Causal DAG Extraction from 3 Short Stories and 3 Movie Scripts (V2)

Robert R. Tucci
tucci@ar-tiste.com

March 7, 2024

Abstract

I improve an algorithm previously proposed by me for doing causal DEFT (DAG Extraction from Text), and then I apply the new algorithm to 2 usecases: 3 short stories by P.G. Wodehouse and 3 movie scripts by Pixar/Disney. The software used to accomplish this endeavor is called “Mappa Mundi” (MM) and is available as open source at GitHub. I discuss possible ways of improving the MM algorithm using LLMs (Large Language Models such as ChatGPT). I also travel this road in the opposite direction—I discuss possible ways of improving LLMs using the Mappa Mundi algorithm.

> *"If humans were so good at causal inference, religion would not exist."*
>
> - Yann LeCun, *Ref.[5]*

> *"How much of human knowledge is captured in all the text ever written? To which my answer is: not much. "*
>
> - Yann LeCun, *Ref.[4]*

1 Introduction

In this paper, I improve an algorithm for doing causal DEFT (DAG Extraction from Text) that was first proposed by me in Ref.[9] I then apply the new algorithm to 2 usecases:

1. 3 short stories by P.G. Wodehouse (the text for these was obtained from the Project Gutenberg website Ref.[2])
    - Bill the Bloodhound
    - Extricating Young Gussie
    - Wilton's Holiday
2. 3 movie scripts by Pixar/Disney. (the text for these was obtained from the IMSDb website Ref.[1])[1]
    - Toy Story
    - Up
    - WALL-E

The Python software that was used to accomplish this endeavor is called Mappa Mundi (MM). It is open source and available at GitHub (Ref.[10]).

So what is MM good for? The goal of DEFT in general and MM in particular, is to create a directory of DAGs ("DAG atlas"). Conjecturing a DAG is always the first step in doing Judea Pearl's causal inference (CI).[2] Once a DAG is available, one can use it to do Pearl's 3 rungs of CI, using tools such as SCuMpy.[3][4]

---

[1] The Mappa Mundi repo at GitHub contains a Python script called `downloading.py` that uses the BeautifulSoup Python package to scrape all the 1100+ movie scripts available (about 230 MB) at the IMSDb website. My original intention was to apply my algorithm to all of those movie scripts. However, due to lack of hardware resources, I had to settle for just 3 movie scripts.

[2] Judea Pearl's CI is described in detail in Pearl's *The Book of Why* (Ref.[6]) and in my free, open source book *Bayesuvius* (Ref. [8]).

[3] DAG = Directed Acyclic Graph, SCM = Structural Causal Model, a type of DAG

[4] Shameless plug: my free, open source software SCuMpy (Ref.[11]) can be used to do all 3 rungs of CI. It can handle all types of linear SCM, including SCM with feedback loops and hidden variables.

As I explained in my previous paper Ref.[9], the scientific method (SM) looks for causation, not correlation. Pearl CI is the gold standard theory for distinguishing between correlation and causation. Hence, the SM and Pearl CI are closely related. Pearl CI can be viewed as an application of the SM, wherein the DAG is the hypothesis part of the SM—what we want to prove or disprove. DEFT provides DAG hypotheses. For example, DEFT could be used to discover causal DAGs that indicate pathways to diseases.

At the end of this paper, I discuss possible ways of improving the MM algorithm using LLMs (Large Language Models such as ChatGPT). I also travel this road in the opposite direction— I discuss possible ways of improving LLMs using the MM algorithm.

2 MM Algorithm Overview

The MM algorithm proposed in this paper can be applied to a broad range of texts. The only constraint is that those texts do "story-telling" in a chronological order. That is why I decided to use movie scripts, because movie-scripts usually do story-telling in a chronological order (except when they do flashbacks or time travel, but that isn't very common in movies.)

The MM algorithm would not work well if applied to the corpus of science papers at arXiv, because scientific papers normally don't do chronological story-telling. On the other hand, it might work well if applied to a corpus of (time stamped) lab notebooks maintained by one or more experimental scientists. It might also work well on a corpus of time-stamped logs maintained by one or more person trying to figure out the cause of a disease.

