Reproducibility Report for CSE 481N

Griffin Golias, Alex Dundarov, Charles Immendorf, Eric Yeh
Team 4 - "Paul G. Allen's Card"
{goliagri, alexd02, chazi, yehe} [at] uw [dot] edu

May 31, 2023

1 Introduction

We attempt to reproduce the results of *Logical Fallacy Detection* (Jin et al. 2022)[2]. Our results significantly differ from the ones presented in the original paper, and additional experiments we conduct indicate deficiencies in the models.

Jin et al. creates 2 new datasets for a logical fallacy detection task. Given a short statement, a model is tasked with predicting which of 13 logical fallacy classes is present. The authors introduce a general LOGIC dataset and a themed LOGICCLIMATE dataset for evaluating models on the logical fallacy detection task. An NLI model is used with the statement as the premise and a hypothesis that this is an example of a given logical fallacy. The authors propose a "Structure Aware" (StructAware) modification to a pretrained Electra [1] model which applies a masking to the premise to emphasize its logical form and uses a hypothesis asking if the logical form matches a fixed base logical form for each fallacy. The authors report that the StructAware version of Electra outperforms the base version by a micro F_1 increase of about 5% fine-tuned on LOGIC and evaluated on LOGIC as well as LOGICCLIMATE, and 6% fine-tuned on LOGIC and LOGICCLIMATE and evaluated on LOGICCLIMATE.

2 Scope of Reproducibility

Jin et al. finds that out of the fine-tuned, pre-trained language models tested, Electra performed the best. They go on to compare the standard Electra model with their modified StructAware version, which first attempts to extract the logical form of a statement before Electra evaluates it. Electra and Electra StructAware are evaluated for accuracy, precision, f1, and recall, over LOGIC fine-tuned on LOGIC, LOGICCLIMATE fine-tuned on LOGIC, and LOGICCLIMATE fine-tuned on LOGIC and LOGICCLIMATE. The paper reports the StructAware modification significantly increases performance across all metrics and tasks. The authors only saved the Electra and Electra StructAware models. Since the other models are no longer available, and Electra was the primary focus of the original paper, we have focused solely on reproducing the metrics of the Electra models on each task.

We aim to test the following hypotheses in our reproduction:

1. The Electra Models saved by the authors achieves results similar to those presented in the paper (see Table 1 for exact expected and reproduced results).
2. The Electra StructAware Models saved by the authors achieves results similar to those presented in the paper (see Table 1).
3. For models concerned by hypotheses 1 and 2 that fail to achieve the paper's metrics, retrained versions of those models achieve those metrics.
4. Electra StructAware consistently outperforms standard Electra on all logical fallacy classification tasks.

3 Methodology

3.1 Model Descriptions

Jin et al. uses the standard mnli-Electra implementation as well as a modified StructAware model which adds new elements to the processing pipeline without directly modifying the architecture of the Electra model itself. Electra is a BERT-based language model with 110M total parameters. The main novel element Electra introduces is the replaced token detection training task, where the model learns to distinguish between naturally occurring and synthetic tokens in a text rather than generating tokens to fill masks. The same pre-trained Electra model is used directly and modified with the StructAware wrapper, then fine-tuned on the respective datasets. In either case an NLI method is used to perform the classification. In the non StructAware case, the statement to be classified is used as the premise and "This is an example of " plus a particular logical fallacy type is used as the hypothesis.

The Electra-StructAware model adds additional steps modifying the statement before classifying it with the Electra model.

The structure-aware premise modifies the original statement through a series of steps. First co-reference resolution is used to group words which refer to the same object. Then lemmatization is applied to recover the Canonical forms of each word and an embedding is generated using Sentence-BERT. Cosine similarities are calculated between the representations and if it exceeds a threshold (0.7) they are considered similar. Finally if a sequence of words is found to match another sequence of words in similarity, they are grouped and each group is replaced by a mask of the same token. The ultimate effect is to mask content words and distill the statement's structure, intending to bring out its logical form.

