Investigating Implicit Reasoning in Counter-Argument Logical Structure Analysis

Wenzhi Wang[1,2] Paul Reisert[3] Shoichi Naito[1,2,4] Naoya Inoue[5,2]
Machi Shimmei[1] Surawat Pothong[5] Jungmin Choi[2] Kentaro Inui[6,1,2]
[1]Tohoku University [2]RIKEN [3]Beyond Reason
[4]Ricoh Company, Ltd. [5]JAIST [6]MBZUAI
{wang.wenzhi.r7, naito.shoichi.t1}@dc.tohoku.ac.jp, beyond.reason.sp@gmail.com
{naoya-i, spothong}@jaist.ac.jp, machi.shimmei.e6@tohoku.ac.jp, jungmin.choi@riken.jp, kentaro.inui@mbzuai.ac.ae

Abstract

Counter-**A**rgument **L**ogical **S**tructure **A**nalysis (**CALSA**) is a task that analyzes logic patterns of a counter-argument in relation to an initial argument. It holds substantial educational value, as informative feedback for improving counter-arguments can be provided based on the analyzed logic pattern. However, due to the complex nature of the task, the implicit reasoning skills required to identify these underlying logic patterns present significant challenges for current LLMs. To address this, we explore decomposing the logic patterns into fine-grained *logic components* and tackling them individually. Our experimental results demonstrate improvements compared to identifying coarse-grained logic patterns. More importantly, we find that whether predicted logic patterns can be considered plausible deeply depends on the degree of implicitness involved in interpreting an argument.