The MM algorithm would also work well on a corpus of videos or movies that do chronological story telling, even if the movie scripts were unavailable. This would require that some human or AI narrated the movie/video as the action happened. Such "in-time" movie and video narration, often referred to as AD (audio description) (Ref.[13]), and also closed captioning, are becoming increasingly widespread in the movie, TV and internet video streaming industries. In certain cases, they are mandated by laws (like the CVAA, Twenty-First Century Communications and Video Accessibility Act of 2010) that address the needs of persons with disabilities.

The essence of the MM algorithm can be described as follows.

Given a set of N movie scripts (or short stories), the algorithm compares all possible pairs of movies. Hence, it makes $\frac{N^2-N}{2}$ movie pair comparisons.

Before comparing movies, each movie script is simplified as follows. Each complex or compound sentence is split into simple sentences (ssents).[5] Then we define a node for each ssent, and collect those nodes to form a DAG for each movie.

---

[5] Ssents have a single subject and verb. They are essentially the same as the (subject, relation, object) triples used to construct "knowledge graphs".

To compare two movies 1 and 2, we compare every node `node1` in movie 1 with every node `node2` in movie 2. To compare two nodes (`node1`, `node2`), we compute what is called, in the NLP (Natural Language Processing) field, a **similarity between two sentences**. If the similarity between `node1` and `node2` exceeds a certain threshold that we call `SIMI_THRESHOLD`, then we store that node pair and its similarity in a dictionary with a node pair as key and its similarity as value. Call this dictionary `nd1_nd2_bridges`. We say that there is a **bridge** between `node1` and `node2` if (`node1`, `node2`) is contained in the keys of `nd1_nd2_bridges`.

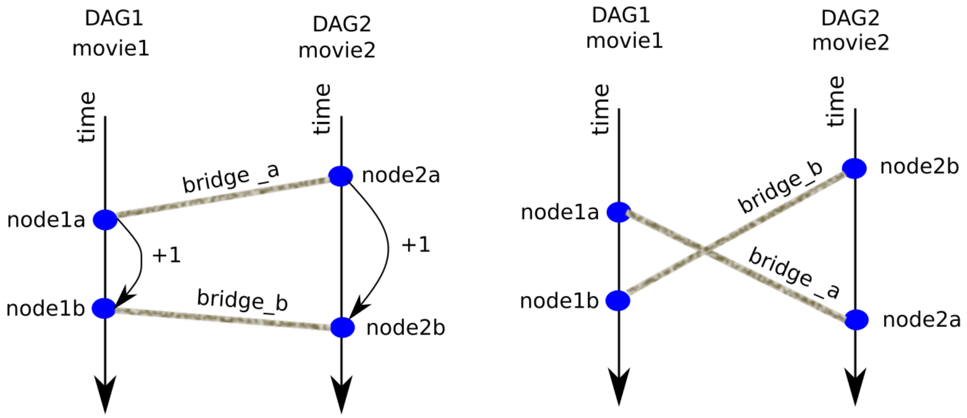

Figure 1: Bridges span two DAGs (i.e., movies). We consider 2 possibilities: bridges a and b cross, or they don't.

Next we consider every pair $\{a, b\}$ of bridges. Suppose bridge a connects node `node1a` in movie 1 to node `node2a` in movie 2. Likewise, suppose bridge b connects `node1b` in movie 1 to node `node2b` in movie 2. Let `node1a.time` be the time at which `node1a` occurs and define `node1b.time`, `node2a.time`, and `node2b.time` similarly. Assume that `node1a.time` < `node1b.time`. Then there are two possibilities that we wish to consider. These 2 possibilities are illustrated in Fig.1. Either the bridges don't cross (i.e. a occurs before b in both movies) or they cross (i.e. a occurs before b in movie 1 but after in movie 2). Let N_{rep} be the **number of repetitions of an arrow**. If bridges a and b cross, we do nothing. If they don't cross, we do the following for both DAG1 and DAG2. If an arrow between the earlier and latter of the two nodes doesn't already exist, we add the arrow with $N_{rep} = 1$. If such an arrow already exists, we increase its N_{rep} by one.

That's basically the whole algorithm. At the end of it, we will have generated DAG1 for movie 1 and DAG2 for movie 2.

When drawing one of those DAGs, one specifies a number `reps_threshold`$= N^*_{reps}$. Only the arrows with $N_{reps} > N^*_{reps}$ are drawn. The number N_{reps} for each arrow is drawn in the middle of the arrow.

