Using Language Models to Track Causal Links in Software Logs

Vithor Bertalan
Prof. Daniel Aloise

Polytechnique Montréal

DORSAL Laboratory

Motivation

- Software logs are central to debugging and observability
- Identifying **true causal relations** between events is difficult
- Timestamps are unreliable in distributed and multithreaded systems
- Causality is essential for root cause analysis and system reliability

Source Log Line	Target Log Line
Loading authentication module	Authentication module failed to initialize
User clicked "Save Settings"	Settings updated successfully
Authenticating user *jdoe*	User *jdoe* not found. Access denied

Problem Statement

- Given large volumes of software logs, how can we determine which events cause others?
- Correlation does not imply causation
- Traditional methods struggle in complex environments
- Need for a data-agnostic, scalable solution

Example of Log

Causality

Source Log Line	Target Log Line
Loading authentication module	Authentication module failed to initialize
User clicked “Save Settings”	Settings updated successfully
Authenticating user *jdoe*	User *jdoe* not found. Access denied

Conventional Approaches

- PC algorithm
- Granger causality
- Learned causal DAGs
- Strengths: Formal statistical grounding
- Limitations: Require structured logs, Semantic information is often lost

Language Models for Log Causality

- Logs treated as natural language sequences
- Capture: Intra-line semantics, Inter-line dependencies
- Recent work on: Root cause analysis, LLM-assisted causal discovery
- Current limitations: Generic models, Limited specialization for logs

Research Gap

- Lack of methods that:

1)Operate directly on raw logs

2)Avoid templates and manual parsing

3)Use the language model itself as a causal mechanism

- Core idea: Use the decoder as a causal transition scorer

Proposed Approach

- Four-stage pipeline:

1)Synthetic log generation with known causality

2)Automatic hyperparameter estimation

3)Training a GPT-NeoX language model

4)Causal evaluation against PC and Granger

Synthetic Log Generator

- Controlled generation of CI/CD-like logs
- Five block types:

1)Causal bursts

2)Spurious pairs

3)Common-cause bursts

4)Noise bursts

5)Single actions

- Explicit and directional causal ground truth

Ground-Truth

Causal Graph

- Hand-crafted causal chains (e.g., build → test → deploy)
- Each adjacent pair forms a directed causal edge
- Only these edges are considered true causality
- Correlation-only patterns are penalized

Source Log Line	Target Log Line 1	Target Log Line 2
"Cloning repository"	"Checking out branch"	"Installing dependencies"
"Building CXX object"	"Linking CXX executable"	"Running unit tests"
"Deploying to staging environment"	"Deploying to production"	"Validating deployment"

Hyperparameter Estimation

- Automatically estimates: Vocabulary size, Context window length
- Uses dataset statistics
- Evaluates SentencePiece tokenization
- Goal: Maximize information density, Control memory and compute cost

Language Model Training

- GPT-NeoX trained from scratch
- SentencePiece tokenization
- Each log line ends with a special *SENT_END* token
- Model jointly learns:

1)Line-level semantics

2)Event-to-event transitions

Causal Inference

- Decoder predicts the next log event
- Uses logits of the final token
- Top-*1* and Top-*k* predictions
- Predicted transitions → causal edges
- No prompting, no chat, pure sequence modeling

Evaluation

Setup

- Compared against:

1)PC algorithm

2)Granger causality

- Same input data
- Same causal ground truth
- Main metric: Next@*k* (Top-*1*, Top-*5*)

Results

- Top-*k*

Method	**Top-1**	**Top-5**
Our Method	**98.6%**	**99.7%**
Granger	59.1%	78.2%
PC	40.2%	61.2%

- Execution Speed: Dataset w/ 250k lines

Method	**Running Time (in min)**
Our Method	**92**
Granger	107
PC	Not applicable (see info)

Conclusions

- Fully data-agnostic pipeline
- No parsing or handcrafted rules
- Decoder-based causal discovery
- Practical insights:

1)Ground truth is iterative and collaborative

2)Human-in-the-loop is essential