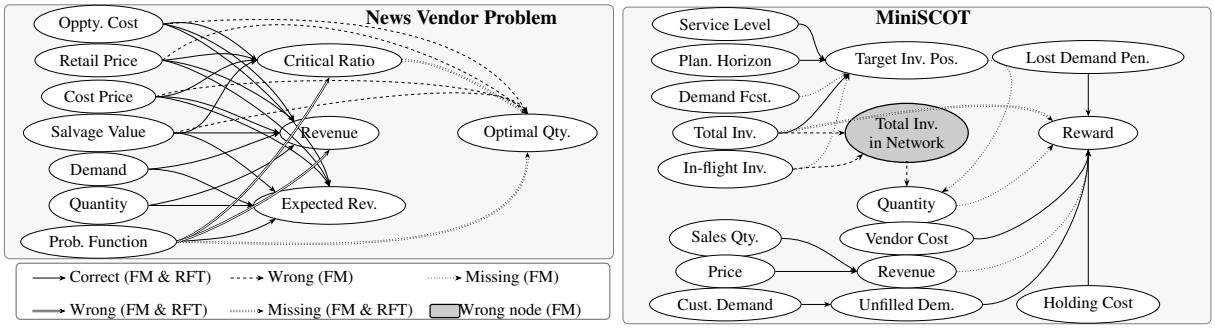

Figure 2: Ground truth and learned causal graphs with full entity list using Qwen 7B-Instruct models for News Vendor Problem (left); and Phi-4 models for MiniSCOT (right). RFT significantly improves the causal discovery accuracy.

Use Case	Model	Full entity list		LLM + H + AST		LLM + H	
		FM	RFT	FM	RFT	FM	RFT
Newsvendor	Claude 3.7 Sonnet	0.765	–	0.710	–	0.728	–
	Qwen-7B-Instruct	0.810	0.895	0.571	0.687	0.462	0.518
	Phi-4	0.737	0.800	0.647	0.647	0.500	0.533
MiniSCOT	Claude 3.7 Sonnet	0.645	–	0.400	–	0.487	–
	Qwen-7B-Instruct	0.667	0.857	0.425	0.519	0.182	0.364
	Phi-4	0.690	0.968	0.375	0.467	0.267	0.323
Black-Scholes	Claude 3.7 Sonnet	0.593	–	0.417	–	0.291	–
	Qwen-14B-Instruct	0.552	0.573	0.483	0.569	0.256	0.286
	Phi-4	0.556	0.562	0.412	0.497	0.284	0.284

Table 2: Graph relation F1-score of RECoRD with different configurations on 3 different use cases. The F1 score is computed against the set of edges in the ground truth causal graph.

Use Case	Model	Full entity list		LLM + H + AST		LLM + H	
		FM	RFT	FM	RFT	FM	RFT
MiniSCOT	Qwen3-32B	0.750	-	0.412	-	0.303	-
	Qwen2.5-7B-Instruct	0.786	**0.857**	0.364	**0.452**	0.182	**0.323**
	Phi-4	0.690	0.733	0.375	0.438	0.121	0.250

Table 3: Graph relation F1-score of RECoRD with different configurations on MiniSCOT use case with model fine-tuned on News Vendor. The F1 score is computed against the set of edges in the ground truth causal graph. Note that since neither the 32B model nor the 7B model possess relevant domain knowledge, we do not expect larger FM models to always outperform smaller models.

for lower training loss and faster convergence time. We set the alpha to four times the rank to improve the LoRA adapter's influence on the full model weights. To analyze the impact of the LoRA adapter size, we conduct an ablation study on the accuracy of the full causal graph construction (Table 3 in Appendix).

All reinforcement fine-tuning runs were executed on a single cloud node. For Qwen-7B-Instruct we used an AWS `p3.16xlarge` instance equipped with 8 × NVIDIA Tesla V100 GPUs (16 GB each), 64 vCPUs, and 488 GB RAM; with LoRA-rank 256 a full DPO cycle required about 4 wall-clock hours. For Phi-4 we used an AWS `g5.48xlarge` instance featuring 8 × NVIDIA A10G GPUs (24 GB each), 192 vCPUs, and 768 GB RAM; the same configuration completed in roughly 4–5 hours. Lower LoRA ranks reduced both memory footprint and time almost proportionally.

