ArgueMapper Assistant: Interactive Argument Mining Using Generative Language Models

Mirko Lenz[1,2][0000−0002−7720−0436] and Ralph Bergmann[1,2][0000−0002−5515−7158]

[1] Trier University, Universitätsring 15, 54296 Trier, Germany
info@mirko-lenz.de, bergmann@uni-trier.de
[2] German Research Center for Artificial Intelligence (DFKI)
Branch Trier University, Behringstraße 21, 54296 Trier, Germany
{mirko.lenz,ralph.bergmann}@dfki.de

Abstract. Structured arguments are a valuable resource for analyzing and understanding complex topics. However, manual annotation is time-consuming and often not feasible for large datasets, and automated approaches are less accurate. To address this issue, we propose an interactive argument mining system that takes advantage of generative language models to support humans in the creation of argument graphs. We present the open source ArgueMapper Assistant featuring two prompting strategies and evaluate it on a real-world news dataset. The resulting corpus containing 88 argument graphs is publicly available as well. With generative models, the annotation time is reduced by about 20% while the number of errors is slightly increased (mostly due to missing argumentative units and wrong relation types). A survey provides insights into the usefulness and reliability of the assistant features and shows that participants prefer to use the assistant in the future.

Keywords: Argument Mining · Argument Graphs · Large Language Models · Interactive Systems · Data Annotation.

1 Introduction

Argumentation is available in many forms and plays a crucial role in various domains such as law, politics, and science. A common way to represent it is using natural language texts—for instance, in the form of news articles or scientific papers. Although these texts contain valuable information that can be understood by humans, they are not directly usable by machines. To bridge this gap, Argument Mining (AM) [16] aims to extract structured argumentative elements from natural language text. This may be done manually by trained annotators—leading to high-quality structures—or automatically—which is often less accurate. Consequently, a major challenge in AM is the time-consuming nature of manual annotation, leading to the lack of high-quality datasets for many topics/domains. Research in this field typically focuses on improving automated approaches or interfaces for manual annotation, but to the best of our knowledge, no work has been done to bring together both worlds.

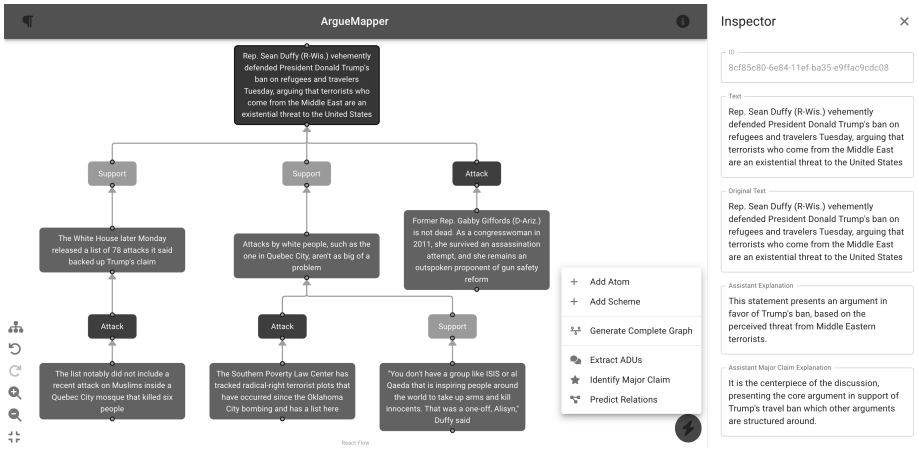


Fig. 1: Screenshot of the interactive mining system with a graph excerpt from the *News Articles* dataset [8]. The assistant button (lightning bolt) contains the actions and the inspector (right sidebar) the explanations.

In this paper, our aim is to bridge this gap through an *interactive* argument mining system that *assists* human annotators in the process of converting plain text arguments to a structured graph-based representation. With recent advances in Natural Language Processing (NLP) [2], AM has seen a shift toward the use of transformer-based language models—either in the form of end-to-end models [15] or as part of a pipeline [18]. Although these supervised approaches reach State of the Art (SOTA) performance, large amounts of labeled data is needed for training—which is scarce for certain domains—and their predictions cannot be adjusted on the fly. In contrast, Large Language Models (LLMs) work in an unsupervised manner and can easily be adjusted to the user's needs through prompting. We leverage this capability to create an interactive system that allows human annotators to hand over parts of the annotation process to the model, reducing the manual effort. Consequently, we seek to answer the following research question: *How to decrease the manual annotation time of argument graphs while maintaining the output quality through LLM-based assistance?* We make the following contributions to answer this question: (i) Two prompting strategies to convert plain text arguments to graphs via LLMs, (ii) a ready-to-use and open–source application enabling interactive argument mining (see Figure 1), (iii) a qualitative evaluation of the system on a real-world news dataset, and (iv) a publicly available corpus containing 88 argument graphs for future research.

