Towards Formally Describing Program Traces of Chains of Language Model Calls with Causal Influence Diagrams: A Sketch

v1.1

Brian Muhia,
CTO, Fahamu Inc

May 30 2023

Abstract

In this report, we analyze five agent designs implemented by an interface built within the factored cognition framework (https://primer.ought.org/), and use casual influence diagrams (CIDs) (Everitt et al, 2021) to formally describe the agent interactions, their influences and how they contribute towards performing a user query, based on context from a document archive.
This helps us in documenting the system while making plans for future changes, such as adding a reranking step, or adding verifiers that remove unwanted outputs or associations.
A research program on interface design is outlined, wherein we plan the use of mechanistic interpretability tooling to improve the causal influence diagrams by adding some weak notion of conditional probability distributions, and hypothesize their possible contribution towards formally modeling and representing the introduction of deceptive answers in a dataset of responses from agents involved in generating subquestions, filtering, answering sub-questions and aggregating context.

Introduction

When we compose small, independent contributions from agents that know about the data they're being asked about, we introduce a bias for representing the documents provided, which is counterbalanced by the bias from the models involved and how they have been trained, which may improve the answer quality by faithfully and factually representing the context, or they may entirely ignore relevant parts of the context and focus on others, or go off topic entirely.

We aim to illuminate this complex problem first by formally describing how the designs of two agent calls interact with context data and then by discussing how we might in the future analyze the dataset using a smaller local model with a causal tracing framework in order to visualize the factuality of agent responses. The specifics of this and analysis of the related data are left to future work.

Related Work

The Causal Incentives working group (https://causalincentives.com/) has supported the release of the Python library PyCID (https://github.com/causalincentives/pycid), which we use to automate the visualisation process.

Process-Based Supervision

We aim to understand how systems designed through the task decomposition framework generate answers by observing and controlling data flow through the process by which answers are generated. We hope to learn more about how the specific type of answering process is working in practice by tracing chained calls to large language models and formalising the resulting program trace in a diagram that documents it. For this, we have several data points to build on, including:

- User question
- Answer
- Data Source (list of paragraphs before filtering)
- Context (list of paragraphs after classifying and filtering)
- (Optional) Sub-questions & Sub-answers (Generated by sub-agents)

We aim to use data to study the causal relationships between each of these steps, as they are occurring at different times in the process of answering a user's question, and thus influence each other in a directed acyclic graph.

Causal Influence Diagrams in Practice

Definition: A **Causal influence Diagram (CID)** (Everitt et al, 2021) is a tuple ($\mathbf{V}$,$\mathbf{E}$) where:

- ($\mathbf{V}$,$\mathbf{E}$) is a directed acyclic graph with a set of
- vertices $\mathbf{V}$
- connected by directed edges $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$

  These vertices are partitioned into:

  - $\mathbf{D} \subseteq \mathbf{V}$ is a set of decision nodes represented by rectangles.
  - $\mathbf{U} \subseteq \mathbf{V} \setminus \mathbf{D}$ is a set of utility nodes represented by diamonds and utility nodes have no successors.
  - $\mathbf{X} = \mathbf{V} \setminus (\mathbf{D} \cup \mathbf{U})$ is the set of chance nodes represented by ovals.

Definition: A **Causal influence Model (CIM)** (Everitt et al, 2021) is a tuple
($\mathbf{V}$,$\mathbf{E}$,θ) where ($\mathbf{V}$,$\mathbf{E}$)
is a CID and:

- Θ is a parameterisation over the nodes in the CID specifying:
  - A finite domain $dom(V)$ for each node $V \in \mathbf{V}$
  - Real-valued domains $dom(U) \subset \mathbb{R}$ for all utility nodes $U \in \mathbf{U}$.
  - A set of conditional probability distributions (CPDs), $\Pr(\mathbf{V} \mid pa_V)$, for every chance and utility node $\mathbf{X} \cup \mathbf{U} \in \mathbf{V}$. Taken together, these form a partial distribution $\Pr(\mathbf{X},\mathbf{U}:\mathbf{D}) = \prod_{V \in \mathbf{V} \setminus \mathbf{D}} \Pr(V|Pa_V)$ over the variables in the CID.

In this report we only describe CIDs, leaving the automated program tracing and mechanistic interpretability work that would get us CIMs for future research.