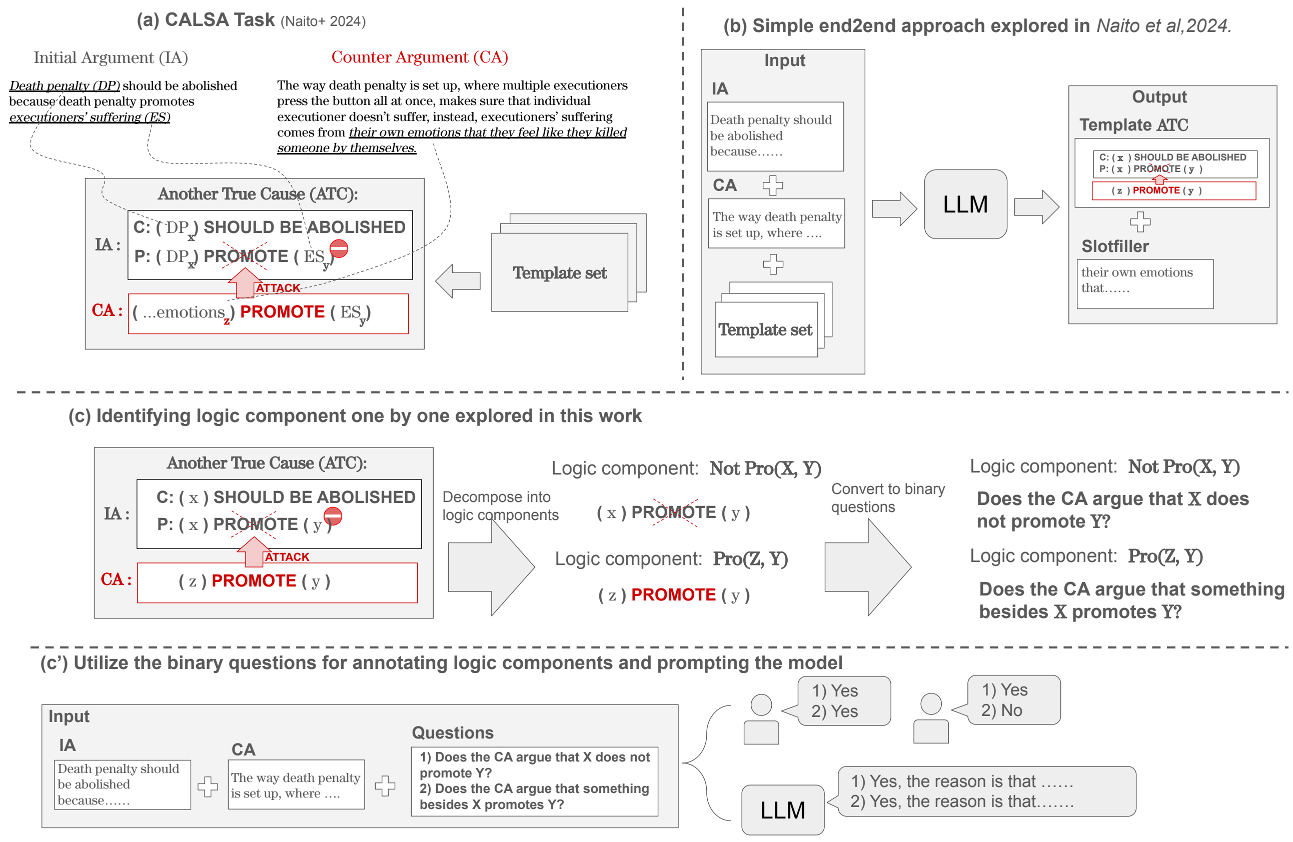
1 Introduction

CALSA is a task for analyzing the logic patterns of counter-arguments (CA) in relation to initial-arguments (IA) [1] in the setting of debates. Figure 1(a) shows an example of a logic pattern annotated on top of a CA from CALSA. The task is beneficial as capturing the underlying logic patterns of CAs can be potentially leveraged for providing constructive feedback to a learner's CA, which would help foster their critical thinking skills.

Previous work on CALSA has only explored a simple end-to-end approach where they utilize Large Language Models (LLMs) to directly generate the logic patterns of a given CA [1]. (Figure 1(b)) They conclude that the task is challenging for current LLMs due to the implicit reasoning abilities required to identify the underlying logic patterns. Furthermore, while they claim that multiple logic patterns may exist due to multiple possible interpretations of a CA, they did not collect an exhaustive list of logic patterns for CAs, which hinders the reliability of evaluating LLMs' abilities in solving the task.

Divide and conquer is a common strategy of decomposing a challenging task into simpler subtasks, which has been widely applied in computer science [2, 3]. In the field of NLP, many works have exhibited the effectiveness of problem decomposition for various tasks [4, 5, 6, 7, 8, 9, 10, 11]. Upon further investigation on the logic patterns proposed in the CALSA task, we found that most logic patterns can be considered as consisting of multiple finer *logic components*, such as causal relations (i.e., *Promote(X, Y)*/*Suppress(X, Y)*) defined in the previous work [12, 13]. Therefore, it begs the question: If the implicit reasoning required to identify a combination of *logic components* all at once causes LLMs to struggle with the task, would identifying each *logic component* one by one (i.e., *divide and conquer*) reduce the complexity of the implicit reasoning and consequently make the overall task easier?

To answer the question, in this work, we explore evaluating LLMs on identifying each logic component individually. Given that the CALSA dataset does not provide an exhaustive list of logic patterns, we do not have a complete set of labels for all *logic components* for a given CA. Therefore, we first conduct an annotation study for collecting labels for all the *logic components* for a given CA. We consider identifying each *logic component* as a



(a) CALSA Task (Naito+ 2024)
Initial Argument (IA)
Counter Argument (CA)
Death penalty (DP) should be abolished because death penalty promotes executioners' suffering (ES)
The way death penalty is set up, where multiple executioners press the button all at once, makes sure that individual executioner doesn't suffer, instead, executioners' suffering comes from their own emotions that they feel like they killed someone by themselves.
Another True Cause (ATC):
IA :
C: (DP x) SHOULD BE ABOLISHED
P: (DP x) PROMOTE (ES y)
ATTACK
CA :
(...emotions z) PROMOTE (ES y)
Template set
(b) Simple end2end approach explored in Naito et al,2024.
Input
IA
Death penalty should be abolished because……
CA
The way death penalty is set up, where ….
Template set
LLM
Output
Template ATC
C: (x) SHOULD BE ABOLISHED
P: (x) PROMOTE (y)
(z) PROMOTE (y)
Slotfiller
their own emotions that……
(c) Identifying logic component one by one explored in this work
Another True Cause (ATC):
IA :
C: (x) SHOULD BE ABOLISHED
P: (x) PROMOTE (y)
ATTACK
CA :
(z) PROMOTE (y)
Decompose into logic components
Logic component: Not Pro(X, Y)
(x) PROMOTE (y)
Logic component: Pro(Z, Y)
(z) PROMOTE (y)
Convert to binary questions
Logic component: Not Pro(X, Y)
Does the CA argue that X does not promote Y?
Logic component: Pro(Z, Y)
Does the CA argue that something besides X promotes Y?
(c') Utilize the binary questions for annotating logic components and prompting the model
Input
IA
Death penalty should be abolished because……
CA
The way death penalty is set up, where ….
Questions
1) Does the CA argue that X does not promote Y?
2) Does the CA argue that something besides X promotes Y?
1) Yes
2) Yes
1) Yes
2) No
LLM
1) Yes, the reason is that ……
2) Yes, the reason is that…….