The structure-aware hypothesis consists of a statement predicting the logical fallacy type of the structure-aware premise. The name of the predicted label in the hypothesis statement is replaced by the logical form of the statement. For example, instead of passing in "This is an example of deductive fallacies", the structure-aware model uses "The example matches the logical following form: 'If [MSK1], then [MSK2]' leads to 'If [MSK2], then [MSK1]'"

3.2 Dataset description

We utilize 2 different datasets for training and evaluation: LOGIC and LOGICCLIMATE, both split into train, dev, and eval sets and can be found in the codebase. The links to the articles from these datasets can be found on the original Github project repository.

LOGIC is a dataset created by lifting statements containing 13 specific types of logical fallacies from various educational sources, such as Quizziz, study.com, and ProProfs, as well as from various sources found manually by searching on Google. There are a total of 2449 logical fallacy instances split among the 13 classes, with an average of 29.02 tokens per sample and a 7,624 word vocabulary. LOGIC is unevenly distributed, with the most common fallacy (faulty generalization) encompassing 18.01% of the samples, and the least common fallacy (Equivocation) 2%.[1]

LOGICCLIMATE is similarly a dataset consisting of statements exemplifying a logical fallacy, but additionally themed to deal specifically with climate change related topics. Examples were manually collected from climate change news articles on the Climate Feedback website by native English speaking annotators. The annotators selected specific text spans within the article and assigned a logical fallacy type label to each. LOGICCLIMATE has 1,079 total samples with on average 35.98 tokens per sample and a vocabulary size of 5,800 words. The classes are even more unevenly distributed, with the most common (Intentional Fallacy) at 25.58% and the least common (Circular Claim) at 0.51% of the population.[2]

3.3 Hyperparameters

Our results were achieved using the same hyperparameter specifications as the original paper. We train using a fixed learning rate of 2e-5 and a batch size of 32 with the AdamW optimizer. A maximum number of epochs of 10 was used but the model was trained to convergence in 5 or less each time.

One hyperparameter we experimented with was the threshold that determines if the given sample is marked as a positive for the given class. Note that the original paper implicitly had a fixed threshold of 0.5. Since this hyperparam-

[1] Frequencies of each class are given in table 6 of Jin et al [2]

[2] Frequencies of each class are given in table 4 of Jin et al [2]

eter has an impact on the precision and recall values, we finetuned the threshold at different values from 0.01 to 0.99 with 0.01 increments to see if this could improve the metrics we received.

3.4 Code

We built on top of the author's existing codebase to run our experiments. Our fork of the authors' GitHub repository can be found here. The original repository is here. Our main documentation is in the `reproduction_notes.md` file located in the top-level directory of our forked repository. It details how to set up the environment, how to run the various experiments (obtaining models, training, evaluation, plotting, etc.), more details on the various experiments, where experiment results are stored, etc.

The main modules that we used (and modified) from the author's code are `logicedu.py` and `logicclimate.py`, which train/evaluate models (with calculation of metrics) on the LOGIC and LOGICCLIMATE datasets, respectively.

3.5 Experimental Setup

Most of our retraining (i.e., using `electra-small-mnli`) and evaluation experiments were run on the NLPG machines using the commands documented in `reproduction_notes.md`. For retraining using `electra-base-mnli`, we used a Google Colab notebook, connected to a T4 GPU runtime. The notebook contained the `logicedu.py` code needed to train and evaluate the model. The authors' saved Electra models are in a Google Drive linked on the original repository, but we decided to use retrained models too due to not all their results matching what was in the paper.