Our results suggest that smaller size models (7B and 14B parameters) fine-tuned with carefully curated domain-specific sample data outperform a large size SOTA reasoning model (Qwen3-32B) with higher accuracy, lower latency, and cost. The RFT model demonstrates improved reasoning over the foundation model, capable of explicitly identifying patterns within the code, such as variables that are input to a method or derived from other variables. These results confirm that entity recall plays a critical role in determining overall graph quality,

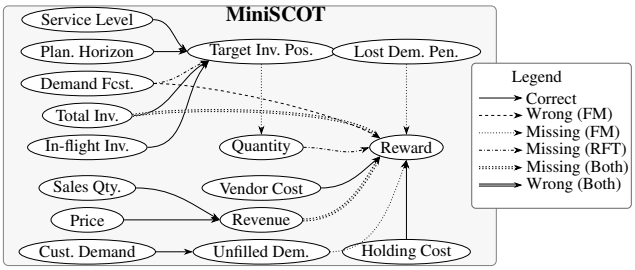

Figure 3: Comparison of ground truth and learned causal graphs for MiniSCOT with full entity list using Qwen 7B-Instruct FM and RFT trained on News Vendor. RFT significantly improves the causal discovery accuracy.

as missing entities inherently limit the ability to recover corresponding edges. AST-guided methods, augmented by simple human filtering, strike an effective balance by maximizing recall without introducing excessive noise, resulting in the highest end-to-end graph F1 scores across all benchmarks (see Table 2).

An LLM Deep-Dive Assistant that Reasons Like Domain Experts

We illustrate the real-world impact of RECoRD using Amazon's supply chain simulation system. This system generates 12-week inventory flow predictions on a weekly basis, modeling events like customer demand, vendor orders, and shipments. As a result, adjacent simulation runs often produce different outputs for a given target week due to evolving inputs.

To understand the root causes of these week-over-week changes, we leverage the causal graph extraction capabilities of RECoRD in combination of causal models such as those described in (Budhathoki et al. 2021). Figure 4 illustrates the architecture behind this deep-dive assistant. By mapping the variables and relationships within the simulation codebase, we can trace how a specific input's change propagates through the system to impact the final predicted outputs.

For example, if a simulation shows a significant deviation in predicted inventory levels, we can quickly identify the key upstream variables responsible - such as shifts in forecasted customer demand or vendor lead times. Armed with this granular understanding of causal pathways, business leaders can then take targeted actions to address the root issue, whether adjusting procurement strategies, optimizing fulfillment, or validating input data.

One such action was the timely cancellation of excess orders placed due to a vendor lead time prediction model deployment error. A shadow model had been unintentionally deployed, causing vendor lead time predictions to increase significantly which led to increase in order quantity that week. This could have created a significant backlog burden on Amazon's fulfillment network. Without automated root cause analysis, this issue may have gone unnoticed until much later, leading to costly over-ordering and significant operational churns. By systematically tracing the causal linkages, the team was able to quickly identify and rollback the

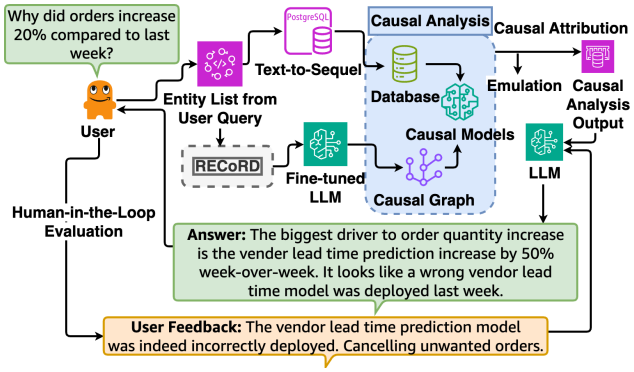

Figure 4: Architecture of the deep-dive assistant powered by RECoRD. A weekly simulation run produces inventory-flow forecasts; RECoRD maps the code that generates those forecasts, while a causal attribution layer pinpoints the inputs most responsible for week-over-week changes.

problematic model, and cancel the excess orders before they were processed, minimizing the operational impact.