Figure 1 **(a)**: The overview of CALSA task. The task is composed of two parts: 1) selecting a logic pattern template from a predefined template set and 2) extracting slotfillers from the CA that fill into the placeholders (Z) in the template. ATC refers to the original label for a logic pattern proposed in the CALSA paper. **(b)**: The input/output structure of the simple end2end approach explore in the original CALSA paper. The input contains an IA passage, a CA passage, and a list of all proposed CA logic pattern templates described in natural language, while the output contains the identifiers of the corresponding logic pattern templates and the slotfillers that fill into the placeholder of the templates. **(c)**: An example showing how we decompose a logic pattern into its constituent *logic components*, and the binary question created for each component. **(c')**: An example showing how the binary questions associated with *logic components* are utilized for annotations and prompting LLMs.

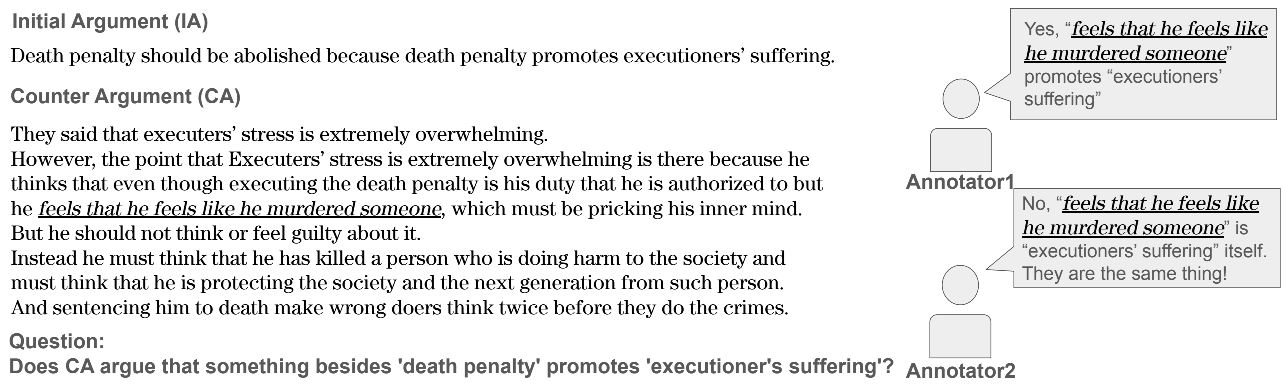
Question-Answering (QA) task where we create a binary question asking for the existence of a corresponding *logic component* for a given CA for both annotation study and modeling experiments (Figure 1(c) and Figure 1(c')). It is worth noting that an exhaustive list of logic patterns can be obtained by aggregating the labels for *logic components*, which can be subsequently utilized for comprehensively evaluating LLMs' performance at a logic pattern level.

As a result, we collect 250 annotations for different *logic components*. Our key insight from the annotation study is that: It is possible to consider that a CA contains and does not contain a given *logic component* simultaneously, depending on the degree of implicit reasoning involved in the annotators' decision process. Furthermore, the modeling experiments using the collected annotations show that: The LLM is able to conduct implicit reasoning required to identify each *logic component* individually to some extent , and the overall performance also improved compared to identifying logic patterns directly. However, LLM's reasoning process may or may not align with human annotators, depending on the degree of implicit reasoning being considered.

Overall, we claim that the real challenge of CALSA task lies in how to determine the desirable **degree of implicit reasoning** to be considered when identifying a *logic component*, and how to align model's reasoning process with the desirable implicit reasoning. We provide further details and discussion in the subsequent sections.