2 Foundations

In the following section, we first provide an overview of Computational Argumentation (CA) with a focus on AM, followed by a brief introduction to LLMs.

Computational Argumentation A structured *argument* typically consists of one claim and a set of premises that either *support* or *attack* the claim [22]. Being the smallest units of argumentation, they are also commonly called Argumentative Discourse Units (ADUs) [20]. Claims may also be used as premises in other arguments, enabling the creation of complex argumentative structures. Such chains of arguments often have one central conclusion, the major claim. A common way to represent these structures is to use a directed graph $G = (V, E)$ with the set of nodes V and the set of edges E. The Argument Interchange Format (AIF) [7] defines two types of nodes: (i) *Information* or *atom nodes* being the ADUs, and (ii) *scheme nodes* being the relationships between the ADUs. An example of this format is shown in the screenshot of our annotation tool in Figure 1. Multiple argumentation microstructures have been proposed to represent the relationships between ADUs in a graph, such as serial, linked, and convergent arguments [23]. This distinction has been subject to discussion in the literature—for instance, Goddu [14] argues that there "is no good reason to bother making the distinction" between linked and convergent arguments. Consequently, we focus on linked arguments in our work—that is, a scheme node may only have one premise and one conclusion. AM is concerned with extracting these argumentative elements from natural language text. The term AM comprises a variety of tasks, including (but not limited to) ADU extraction, claim/premise classification, and relation prediction [5]. They may be combined to form a pipeline, allowing the extraction of complete argumentative structures, including argument graphs [19, 18]. We refer the interested reader to the study conducted by Lawrence and Reed [16] for a complete overview of the available approaches.

Large Language Models Arguments are expressed mainly in natural language, which means that there is a strong connection between AM and NLP. Providing a proper introduction to the topic is beyond the scope of this paper, but we briefly introduce the concept of LLMs. They are based on the transformer architecture and its *attention* mechanism, allowing the model to consider the entire input sequence at once [24]. OpenAI's Generative Pre-trained Transformer (GPT) family of LLMs popularized by ChatGPT uses a decoder-only variant that predicts the next token given only the previous sequence as input. Instead of fine-tuning the model for a specific task, *prompting* can be used to guide the model towards the desired output via *few-shot learning* (i.e., giving examples of user input and the desired model output) or even *zero-shot learning* (i.e., providing only user input without output samples) [4].

3 Related Work

To the best of our knowledge, there is no prior work combining manual and automated approaches to AM. In Section 5, we analyze several existing software tools to manually create argument graphs, while in this section we focus on the use of LLMs in text annotation and AM tasks.

Text Annotation with LLMs ChatGPT has been used in various annotation tasks in the field of NLP. For stance and topic detection, it has been found to outperform humans by 25% while being about 30 times cheaper than crowd-sourced labor [13]. In another study, the model has been used to reproduce human labels for sentiment analysis and stance detection tasks, achieving an average accuracy of 0.6 [26]. In addition, ChatGPT has been proposed as an annotation metric for Natural Language Generation (NLG) tasks, showing competitive performance with human ratings [25].

Argument Mining with LLMs AM tasks have also been investigated with LLMs. When applied for common tasks such as claim detection and summarization, ChatGPT achieved average accuracy values of 0.6 for binary classification and 0.5 for multi-class classification with performance varying depending on the number of shots used for prompting [6]. Another study investigated two prompt settings for AM tasks with GPT-4, achieving F metrics of up to 0.7 for ADU detection and 0.5 for relation detection [10]. Compared to a specialized model, GPT-4 was found to be competitive in predicting discourse markers in argumentative texts, even exceeding the specialized model in one metric [21]. Unlike these results, small domain-specific models have been found to outperform GPT-4 for ADU classification in the legal domain—possibly due to its structural complexity [1]. AM has also been treated as an end-to-end text generation task where ADUs and their relations are generated in a single step [9, 15].