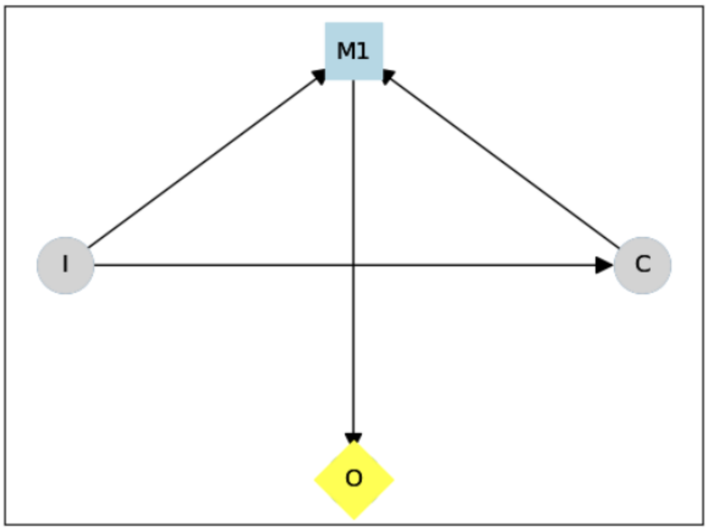
An example showing five pycid CIDs. These are manually written to describe two LLM chains that are currently being tested. The future hope is to use mechanistic interpretability first to introduce numerical weights and thus find a basis for representing CPDs with some *weak* notion of (positive or negative) contribution to the total factuality of the response. If there is merit to this, we aim to design a mechanism for automating the generation of these *weakly* factuality-weighted CIDs when provided with a program trace.
Notation:

- **Mn** = Models/Agents are represented as **decision nodes**
- **C** = Context
- **I** = User Input (or Intent)
- **O** = Output is a utility node

Agent Diagram 1

This first call instantiates an agent **M1** to answer a single user question, and a list of paragraphs **C** obtained after a separate filtering process as context, using it to answer the question as output **O**

```
# !pip install pycid
# code to generate the diagrams appears next to each diagram

import pycid
m1_cid  = pycid.CID(
    [
        ("C", "M1"),
        ("I", "C"),
        ("I", "M1"),
        ("M1", "O")
    ],
    decisions=["M1"],
    utilities=["O"]
)
m1_cid.draw()
```



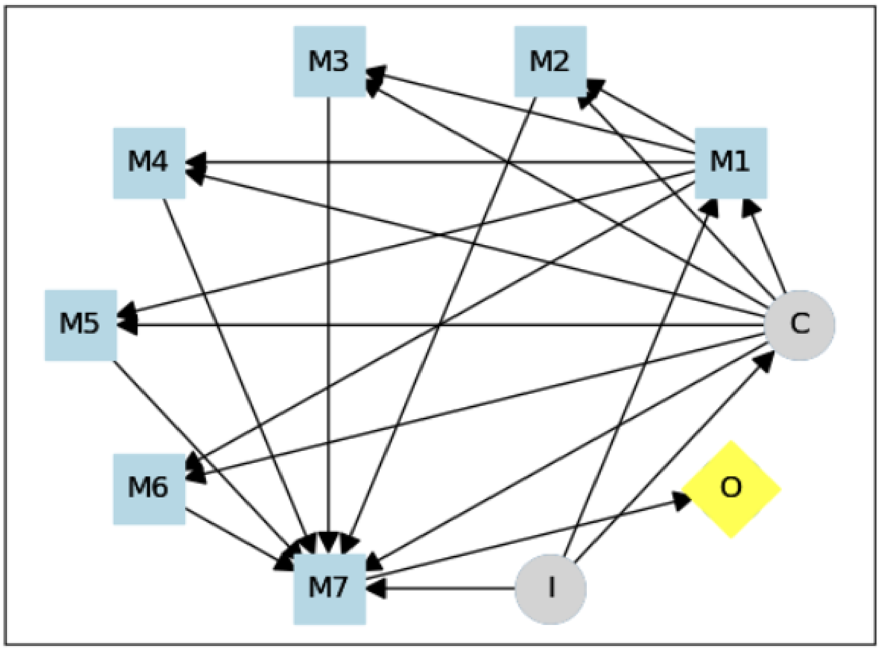
Single interaction with the model M1

Agent Diagram 2

Parallel Scenarios

The second call first instantiates a single agent **M1** to generate 2-5 sub-questions based on user input **I** and the document context **C**. This instantiates 5 parallel sub-agents (**M2**-**M6**) to answer each of these sub-questions using the context, and a 7th agent **M7** to aggregate the sub-answers and context into a more detailed answer as output **O**.