2 Constituent *logic components* for CA logic patterns

We observe that the CA logic patterns proposed in the original CALSA paper can be considered as composed of multiple finer *logic components*. Most of the *logic components* are related to the causal relations between two


Initial Argument (IA)

Death penalty should be abolished because death penalty promotes executioners' suffering.

Counter Argument (CA)

They said that executers' stress is extremely overwhelming.
However, the point that Executers' stress is extremely overwhelming is there because he thinks that even though executing the death penalty is his duty that he is authorized to but he *feels that he feels like he murdered someone*, which must be pricking his inner mind.
But he should not think or feel guilty about it.
Instead he must think that he has killed a person who is doing harm to the society and must think that he is protecting the society and the next generation from such person.
And sentencing him to death make wrong doers think twice before they do the crimes.

Question:
Does CA argue that something besides 'death penalty' promotes 'executioner's suffering'?

Yes, "*feels that he feels like he murdered someone*" promotes "executioners' suffering"

No, "*feels that he feels like he murdered someone*" is "executioners' suffering" itself. They are the same thing!

Figure 2 An example of annotation for *Sup(Z, Y)* where both annotators' interpretations are plausible.

concepts (i.e., *Promote(X, Y) or Suppress(X, Y)*), defined in previous work [12, 13]. Other types of *logic components* are fairly self-explanatory. For instance, one of the original logic patterns named *Mitigation* is defined as *While CA acknowledges IA's logic, it argues that the causal relationship stated in the IA can be mitigated by something Z.* It is obvious that this logic pattern can be considered as a combination of two *logic components*: *Acknowledge(IA)* and *Mitigation(IA, Z)*. We show the mapping between logic patterns and *logic components* in Table 3.

3 Annotation Study

3.1 CALSA Dataset

We utilize Naito et al. [1]'s CALSA dataset where each CA is annotated with underlying logic patterns in relation to an IA. The dataset was annotated by crowdworkers, where workers were only able to select one pattern per sentence for a given CA. The results were aggregated to allow one pattern per sentence, and a follow-up annotation by one expert indicated that multiple patterns could be considered per one sentence. Although multiple patterns were considered, they were not exhaustively annotated as a result of the follow-up annotation.

3.2 Annotating *Logic Components*

We consider the identification of a *logic component* as a QA task. Each *logic component* is associated with a binary question, e.g., *Sup(Z, Y)* $\rightarrow$ *"Does CA argue that something suppresses Y?"*, *Y* is filled with a concept depending on IA's logic (e.g., *misjudgment*), while *Z* is supposed to be extracted from the given CA. A list of binary questions associated with all *logic components* is shown in Table 4.

As a start, we select 25 IA and CA pairs spanned across 3 varying topics from the CALSA devset for annotation. Two authors of this paper (expert annotators) annotate the binary questions associated with 10 *Logic Components* for all 25 CAs (in total 250 annotations per annotator).

3.3 Annotation Results and Analysis

We calculate the Cohen's Kappa for 250 binary labels. We achieve a score of 0.49, indicating moderate agreement between the annotators. The observed agreement is 189/250 annotations. In order to better understand the disagreements, both annotators had a discussion and provided reasoning for each of their disagreed annotations.

Analyzing disagreements, for 17 instances, either one annotator agreed to the other's choice after discussing and considering the reasoning provided by the other annotator. However, we discover that **for 44 disagreements, both annotators' reasoning can be considered plausible depending on how implicit we interpret the target CA.** Moreover, the degree of implicit interpretation depends on the annotator's background knowledge. Figure 2 shows an example of a disagreeing instance in which both annotators' interpretations are plausible. The key disagreement lies in whether to consider *Z* and *Y* as the same concept, and whether to think so depends on how much implicit knowledge utilized to interpret the CA's target expression.

We aggregate the labels for all *logic components* to obtain an exhaustive list of labels for logic patterns for each CA. A logic pattern is labeled only if the answers to the binary questions associated with all its constituent *logic components* are "Yes". For CAs where both annotators' interpretations are plausible, we consider both annotators' labeling correct and consider the union of their annota-

Table 1 Precision (P), Recall (R), F1 scores of logic pattern identification. G4o-mini: GPT-40-mini

	Baseline			*Decomp*		
Model	**P**	**R**	**F1**	**P**	**R**	**F1**
G4o-mini	.50	.29	.35	.52	.51	.50

tions as the final set of labels for those CAs. As a result, on average, each CA ends up with 3.2 labeled logic patterns.