3.6 Computational Requirements

The original paper reports that all models were trained only once using a single GPU for under two hours. Most of the reproduced experiments can be completed in less than an hour on a single GPU. Retraining the Electra-StructAware model, from `electra-base-mnli`, on the LOGIC dataset is more expensive, taking approximately 20 minutes to train each epoch for a total of 5 epochs. The original authors reported using 32GB of memory to account for the large memory usage of the training task. We estimated that a minimum of 12GB of memory was needed to complete training, which exceeded the allocation limit of 10GB on the NLPG machines. As a result, when training from `electra-base-mnli`, we instead transferred our code to a Colab notebook, which provided a GPU runtime with a 14GB memory limit, which was sufficient for such retraining.

4 Results

We first try to reproduce the paper's results using the authors' saved models, which are available in their repository, here. There are four models: two are just trained on LOGIC, and the other two are trained on LOGIC and further finetuned on LOGICCLIMATE. One model from each of these groups of two is StructAware. Using the authors' saved models, we were successfully able to reproduce the results of the Electra Non-StructAware models, getting fairly similar results to that of the original paper (see Table 1). However, we have found that the results of evaluating the saved Electra StructAware models do not reflect the results reported in the original paper.

The rest of this section details our efforts to try and get StructAware Electra models to match the corresponding metrics from the paper. In summary, by using different models (some being retrained (see sections 4.3 and 4.4)), and by changing the thresholds at which models consider examples as having given fallacies (see section 4.2), we obtained metrics that are close to, but not quite, the paper's metrics. Table 2 shows these metrics, meant to be compared with the paper's metrics for StructAware models on Table 1. Obtaining these metrics require picking different models for different evaluation types (e.g. Table 2's columns), and, for each model, picking the threshold based on test-set evaluations (the threshold-testing precision-recall plots). Whereas the paper originally did evaluations on the LOGIC and LOGICCLIMATE datasets using the same models (i.e., each row of the first two columns of Table 1) and same (implicit) thresholds. Because of this, it could be of the question of whether the StructAware metrics were really reproduced.

In terms of the respective hypotheses from section 2,

1. We managed to get similar results in the paper for the authors' saved non-StructAware Electra models, without having to do threshold testing.

Electra		Eval'd on LOGIC		LOGICCLIMATE		LOGICCLIMATE (f.t.)	
		Saved Model	Paper's	Saved Model	Paper's	Saved Model	Paper's
Non-S.A.	Micro F_1	50.58	53.31	22.72	22.72	29.21	23.71
	Precision	51.23	51.59	18.68	18.68	19.49	20.86
	Recall	80.33	72.33	35.95	35.85	66.29	23.09
	Accuracy	33.33	35.66	–	–	–	–
StructAware	Micro F_1	41.20	58.77	25.65	27.23	0.0	29.37
	Precision	37.79	55.25	18.51	20.46	0.0	17.66
	Recall	71.00	63.67	39.88	45.12	0.0	67.22
	Accuracy	18.00	47.67	–	–	–	–

Table 1: Comparison of saved model's performance metrics vs. metrics stated in the original paper. The "Eval'd on LOGIC", "LOGICCLIMATE", and "LOGICCLIMATE (f.t.)" columns have metrics for the Electra models evaluated on the LOGIC, LOGICCLIMATE, and LOGICCLIMATE dataset, respectively. For "LOGICCLIMATE (f.t.)", the Electra model used was further finetuned on the LOGICCLIMATE dataset. The "StructAware" row is for the StructAware version of the Electra model, and "Non-S.A." is for the non-StructAware version. Note that the original paper omits accuracy values for LOGICCLIMATE evaluations since it is a multi-label classification.