Here is a simple argument for why this algorithm should work. Consider Fig.2. The figure depicts a DAG that expresses the fact that both shark attacks and

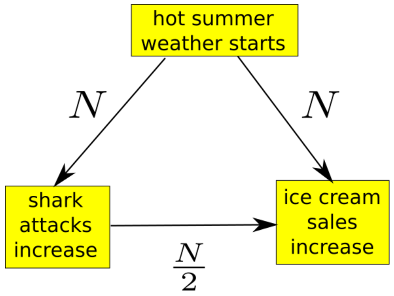

Figure 2: DAG that expresses the fact that both shark attacks and ice cream sales increase during the summer, because both are caused by hot summer weather.

ice cream sales increase during the summer, because both are caused by hot summer weather. Let

H = hot summer weather starts,
Sh = shark attacks increase,
IC = ice cream sales increase.

If we compare this DAG to N other DAGs that contain these 3 nodes, then, since it is always true that summer precedes shark attack increases and ice cream sales increases, the two arrows emanating from the H node will have $N_{reps} = N$. On the other hand, we expect that half of the time, the Sh node will occur before the IC node, and half of the time it will happen after. Hence, $N_{reps} = \frac{N}{2}$ for the arrow $Sh \rightarrow IC$.

Let `reps_threshold` $= N^*_{reps}$. Select N^*_{reps} such that $\frac{N}{2} < N^*_{reps} < N$. If when we draw this DAG, we only draw arrows with $N_{reps} > N^*_{reps}$, then only the two causal arrows will be visible. The difference between N_{reps} for the causal and non-causal arrows (call it the **repetition gap**) will grow as $\frac{N}{2}$.

It's important to point out that I expect this algorithm to produce good DAGs only after a large number N of DAGs (i.e., movie scripts or short stories) are compared. That's because the repetition gap grows relatively slowly, just linearly in N. Since, due to lack of hardware resources, this paper only compares a minuscule $N = 3$, one can't expect very dramatic causal revelations from this paper.

It's also important to point out that every time a new movie script is added to the list of the N already analyzed movie scripts, that new movie script must be compared to the preceding N movie scripts. If we imagine a robot watching a movie daily, then his daily dream time, if used solely for movie comparison, would grow linearly in time. At some point, it would take more than a day of dream time to compare today's movie to all movies in its past. To keep his dream time constant, at 8 hours a day, the robot would have to compare today's movie to a fixed number, say the most recent 365 movies viewed previously.

This section has described how the original version of MM did DEFT. More recent versions of MM do DEFT slightly differently. For each arrow, the new versions store two weights n_{acc} and n_{rej} instead of just one weight N_{reps}. As we explain in

Appendix A, this 2 weight per arrow DEFT is stricter and more specific.

3 Software Description

The full MM process applied to the 3 short stories, is documented in the jupyter notebook

`jupyter_notebooks/navigating_short_stories.ipynb`

The full MM process applied to the 3 movie scripts, is documented in the jupyter notebook

`jupyter_notebooks/navigating_m_scripts.ipynb`

For a detailed description, encompassing every method and variable used in the MM software, please consult the software's Python "docstrings". This section of the paper merely presents a brief overview of what you will find in those jupyter notebooks and docstrings.

The full process from raw data to DAG atlas is broken down in MM to performing the following steps. Most of my time programming was spent on the python scripts that do pre-processing of the data (i.e., steps 2, 3 and 4 below). Pre-processing data is hard!, and not doing it is fatal to this algorithm. When evaluating the similarity between two ssents, even small misspellings can change that value substantially.

1. Data scraping using methods in `downloading_imsdb.py`.

   This python script was only used for the movie scripts, not for the short stories. It scrapes all the movie scripts from the IMSDb website using the Python package Beautiful Soup.

2. Cleaning using methods in `cleaning.py`

   Here I remove contractions like "didn't", and replace exclusively unicode symbols by their closest ANSII analogues (e.g., curly quotes are replaced by straight quotes).

   Here I also use the software SpaCy[6] to break up the movie script into separate sentences, and return a file with only one sentence per line.

   For the case of movie scripts (but not for short stories), I also try to distinguish between dialog lines and narration lines. In many but not all movie scripts, the dialog lines are indented with respect to the narration lines. In the case of Pixar/Disney, they don't indent dialog. In cases where the movie script

---

[6] In the Python world, there are 2 general, dominant NLP libraries, SpaCy and NLTK (Natural Language Tool Kit). MM uses both. There is much overlap between the 2. In case of overlap, I tried to use the one that was fastest.

indents, the MM software gives the option of throwing away all the dialog lines and keeping only the narration ones.