4 Model Experiment

To evaluate the efficacy of the identifying *logic component* one by one, we conduct experiments around GPT-4o-mini with the collected 250 instances. Given the comparably small size of our collected dataset, we only consider zero-shot setting as a start in this paper. We compare the results of identifying *logic components* one by one and then aggregate (***Decomp***) with that of prompting the model to identify all logic patterns at once (***Baseline***). To avoid brittle hand-crafted prompts for ***Baseline*** setting, we utilize DSPy [14] to programmatically perform chain-of-thought prompting, where we set the output signature to be a list of identified logic patterns and their corresponding slotfillers. For ***Decomp*** setting, we simply prompt the model to answer the given binary question as shown in Figure 1(c') without using DSPy as the objective is fairly simple.

4.1 Results

Given that both annotators' interpretations are plausible for a comparably large portion of instances, we divide the model results (for *logic component* identification) into two parts for further evaluation and analysis. For the portion where both annotators finally agree with each other (*agree portion*, 206 instances), one annotator manually checked the results, including the model's answer to the question and the reasoning, for 83 instances where the model's generated answer to the binary question is not consistent with the annotators'. For the portion where both annotators reasoning can be considered plausible (*plausible portion*), one annotator manually checked the results for all 44 instances. The accuracy of *logic component* identification is shown in Table 2. The results of logic pattern identification are obtained by aggregating the results of *logic component* identification (i.e., a logic pattern is identified only if all of its constituent *logic components* are successfully identified). We show that ***Decomp*** is more effective than ***Baseline*** in Table 1.

4.2 Analysis

For *agree portion*, we found that for 18/83 instances, model's generated reasoning is plausible despite its answer being different than the labels, and that whether the reasoning can be considered as correct depends on how implicit we interpret the CA. Furthermore, for *plausible portion*, for 43/44 instances, model's generated reasoning aligns with either one of the annotators' reasoning. In total, apart from the 123 instances where the model's answer agrees with both annotators', model's predictions for **61/127 instances can be considered plausible depending on the way we interpret the CA as well as the degree of implicit knowledge we incorporate into the decision process.**

5 Discussion

The plausible reasoning problem found in both annotation phase (Section 3.3) and model experiment phase (Section 4.2) poses a huge challenge for CALSA task regarding both obtaining a reliable dataset and subsequently training/evaluating the computational models. It is an extremely difficult problem as it is almost impossible to deterministically define the degree of implicitness required to interpret an argument. We argue that one potential way to alleviate the problem could be to hire a large number of people for annotations, similarly to [15], and subsequently consider the majority answer as the final answer. However, given the complexity of the CALSA task, this option would be time-consuming, require sophisticated training for annotators, and be financially costly. Therefore, how to effectively address the plausible reasoning problem could be a challenging yet interesting future work.

6 Conclusion

In this work, we explore addressing the CALSA task by decomposing the logic patterns into their constituent *logic components* and identifying each component one by one. We collect 250 annotations for *logic components* and use them for subsequent model experiments. Our findings in both annotation phase and modeling phase reveal that it is the various degrees of implicit reasoning involved in the identification process that renders the overall task challenging. We plan to address that in our future work.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Number 22H00524.

References

[1] Shoichi Naito, Wenzhi Wang, Paul Reisert, Naoya Inoue, Camélia Guerraoui, Kenshi Yamaguchi, Jungmin Choi, Irfan Robbani, Surawat Pothong, and Kentaro Inui. Designing logic pattern templates for counter-argument logical structure analysis. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, **Findings of the Association for Computational Linguistics: EMNLP 2024**, pp. 11313–11331, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[2] Douglas R. Smith. The design of divide and conquer algorithms. **Science of Computer Programming**, Vol. 5, pp. 37–58, 1985.

[3] Zepu Yi. Research on division and conquer algorithm. **AIP Conference Proceedings**, Vol. 2073, No. 1, p. 020086, 02 2019.

[4] Alon Talmor and Jonathan Berant. The Web as a Knowledge-Base for Answering Complex Questions. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, **Proceedings of the 2018 NAACL Linguistics: Human Language Technologies, Volume 1 (Long Papers)**, pp. 641–651, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[5] Sewon Min, Victor Zhong, Luke Zettlemoyer, and Hannaneh Hajishirzi. Multi-hop Reading Comprehension through Question Decomposition and Rescoring. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, **Proceedings of the 57th ACL**, pp. 6097–6109, Florence, Italy, July 2019. Association for Computational Linguistics.