Electra		Eval'd on LOGIC	LOGICCLIMATE	LOGICCLIMATE (f.t.)
StructAware	Model used	Retrain `base` Electra (e.s.)	Authors' Saved Model	Retrain `small` Electra; (b.o. 20)
	Threshold used	0.84	0.32	0.20
	Micro F_1	58.11	26.51	26.22
	Precision	52.53	17.76	16.77
	Recall	60.33	51.05	69.10
	Accuracy	45.00	–	–

Table 2: The best performances gotten from StructAware models, in order to try to reproduce the paper's StructAware metrics. To get these metrics, the threshold-testing precision-recall plots (which have data points for different thresholds tried) for StructAware model evaluations were looked at. For each type of evaluation (table column), the model with data points closest to the paper's metric was chosen. Among those closest data points, the one with the highest Micro F_1 score was taken. See section 4.2 for more details on this threshold testing. ("e.s." is "early stopping", and "b.o. 20" means "best of 20 epochs")

2. With the right threshold, we got a result close to the paper's metrics for the StructAware Electra model (not finetuned on LOGICCLIMATE) evaluated on LOGICCLIMATE, but not with the other StructAware model evaluations.

3. The saved models fail to meet the paper's metrics for the StructAware Electra model evaluation on LOGIC, and for the StructAware Electra model (finetuned on LOGICCLIMATE) evaluation on LOGICCLIMATE, even after threshold testing. Retrained models using `electra-base-mnli` and `electra-small-mnli`, respectively, gets metrics close to (but not quite matching) the paper's metrics. See Table 2, section 4.3, and section 4.4 for more details on these models.

4. We cannot say that the StructAware models consistently outperform the non-StructAware models. There are some metrics that the non-StructAware models from table 1 still outperform the StructAware models from table 2 on: such as the recall when evaluating on LOGIC, etc. And the former's thresholds aren't necessarily optimally chosen, while the latter's are.

4.1 Evaluation Metrics

Table 1 contains the results we reproduced compared to the ones reported in the original paper. There are six evaluations in it. Table 2 summarizes our efforts to reproduce the StructAware metrics in Table 1, and contain the best