(PV1) Parallel Variant 1 where the user's intent is omitted from the context of agents M2-M6.

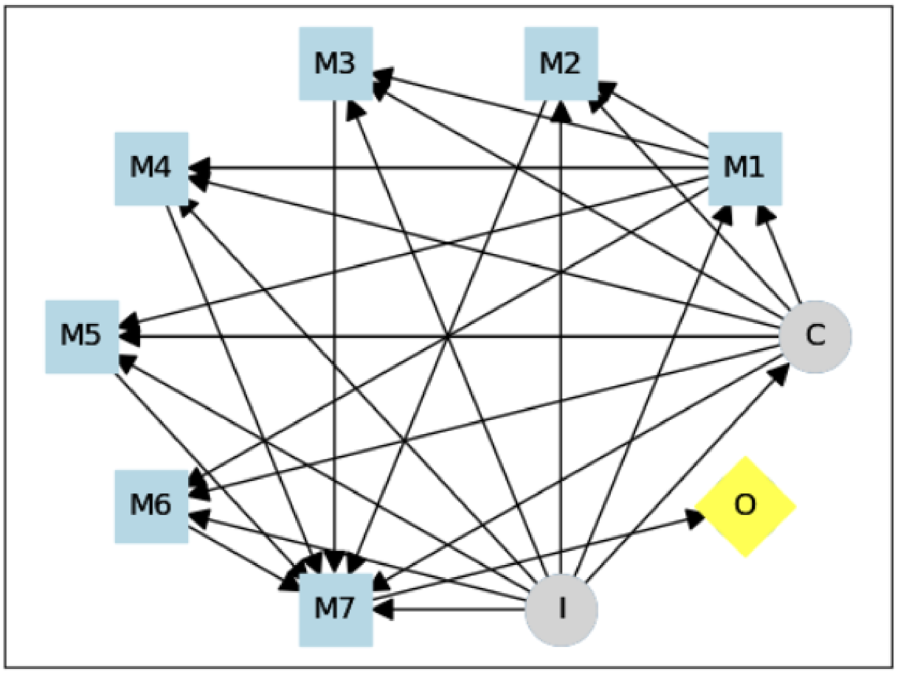


```
variant_1  = pycid.CID(
    [
        ("C", "M1"),
        ("C", "M2"),
        ("C", "M3"),
        ("C", "M4"),
        ("C", "M5"),
        ("C", "M6"),
        ("C", "M7"),
        ("I", "M1"),
        ("I", "M7"),
        ("M1", "M2"),
        ("M1", "M3"),
        ("M1", "M4"),
        ("M1", "M5"),
        ("M1", "M6"),
        ("M2", "M7"),
        ("M3", "M7"),
        ("M4", "M7"),
        ("M5", "M7"),
        ("M6", "M7"),
        ("M7", "O"),
    ],
    decisions=["M1", "M2", "M3", "M4", "M5", "M6", "M7"],
    utilities=["O"]
)
variant_1.draw()
```

My current hypothesis is that there are millions of these diagrams and also that there are easy ways to see possible failure modes in the designs of factored cognition schemes before they are implemented.

(PV2) Parallel Variant 2 where the user's intent is included in the context of all agents M1-M7.

```
variant_2  = pycid.CID(
    [
        ("C", "M1"),
        ("C", "M2"),
        ("C", "M3"),
        ("C", "M4"),
        ("C", "M5"),
        ("C", "M6"),
        ("C", "M7"),
        ("I", "M1"),
        ("I", "M2"),
        ("I", "M3"),
        ("I", "M4"),
        ("I", "M5"),
        ("I", "M6"),
        ("I", "M7"),
        ("M1", "M2"),
        ("M1", "M3"),
        ("M1", "M4"),
        ("M1", "M5"),
        ("M1", "M6"),
        ("M2", "M7"),
        ("M3", "M7"),
        ("M4", "M7"),
        ("M5", "M7"),
        ("M6", "M7"),
        ("M7", "O"),
    ],
    decisions=["M1", "M2", "M3", "M4", "M5", "M6", "M7"],
    utilities=["O"]
)

variant_2.draw()
```