[6] Ethan Perez, Patrick Lewis, Wen-tau Yih, Kyunghyun Cho, and Douwe Kiela. Unsupervised Question Decomposition for Question Answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, **Proceedings of the 2020 EMNLP**, pp. 8864–8880, Online, November 2020. Association for Computational Linguistics.

[7] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models, April 2023. arXiv:2205.10625 [cs].

[8] Andrew Drozdov, Nathanael Schärli, Ekin Akyürek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, and Denny Zhou. Compositional Semantic Parsing with Large Language Models, September 2022. arXiv:2209.15003 [cs].

[9] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed Prompting: A Modular Approach for Solving Complex Tasks, April 2023. arXiv:2210.02406 [cs].

[10] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023.

[11] Jieyi Long. Large language model guided tree-of-thought, 2023.

[12] Chikara Hashimoto, Kentaro Torisawa, Stijn De Saeger, Jong-Hoon Oh, and Jun'ichi Kazama. Excitatory or inhibitory: A new semantic orientation extracts contradiction and causality from the web. In Jun'ichi Tsujii, James Henderson, and Marius Paşca, editors, **Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning**, pp. 619–630, Jeju Island, Korea, July 2012. Association for Computational Linguistics.

[13] Paul Reisert, Naoya Inoue, Tatsuki Kuribayashi, and Kentaro Inui. Feasible annotation scheme for capturing policy argument reasoning using argument templates. In Noam Slonim and Ranit Aharonov, editors, **Proceedings of the 5th Workshop on Argument Mining**, pp. 79–89, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[14] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines, 2023.

[15] Yixin Nie, Xiang Zhou, and Mohit Bansal. What can we learn from collective human opinions on natural language inference data? In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, **Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)**, pp. 9131–9143, Online, November 2020. Association for Computational Linguistics.

Table 2 Accuracy of the identification of each *logic component*. G4o-mini: GPT-4o-mini.

Model	Ack	Miti	No evi	Not Pro(X, Y)	Pro(X, Z)	Pro(Y, Z)	Pro(Z, Y)	Suffi	Sup(X, Z)	Sup(Z, Y)	Avg
G4o-mini	.92	.84	.44	.72	.96	.96	.88	.68	.52	.44	.736

Table 3 The mapping between the original CA logic patterns defined in the CALSA paper and the *logic components* explored in this paper.

logic patterns	*logic components*
Mitigation	Acknowledge(IA), Mitigate(IA, Z)
Alternative	Acknowledge(IA), Sup(Z, Y)
No evidence	No evi(IA)
Another true cause	Not Pro(X, Y), Pro(Z, Y)
Missing mechanism #1	Pro(X, Z), Sup(Z, Y)
Missing mechanism #2	Pro(Z, Y), Sup(X, Z)
No need to address	Sufficient
Negative effect due to y	Pro(Y, Z), Good(Z)
Positive effects of a different perspective from y #1	Pro(X, Z), Good(Z)
Positive effects of a different perspective from y #2	Sup(X, Z), Bad(Z)

Table 4 The mapping between the *logic components* and the template for their corresponding binary questions. Contents within curly brackets will be filled in based on the actual IAs.

logic components	binary questions template
Acknowledge(IA)	*Does CA acknowledge that {IA's logic}?*
Mitigate(IA, Z)	*Does CA {causal relation stated in IA} can be mitigated by something?*
No evi(IA)	*Does CA argue that there is no evidence in the initial-argument to support {IA's logic}?*
Not Pro(X, Y)	*Does CA argue that {X} does not promote {Y}?*
Pro(X, Z)	*Does CA argue that {X} promotes something besides {Y}?*
Pro(Y, Z)	*Does CA argue that {Y} promotes something?*
Pro(Z, Y)	*Does CA argue that something besides {X} promotes {Y}?*
Sufficient	*Does CA argue that {Y} is not a problem that requires any action?*
Sup(X, Z)	*Does CA argue that {X} suppresses something?*
Sup(Z, Y)	*Does CA argue that something suppresses {Y}?*