3. Spell-checking using methods in `spell_checking.py`

   I discovered, to my chagrin, that spell-checking without any input from a human user, is very error prone, unless one takes context into consideration. Certain types of LLMs can do contextual spell-checking, but this project did not use them. Instead of a contextual spell-checker, I used the non-contextual spell-checker `pyspellchecker` that tries to replace infrequent words by more frequent ones (a very risky and error prone approach). To diminish the risk, I constrained it in various ways so that it only makes very conservative corrections.

4. Simplifying using methods in `simplifying.py`

   At this point, the file has only one full sentence per line. Here, I use Openie6 to break each full sentence into ssents.[7] Each line with a full sentence is replaced by all the ssents derived from the full sentence. The ssents are separated by a separator-token (`ZTZ_SEPARATOR`).

   Each ssent becomes a node of the DAG.

   If a ssent (i.e., node) appears at the row t of the file (counting starting with 0), then we say that node occurs at time t. If a ssent appears after zero separator-tokens, we say $x = 0$ for it. If it appears after one separator-token, we say $x = 1$ for it, and so forth. Hence each node (i.e., ssent) can be labeled by its (t, x) coordinates.

5. Creating DAG atlas using methods in `DagAtlas.py`

   Everything up to this point has just been pre-processing of the data. Here, I finally implement the algorithm described in Section 2. The bottleneck and rate-determining-step for the full MM process is calculating the similarity between 2 nodes. If DAG1 for movie script (or short story) 1 has k_1 nodes, and DAG2 has k_2 nodes, then k_1k_2 similarity calculations have to be made. For the short stories I considered, k_1k_2 is typically on the order of $0.1M$. For the movies, it is typically $1.5M$.

   For the similarity measure between nodes (i.e., ssents), I tried many methods. I ended up using sBERT (Ref.[7]).

6. Visualizing DAGs using methods in `Dag.py`.

   This step is easy. I use `graphviz` to accomplish it.

---

[7] Alternatively, one can use my software SentenceAx. SentenceAx is a full rewrite of Openie6. Openie6 and SentenceAx are both available at GitHub (Refs.[3], [12])

4 Possible Improvements of the MM algorithm using LLMs

MM uses LLMs twice: first, it uses Openie6 to do sentence spitting, and second, it uses sBERT to calculate sentence similarities. Openie6 and sBERT are both fine tunings of BERT. BERT is the encoder part of the Vanilla Transformer Network first proposed in 2017 in the famous "Attention is all you need" paper.[8]

MM does not, however, use a promptable LLM such as ChatGPT.[9] But it could use one. A promptable LLM could perform the 3 data pre-processing steps (2-cleaning, 3-spell-checking and 4-simplifying) in lieu of Openie6. It could also calculate sentence similarities in lieu of sBERT.

To demonstrate that promptable LLMs can perform these task superbly, I did the following test. I gave ChatGPT the beginning narrative paragraph of the movie WALL-E, and I asked her to summarize the paragraph as a list of simple declarative sentences. Here is the result:

PROMPT:

Consider the following quote:

> "A range of mountains takes form in the haze. Moving closer. The mountains are piles of TRASH. The entire surface is nothing but waste.
>
> "..We're gonna find adventure in the evening air..."
>
> A silhouetted city in the distance. What looks like skyscrapers turns into trash. Thousands of neatly stacked CUBES OF TRASH, stories high. Rows and rows of stacked cubes, like city avenues. They go on for miles."

Give an enumerated list summarizing this quote. Each list item must be a simple declarative sentence consisting of a subject phrase and a verb phrase. The list items must be in chronological order of occurrence in the quote.

REPLY:

1. A range of mountains appears in the haze.
2. The narrator moves closer.
3. The mountains are revealed to be piles of trash.
4. The entire surface is described as nothing but waste.

---

[8] For more information about Transformer Networks, see my book Bayesuvius (Ref.[8]) which has a long chapter dedicated to them.

[9] Promptable LLMs use the full (i.e., encoder and decoder) Vanilla Transformer Network, whereas BERT only uses the encoder part.