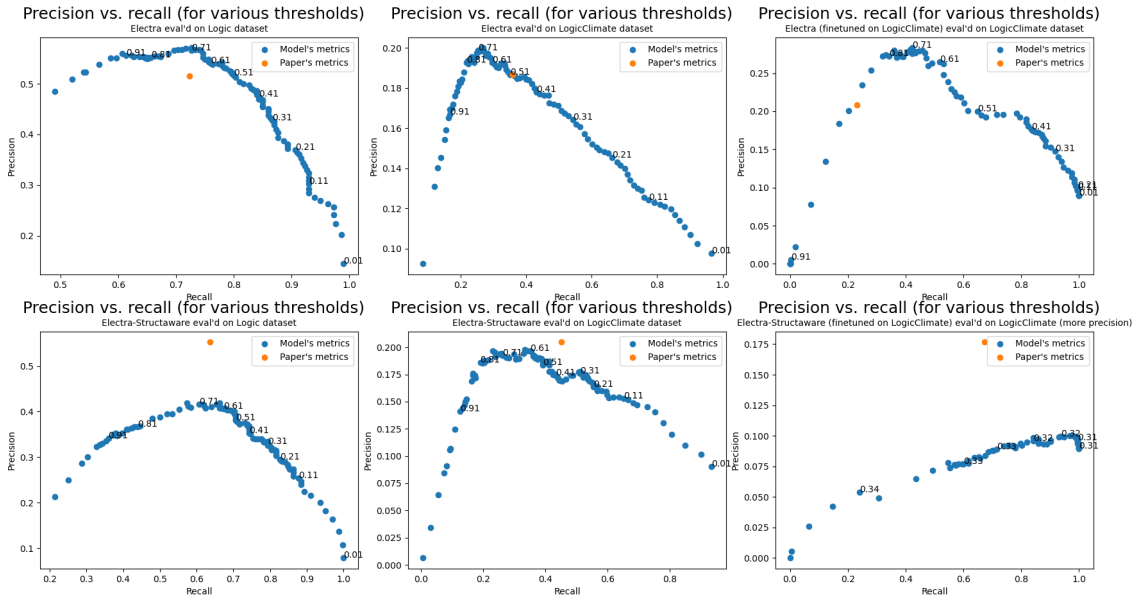

Figure 1: Precision-recall curves for thresholds 0.01 to 0.99 (0.01 increment) for classifying a sample as having a given logical fallacy. Each element in this grid of plots corresponds to the model evaluation in the same grid position in Table 1. Note that for the Electra-Structaware model further finetuned on LogicClimate (bottom-right), the model gave degenerate outputs for threshold values not between 0.3065 and 0.3377, so its plot is with more precise threshold values instead of the normal . Full-size versions of these graphs can be found here (see the "File/directory name abbreviations in `threshold_plots`, etc. folders" section of `reproduction_notes.md` for file name information).

metrics we've gotten for each evaluation, as well as which model those metrics came from.

4.2 Thresholds Testing - Precision and Recall

The output of our network is a real number between 0 and 1. A threshold is used such that if the network's output is above the threshold, a positive label is predicted and otherwise a negative label is predicted. We believe the authors used a threshold of .5 to achieve their results as that is the default in the code, but it is not explicitly mentioned in the paper itself. Because our model was not reaching the desired results, we decided to examine the full range of possible values for this threshold parameter. `reproduction_notes.md` (in the "Other notes" section) has more details on how the threshold values are changed, and how the model made predictions originally.

For each model evaluation we reran it over a range of thresholds (0.01 to 0.99 with 0.01 increments). We plotted the precision and recall values for the evaluations at these different thresholds, along with the paper's reported precision and recall for the corresponding task. The rationale is that if the paper's precision and recall is "captured" by the curve made up of the precision-recall data points (the "precision-recall curve")—that is, if the paper's data point lies on or below the precision-recall curve, or any data point on the curve has better precision and recall than the paper's data point—then we've reproduced the paper's metrics for that evaluation.

In Figure 1, we see that for the Non-StructAware models the data points representing the paper's metrics falls somewhere at or below the precision-recall curves. This means that the model, when using one or more of these thresholds, attains or exceeds the paper's metrics. However, for the StructAware models we see that the data points representing the paper's metrics are above the precision-recall curves. This means that those saved models doesn't attain the paper's metrics, no matter which of the tested thresholds are used.

The curved shape exhibited in the precision-recall curves is due to the authors choice of 'samples' averaging [3] for calculating multi-class metrics, whereby precision and recall are computed for each test sample over the 13 different classes, then the mean over all samples is taken. Since the precision and recall of a set with no true positives is 0, if the true logical fallacy is not classified under a given threshold, the precision and recall of that sample are both 0 and the overall mean will drop. Thus high thresholds lead to decreases in both precision and recall. This is an unusual way of

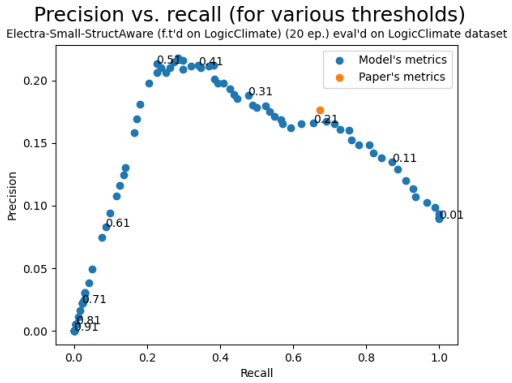

Figure 2: Threshold testing evaluation for StructAware Electra model (with LOGICCLIMATE finetuning) on LOGICCLIMATE, where the model was retrained on `electra-small-mnli` model using best of 20 epochs. The full-size version of this graph and the other precision-recall curves (for best-of-20-epochs retrained `electra-small-mnli` models) can be found here.

computing these metrics and is not directly comparable to more standard 'macro' or 'micro' averaging precision and recall values. The authors don't mention the averaging method or any rationale for it in the paper.