This real-time causal analysis empowers Amazon to stay agile in the face of changing market dynamics, responding proactively to potential disruptions. By combining RECoRD's causal graph extraction with causal attribution models, we deliver a powerful decision support tool that amplifies the intelligence of Amazon's supply chain.

Conclusions

We presented RECoRD, a multi-agent pipeline that fuses deterministic program analysis with reinforcement fine-tuned LLMs to turn production code into accurate, interpretable causal graphs with minimal human input. Across three public benchmarks, the method recovers up to 97% of ground-truth edges—often with zero false relations—while allowing compact 7B–14B models to outperform much larger baselines. By extracting system mechanics directly from evolving codebases, RECoRD delivers a scalable foundation for "why" and "what-if" reasoning, root-cause analytics, and auditability in enterprise settings such as Amazon's supply-chain simulators, shortening diagnostic cycles from weeks to minutes and averting costly operational errors.

Limitations and Future Work. Although the proposed sub-graph sampling strategy allows reinforcement fine-tuning from sparse ground-truth data, it may under-represent rare but business-critical dependencies. Ongoing work therefore focuses on importance-weighted sampling and cost-sensitive evaluation metrics that reflect heterogeneous edge impact. We also plan longitudinal studies on live deployments to quantify productivity gains and to examine scalability on monolithic repositories containing millions of lines of code. Finally, we are developing provenance auditing and policy-controlled model releases to balance the clear benefits of automated causal discovery (e.g., safer software and transparent decision pipelines) against risks such as malicious code exploration or error propagation to high-stakes decisions.

References

Arrow, K. J.; Harris, T.; and Marschak, J. 1951. Optimal Inventory Policy. *Econometrica*, 19(3): 250–272.

Budhathoki, K.; Janzing, D.; Bloebaum, P.; and Ng, H. 2021. Why did the distribution change? In Banerjee, A.; and Fukumizu, K., eds., *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, 1666–1674. PMLR.

Carta, S.; Giuliani, A.; Piano, L.; Podda, A. S.; Pompianu, L.; and Tiddia, S. G. 2023. Iterative zero-shot llm prompting for knowledge graph construction. *arXiv preprint arXiv:2307.01128*.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H. P. D. O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Christiano, P. F.; Leike, J.; Brown, T. B.; Martic, M.; Legg, S.; and Amodei, D. 2017. Deep reinforcement learning from human preferences. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, 4302–4310. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781510860964.

Darvariu, V.-A.; Hailes, S.; and Musolesi, M. 2024. Large Language Models are Effective Priors for Causal Graph Discovery. *arXiv preprint arXiv:2405.13551*.

Glymour, C.; Zhang, K.; and Spirtes, P. 2019. Review of causal discovery methods based on graphical models. *Frontiers in genetics*, 10: 524.

Hernán, M.; and Robins, J. 2020. *Causal Inference: What If*. Chapman & Hall/CRC.

Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; Morrone, B.; de Laroussilhe, Q.; Gesmundo, A.; Attariyan, M.; and Gelly, S. 2019. Parameter-Efficient Transfer Learning for NLP. arXiv:1902.00751.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685.

Jiang, J.; Wang, F.; Shen, J.; Kim, S.; and Kim, S. 2024. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

Jiralerspong, T.; Chen, X.; More, Y.; Shah, V.; and Bengio, Y. 2024. Efficient causal graph discovery using large language models. *arXiv preprint arXiv:2402.01207*.

Krishna, R.; Pan, R.; Pavuluri, R.; Tamilselvam, S.; Vukovic, M.; and Sinha, S. 2024. Codellm-Devkit: A Framework for Contextualizing Code LLMs with Program Analysis Insights. *arXiv preprint arXiv:2410.13007*.