5. The narrator speaks of finding adventure in the evening air.
6. A silhouetted city appears in the distance.
7. The skyscrapers turn out to be made of trash.
8. Thousands of neatly stacked cubes of trash are seen, stories high.
9. Rows and rows of stacked cubes are seen, resembling city avenues.
10. The expanse of trash goes on for miles.

Each of these 10 sentences would make a superb node in a causal DAG. With nodes defined with such clarity, one could avoid much noise in the calculation of the similarity of 2 nodes.

5 Possible Improvements of LLMs using the MM algorithm

LLMs are very good at what they do, but ultimately, they are just curve fitters that cannot perform the scientific method (SM) and causal inference (CI) in a deliberate way. At best, they can perform the SM in a trial an error way, as in Fig.3. This weakness of LLMs can be overcome by adding to them a DAG atlas and an explicit (not an implicit or emergent) CI engine.

A Appendix: Two weights per arrow instead of one

In the original MM version, each arrow carries a single weight N_{reps} initially set to 0. N_{reps} = the number of repetitions of that arrow. When we compare two bridges,

- If the bridges don't cross, we increase by 1 the N_{reps} for the arrow between points a and b in movie 1, and for the corresponding arrow in movie 2.
- If the bridges cross, we do nothing.

We then define a threshold value N^*_{reps} for N_{reps}. When drawing the DAG, we only draw arrows with $N_{reps} > N^*_{reps}$. Each arrow in the DAG looks like

$$A \xrightarrow[5]{} B$$

when $N_{reps} = 5$

In the new version of MM, each arrow carries a pair of weights (n_{acc}, n_{rej}) initially set to $(0, 0)$. n_{acc} = the number of acceptances. n_{rej} = the number of rejections.

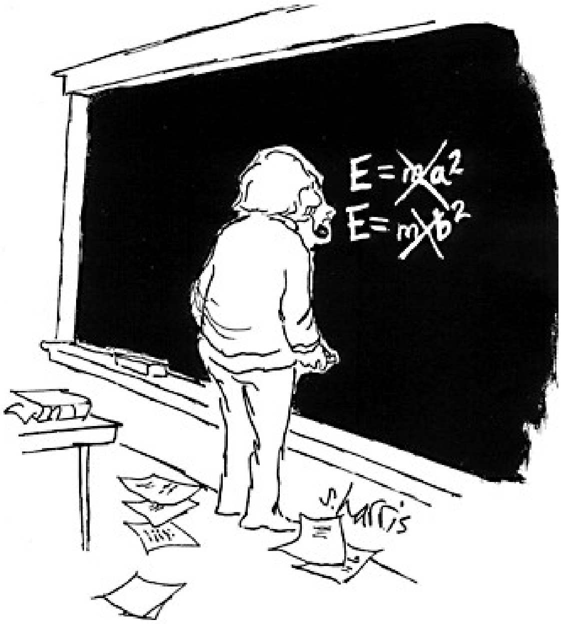

Figure 3: Cartoon by S.Harris about $E = mc^2$.

- If the bridges don't cross, we accept. We increase n_{acc} by 1, for the arrow between bridges a and b in movie 1, and for the corresponding arrow in movie 2.
- If the bridges cross, we reject. We increase n_{rej} by 1, for the arrow between bridges a and b in movie 1, and for the corresponding arrow in movie 2.

We define the acceptance probability p_{acc} for each arrow by

$$p_{acc} = \frac{n_{acc}}{n_{acc} + n_{rej}} \tag{1}$$

We also define the total number N of samples for each arrow by

$$N = n_{acc} + n_{rej} \tag{2}$$

Then we define a threshold value p^*_{acc} for p_{acc} and a threshold value N^* for N. When drawing the DAG, we only draw arrows with $p_{acc} > p^*_{acc}$ and $N > N^*$. Each arrow in the DAG looks like

$$\begin{tikzcd} A \arrow[r, "0.93(5)"'] & B \end{tikzcd}$$

when $p_{acc} = 0.93$ and $N = 5$.

Fig.4 shows a picture of the (n_{acc}, n_{rej}) plane. We assume that p^*_{acc} is fairly close to 1 so the line $\frac{n_{acc}}{n_{acc}+n_{rej}} = p^*_{acc}$ is close to being horizontal. The purple region satisfies the 2 constraints $p_{acc} > p^*_{acc}$ and $N > N^*$.