4.3 Retraining with `electra-small-mnli`

Since the authors' saved models didn't achieve all metrics stated in the paper, we decided to retrain pretrained Electra models of our own. We initially tried to fine-tune the originally used `electra-base-mnli` pretrained model, however, that failed due to out-of-memory errors. Instead we tried retraining with the smaller `electra-small-mnli` model. We made the same four model types as the authors' saved models, training them the same way (i.e., using their main scripts `logicedu.py` and `logicclimate.py`). Except, we tried training them in two ways: one with early stopping (stop on the first worse epoch), which is what the authors did, and one that took the best of 20 epochs. (For models finetuned on LOGICCLIMATE, we took the best of 20 epochs both for the training on LOGIC and then for the finetuning on LOGICCLIMATE). We performed the same threshold testing with these models.

For the models we trained with early stopping, the only precision-recall curve that captured the data point corresponding to the paper's metrics is the curve for the Electra model, which was finetuned on LOGICCLIMATE, evaluated on LOGICCLIMATE. These precision-recall plots are omitted from this paper to save space, but they can be found here.

For the models we trained using the best of 20 epochs, we found that the Electra StructAware model fine-tuned on LOGIC and LOGICCLIMATE evaluated on LOGICCLIMATE produced precision-recall curves that came very close to enclosing the paper's metrics. Figure 2 has this precision-recall curve. The non-StructAware version of that model produced a precision-recall curve that captures the paper's non-StructAware metrics. No other precision-recall curves captured the paper's corresponding metrics.

4.4 Retraining with `electra-base-mnli`

After retraining with `electra-small-mnli`, we tried retraining with the larger `electra-base-mnli` model. We only managed to retrain one Electra StructAware model (not finetuned on LOGICCLIMATE). We trained this model similarly to how we retrained on `electra-small-mnli`; see section 3.5 for retraining environment differences. We trained this model with early stopping.

This model's evaluation on LOGIC produced a precision-recall curve very close to capturing the paper's corresponding metrics: see Figure 3. Though, the evaluation on LOGICCLIMATE produced a precision-recall curve didn't capture or get close to capturing the paper's corresponding metrics.

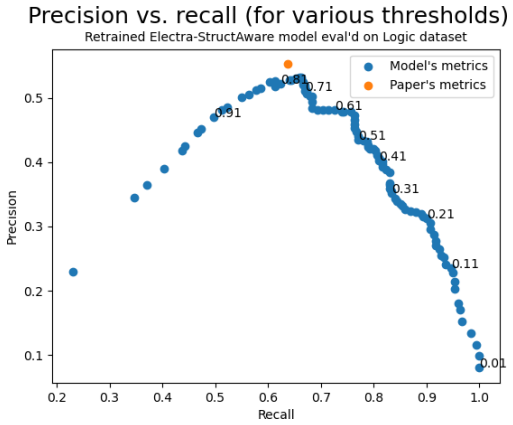

Figure 3: Threshold testing evaluation for StructAware Electra model on LOGIC, where the model was retrained on `electra-base-mnli` using early stopping. The full-size version of this graph and the other precision-recall curves for this model can be found here.

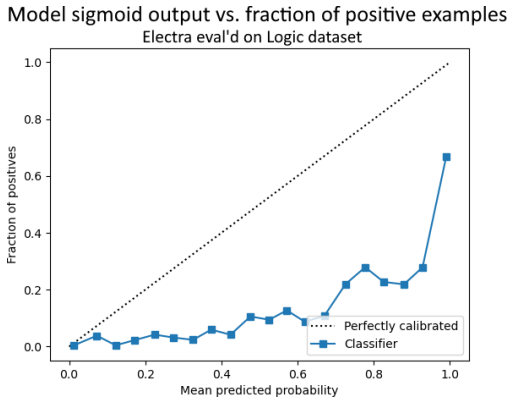

Figure 4: An example of a plot used to determine if a model is calibrated or not. This one tests if the authors' saved non-StructAware Electra model is calibrated on the LOGIC dataset. This and more of these plots can be found here.