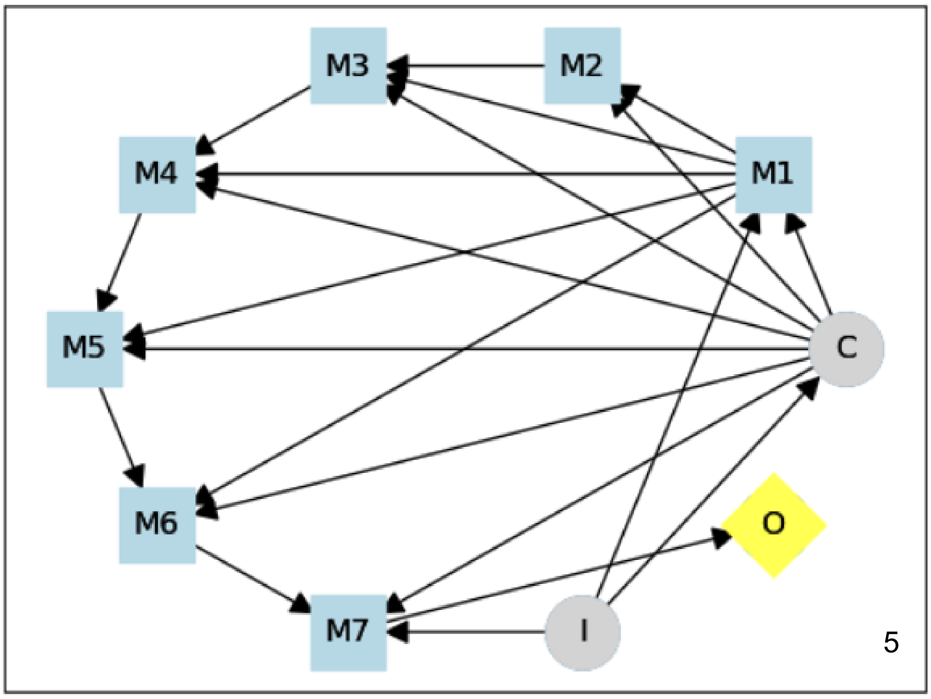


Sequential Scenarios

We clean up the two above variants to prevent data duplication in the prompt, and add a few more links to describe 2 more variants that explore the idea of chaining in a more sequential way, where data flows directly from M_n to M_{n+1}.We use this formalism to note that as **decision nodes** with some level of independence and unreliability, factuality might be broken at any stage and we would want to detect it.

(SV1) Sequential Variant 1 where the user's intent is omitted from the context of agents M2-M6.

```
variant_3  = pycid.CID(
    [
        ("C", "M1"),
        ("C", "M2"),
        ("C", "M3"),
        ("C", "M4"),
        ("C", "M5"),
        ("C", "M6"),
        ("C", "M7"),
        ("I", "M1"),
        ("I", "M7"),
        ("M1", "M2"),
        ("M1", "M3"),
        ("M1", "M4"),
        ("M1", "M5"),
        ("M1", "M6"),
        ("M5", "M6"),
        ("M2", "M3"),
        ("M3", "M4"),
        ("M4", "M5"),
        ("M5", "M6"),
```

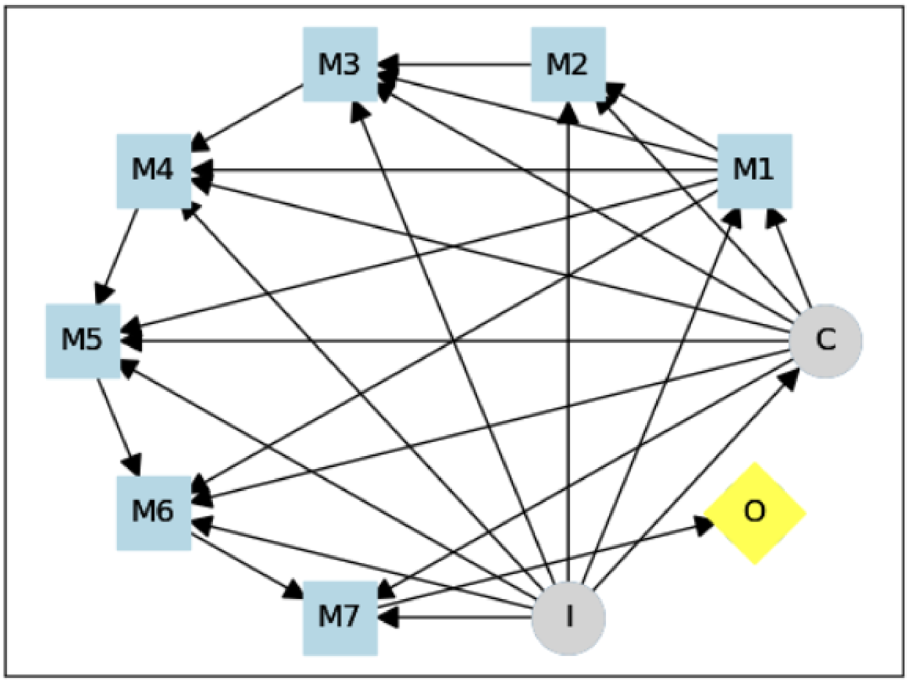


```
        ("M6", "M7"),
        ("M7", "O"),
    ],
    decisions=["M1", "M2", "M3", "M4", "M5", "M6", "M7"],
    utilities=["O"]
)

variant_3.draw()
```