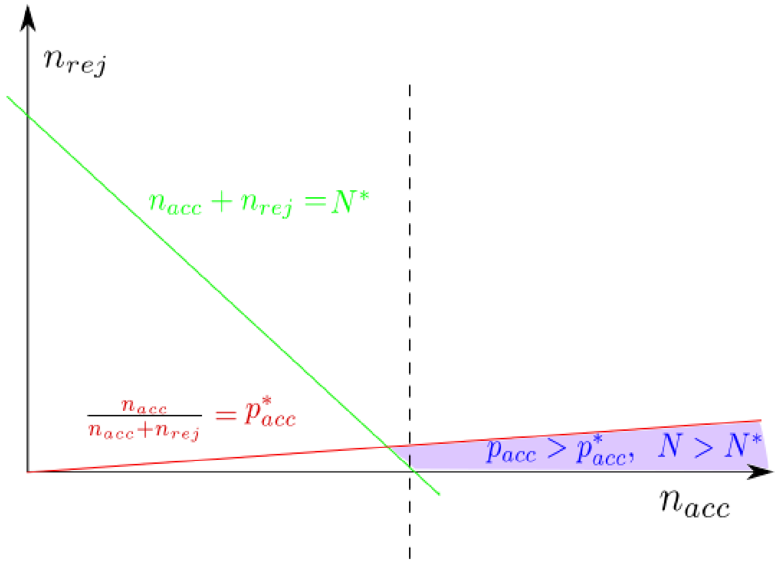
n_{rej}
$n_{acc} + n_{rej} = N^*$
$\frac{n_{acc}}{n_{acc}+n_{rej}} = p^*_{acc}$
$p_{acc} > p^*_{acc}, \quad N > N^*$
n_{acc}

Figure 4: (n_{acc}, n_{rej}) plane. The purple region satisfies the 2 constraints $p_{acc} > p^*_{acc}$ and $N > N^*$.

The two weights per arrow (2W/A) method gives bnets with fewer but more trustworthy arrows, than the one weight per arrow (1W/A) method. This can be easily seen from Fig.4. The N_{reps} in the 1W/A method corresponds to the n_{acc} in the 2W/A method. Thus, the threshold N^*_{reps} for N_{reps} becomes a threshold n^*_{acc} for n_{acc}. The set of points satisfying $n_{acc} > n^*_{acc}$ equals a half plane H in (n_{acc}, n_{rej}) space. For instance, if n^*_{acc} is the value of n_{acc} for the dashed line in Fig.4, then H equals the half plane of all points to the right of that dashed line. Clearly, H contains many more points than the purple region in Fig.4.

Hence, one is justified in saying that the 2W/A method is stricter and more specific (i.e., more restrictive) than the 1W/A method. It is also less lossy. When the 1W/A method does nothing for crossing bridges, it is throwing away some information that the 2W/A method keeps. This extra information is used to filter out less trustworthy points with high n_{rej} counts (high compared to n_{acc})

References

[1] Internet Movie Script Database (IMSDb). `https://imsdb.com/`.

[2] Project Gutenberg website. `https://www.gutenberg.org`.

[3] dair iitd. Openie6. `https://github.com/dair-iitd/openie6`.

[4] Yann LeCun. Twitter, absurd statement about all the text ever written. `https://twitter.com/ylecun/status/1562137291845521408`.

[5] Yann LeCun. Twitter, absurd statement about causal inference and religion. `https://twitter.com/ylecun/status/1577128801620070400`.

[6] Judea Pearl and Dana Mackenzie. *The book of why: the new science of cause and effect*. Basic books, 2018.

[7] sbert.net. sBERT. `https://www.sbert.net/`.

[8] Robert R. Tucci. Bayesuvius (book). `https://github.com/rrtucci/Bayesuvius/raw/master/main.pdf`.

[9] Robert R. Tucci. Causal dag extraction from a library of books or videos/movies. `https://arxiv.org/abs/2211.00486`.

[10] Robert R. Tucci. Mappa Mundi at github. `https://github.com/rrtucci/mappa_mundi`.

[11] Robert R. Tucci. SCuMpy at github. `https://github.com/rrtucci/scumpy`.

[12] Robert R. Tucci. SentenceAx at github. `https://github.com/rrtucci/SentenceAx`.

[13] Wikipedia. Audio description. `https://en.wikipedia.org/wiki/Audio_description`.