Li, R.; Allal, L.; Zi, Y.; Muennighoff, N.; Kocetkov, D.; Mou, C.; Marone, M.; Akiki, C.; Li, J.; Chim, J.; et al. 2023. StarCoder: May the Source be With You! *Transactions on machine learning research*.

Luo, Z.; Xu, C.; Zhao, P.; Sun, Q.; Geng, X.; Hu, W.; Tao, C.; Ma, J.; Lin, Q.; and Jiang, D. 2024. WizardCoder: Empowering Code Large Language Models with Evol-Instruct. In *The Twelfth International Conference on Learning Representations*.

Maggiar, A.; Song, I.; and Muharremoglu, A. 2022. Multi-echelon inventory management for a non-stationary capacitated distribution network. *Optimization Online*.

Melnyk, I.; Dognin, P.; and Das, P. 2023. Knowledge Graph Generation From Text. *arXiv*.

Munos, R.; Valko, M.; Calandriello, D.; Azar, M. G.; Rowland, M.; Guo, Z. D.; Tang, Y.; Geist, M.; Mesnard, T.; Fiegel, C.; et al. 2024. Nash Learning from Human Feedback. In *International Conference on Machine Learning*, 36743–36768. PMLR.

Nguyen, T.-T.; Vu, T. T.; Vo, H. D.; and Nguyen, S. 2024. An empirical study on capability of large language models in understanding code semantics. *arXiv preprint arXiv:2407.03611*.

Nijkamp, E.; Pang, B.; Hayashi, H.; Tu, L.; Wang, H.; Zhou, Y.; Savarese, S.; and Xiong, C. 2022. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*.

Oksendal, B. 1992. *Stochastic differential equations (3rd ed.): an introduction with applications*. Berlin, Heidelberg: Springer-Verlag. ISBN 3387533354.

Pearl, J. 2009. *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

Rafailov, R.; Sharma, A.; Mitchell, E.; Ermon, S.; Manning, C. D.; and Finn, C. 2023a. Direct preference optimization: your language model is secretly a reward model. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23. Red Hook, NY, USA: Curran Associates Inc.

Rafailov, R.; Sharma, A.; Mitchell, E.; Manning, C. D.; Ermon, S.; and Finn, C. 2023b. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36: 53728–53741.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017a. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017b. Proximal Policy Optimization Algorithms. arXiv:1707.06347.

Shimizu, S.; Hoyer, P. O.; Hyvärinen, A.; Kerminen, A.; and Jordan, M. 2006. A linear non-Gaussian acyclic model for causal discovery. *Journal of Machine Learning Research*, 7(10).

Singal, R.; Michailidis, G.; and Ng, H. 2021. Flow-based Attribution in Graphical Models: A Recursive Shapley Approach. In Meila, M.; and Zhang, T., eds., *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, 9733–9743. PMLR.

Sun, W.; Miao, Y.; Li, Y.; Zhang, H.; Fang, C.; Liu, Y.; Deng, G.; Liu, Y.; and Chen, Z. 2024. Source Code Summarization in the Era of Large Language Models. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, 419–431. IEEE Computer Society.

Wang, C.; Zhang, W.; Su, Z.; Xu, X.; Xie, X.; and Zhang, X. 2024. LLMDFA: Analyzing Dataflow in Code with Large Language Models. *Advances in Neural Information Processing Systems*, 37: 131545–131574.

Xu, L.; Xie, H.; Qin, S.-Z. J.; Tao, X.; and Wang, F. L. 2023. Parameter-Efficient Fine-Tuning Methods for Pretrained Language Models: A Critical Review and Assessment. arXiv:2312.12148.

Zhang, K.; Peters, J.; Janzing, D.; and Schölkopf, B. 2011. Kernel-based conditional independence test and application in causal discovery. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, UAI'11, 804–813. Arlington, Virginia, USA: AUAI Press. ISBN 9780974903972.