5 Additional Experiment: Testing Model Calibration

We additionally tested if the models, both the authors' and ours, are well-calibrated with their evaluations. By "well-calibrated" we mean that, for a binary-classifying task (e.g. the classification of each fallacy class), the model's output between 0 and 1 approximates the probability of that sample being in the positive class. For instance, the set of samples with model outputs of approximately 0.8 would be expected to have about 80% positive true labels for a well-calibrated model. The "Where/how results are stored" section of `reproduction_notes.md` has more details on the plots we make to determine if a model is well-calibrated, as well as how those plots are made.

In Figure 4, we can see that the Electra model is not well-calibrated (at least on the LOGIC dataset) since the fraction of positive examples in each bucket (i.e., examples with the given fallacy) is much lower than the sigmoid output the model produces. Each calibration plot produced, as well as every calibration plot that wasn't looking at individual fallacies, had data point lines significantly below the $x = y$ diagonal. This leads us to conclude that none of the models that we tested, ours or the authors', were well-calibrated.

6 Discussion

Although we have been able to rerun the experiments in the original paper, we were unable to reproduce the entirety of the results, specifically matching the evaluation scores obtained by the StructAware model on LOGICCLIMATE.

6.1 What was Easy

The paper's experiments were straightforward to redo thanks to the authors' public codebase, communication with the authors and other reproducers, and accessible resources. The authors provided guidance on how to invoke the code and other reproducers posted issues on the GitHub page, helping us to avoiding errors. Additionally, our team had access to GPU machines, which allowed us to run the experiments more quickly. Finally, the datasets were provided in a Google Drive linked on the main repository, which made it easy to download and use them.

6.2 What was Difficult

Although a repository was provided for us, the saved model provided for Electra-StructAware did not produce the original metrics when reevaluated. As a result, we made extra efforts to retrain the model from scratch, which was not only difficult due to the lack of documentation but also did not get us closer to arriving at the results reported in the original paper. The lack of comments within the code made it difficult to figure out how the components of the project worked together. For example, we were unable to find any documentation that explained how the different modules of the code interacted with each other. This made it difficult to identify areas of the code that may explain why our results were different despite having the same experimental setup.

In addition to the problems mentioned above, the poor documentation also caused us the following issues:

- It was unclear how much resources we needed to run the experiments, leading to out-of-memory errors.
- It was difficult to determine the run arguments, taking up additional time.
- It was challenging to track down the rationale behind a lot of the code, leading to extra time spent analyzing the codebase.

6.3 Recommendations for Reproducibility

It's possible we missed important steps in the setup process that would have caused our results to diverge. The results of the original paper must be easy to reproduce so the concept of structure awareness in the scope of logical fallacy detection can continue to be expanded upon, so some recommendations for future work include:

- Provide more concrete steps to guide reproducers toward the original results.
- Include READMEs and other forms of documentation throughout the repository to explain how
- Modularize the main scripts more (especially logicedu.py) to allow code changes to be made more easily.
- Verify the uploaded models before releasing them to the public.
- Further explore how different hyperparameters such as classification threshold impact the metrics.

Communication with Original Authors

We have contacted the lead author of the paper, Zhijing Jin, and the main contributor to the Github repository, Abhinav Lalwani. The authors advised us directly on what commands to run to conduct the experiments and provided some implementation details not included in the paper. The authors speculated that failure to reproduce their results when evaluating the saved fine-tuned Electra StructAware models may be due to the wrong models having been saved. However, this does not explain why the model we retrained also struggled to replicate the paper's results.

References

[1] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning. Electra: Pre-training text encoders as discriminators rather than generators, 2020. 1

[2] Z. Jin, A. Lalwani, T. Vaidhya, X. Shen, Y. Ding, Z. Lyu, M. Sachan, R. Mihalcea, and B. Schoelkopf. Logical fallacy detection. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 7180–7198, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics. 1, 2

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 5