By intuition, my prediction is that in sequential scenarios errors would compound in subtle ways to more sub-agents faster than in parallel scenarios. There are still more opportunities for either early stopping or self-healing in the sequential case since assuming the detection of a lack of factual grounding in e.g. node **M3** in the two Sequential variants, means that we may have an opportunity to remove that sub-answer and retry, or remove that node entirely by blanking out the text. **Qn:** So what would be a good way to deal with factuality issues in Parallel Agents **M2-M6**?

(SV2) Sequential Variant 2, where the user's intent is included in the context of all agents M1-M7.

```
variant_4  = pycid.CID(
    [
        ("C", "M1"),
        ("C", "M2"),
        ("C", "M3"),
        ("C", "M4"),
        ("C", "M5"),
        ("C", "M6"),
        ("C", "M7"),
        ("I", "M1"),
        ("I", "M2"),
        ("I", "M3"),
        ("I", "M4"),
        ("I", "M5"),
        ("I", "M6"),
        ("I", "M7"),
        ("M1", "M2"),
        ("M1", "M3"),
        ("M1", "M4"),
        ("M1", "M6"),
        ("M1", "M5"),
        ("M5", "M6"),
        ("M2", "M3"),
        ("M3", "M4"),
        ("M4", "M5"),
        ("M5", "M6"),
        ("M6", "M7"),
        ("M7", "O"),
    ],
    decisions=["M1", "M2", "M3", "M4", "M5", "M6", "M7"],
    utilities=["O"]
)

variant_4.draw()
```



Exercises

1. How would one filter the data flow to **M7** and prevent deceptive answers from the previous agents from causing **M7** to deceive the user, assuming each agent is equally unreliable.
2. Which of **variant_1** vs **variant_2** or **variant_3** vs **variant_4** are more aligned with the user's intent?
    a. When you consider all of them, together?
    b. Could we rank them in order of "alignment to user intent"?
    c. Which would be more likely to cause an accident?
    d. Could we tell just from analysing the drawing, i.e. before implementing them?

Hint: We might assume that **M7** gets the same information in both (Sequential and Parallel) cases of **Variant 1**, but that variant either benefits, or fails creatively or catastrophically, from a slightly more independent search for answers that may or may not be faithful to the context, given its instructions and the sub-questions generated by **M1**.

Relationship to ML Safety

1. Assuming a system that is given a lot of and power over its environment is coordinated in this way, we would want to characterise coordination failures from agents that are assumed to be equal to one another but each have the same failure mode that can compound across agents. A coordination failure in **SV1** might get much worse very quickly when left unchecked. The strategy of intervening on the data flow in each step to create an illusion of safety breaks down when the pillars themselves are unstable.
2. Formalising the ways in which multi-agent systems interact with their users is a purpose of the causal incentives program. My aim in this is to add more links between the two dense subgraphs of mechanistic interpretability and causal incentives.
3. Understanding the ways this intersects with questions related to introduced unfairness/bias/lack of generalizability through training methods.

References

1. Causal Incentives Working Group https://causalincentives.com
2. PyCID: A Python Library for Causal Influence Diagrams https://conference.scipy.org/proceedings/scipy2021/pdfs/james_fox.pdf
3. Agent Incentives: A Causal Perspective https://arxiv.org/pdf/2102.01685.pdf
4. Modeling AGI Safety Frameworks with Causal Influence Diagrams https://arxiv.org/pdf/1906.08663.pdf
5. Understanding Agent Incentives using Causal Influence Diagrams, Part 1: Single Action Settings https://arxiv.org/abs/1902.09980
6. Locating and Editing Factual Associations in GPT https://rome.baulab.info/