4 Mining Argument Graphs using Prompts

Through our literature review in the previous section, we know that LLMs like ChatGPT are capable of performing AM tasks with a certain degree of accuracy, sometimes even outperforming specialized models. Anticipating the use of our strategy in an interactive setting, we developed (i) an *end-to-end* approach and (ii) a *pipeline-based* one. While the former makes the annotation rather straightforward by returning a complete graph, the latter allows the annotator to fix wrong predictions and/or add missing elements. The pipeline has the additional advantage of letting the annotator decide which subtasks to perform—for instance, they may choose to identify ADUs manually and let the model predict the relations afterwards. In both scenarios, the model is asked for its reasoning in the form of a *textual explanation*, allowing the annotator to better understand the model's decision when reviewing the results. To ensure reproducibility, we provide the prompt template along with each task.

Prompting Strategy A critical aspect in designing the prompts is the balance between *zero-shot* and *few-shot* learning. The creation of argument graphs is a rather subjective task where multiple different solutions may be equally correct [12], so few-shot learning may lead to a situation where the model leans towards a different annotation style than the human. To mitigate the issue of unpredictable output when using zero-shot learning, we use *function calling* as

offered by OpenAI's recent ChatGPT models. The core idea is to express a function and its parameters as a JSON-Schema object, pass this information to a model specifically trained, and execute some function locally using the generated arguments. JSON-Schema allows the definition of complex structures: for each parameter, it may include additional information such as a description, expected data types, and constraints. As its name suggests, this feature was originally built to call functions, but we found it to work equally well to extract structured data such as ADUs from a free-form text.

Implementation In addition to the interactive system, we provide a Python implementation[3] of the pipeline-based approach to allow batch processing of large datasets. It contains a server component to easily integrate the LLM-based mining to existing systems and an example client to demonstrate its usage.

4.1 Pipeline-Based Graph Generation

Our approach is loosely based on the pipeline described by Lenz et al. [18], consisting of four main steps: (i) Argument extraction, (ii) relation type classification, (iii) major claim detection, and (iv) graph construction. When developing our interactive system, we found steps (ii) and (iv) to be closely related and therefore merged them into a single step to simplify the process. Step (iii) could also be merged into this step (i.e., by inferring the most important claim from the predicted relations), but we decided to keep it separate to allow for more flexibility and control. The three remaining steps are described below.

ADU Detection Given the original text of the resource to be annotated, the model is prompted to extract all ADUs to be used as atom nodes. We do not differentiate between claims and premises here since a claim may also play a double role as a premise in another argument in graphs. The LLM is instructed to extract only ADUs and not modify the text in the process—otherwise, it will not be possible to locate ADUs in the original text. For the time being, we ignore any kind of *reconstruction*—for instance, replacing pronouns with the correct entities—and leave this to the annotator.

> *Prompt:* The user will provide a long text that contains a set of arguments. Your task is to identify all argumentative discourse units (ADUs) in the text. They will subsequently be used to construct a graph. The user will have the chance to correct the graph, so DO NOT change any text during this step. You shall only EXTRACT the ADUs from the text.

Major Claim Identification Given the ADUs extracted from the original text and their IDs, the model is prompted to identify the major claim among them. In case a non-existent ID is returned by the LLM, the response is discarded, and the annotator will either need to retry or manually select the major claim. In this step, we assume that the major claim is part of the ADUs sent to the model. If it is not part of the original text—and thus not automatically extracted—the annotator may choose to add it manually before executing this step.

[3] https://github.com/recap-utr/arg-services-llm

> *Prompt:* The user will provide a list of argumentative discourse units (ADUs). Your task is to identify the major claim / conclusion of the argument. This node will subsequently be used as the root node of an argument graph. Please provide the ID of the ADU that you consider to be the major claim.

Relation Prediction Given the extracted ADUs and the major claim, the model is prompted to predict relations and their type (i.e., support or attack) between them. Each relation is used as a scheme node in the graph with edges connecting it to the source and target ADUs. As in the previous step, the LLM shall return the IDs of the source and target ADUs. If one of them is not part of the ADU set, the predicted relation is discarded. To simplify the graph construction process, we treat the major claim as the root node of the graph—an assumption that has been made in previous work as well [18].

> *Prompt:* The user will provide a list of argumentative discourse units (ADUs) and the ID of the major claim. Your task is to predict sensible relations in the form of support/attack between them. You shall produce a valid argument graph with the major claim being the root node. You shall create a hierarchical graph with the major claim being the root node (i.e., it should have no outgoing relations, only incoming ones). Flat graphs (i.e., all ADUs directly connected to the major claim directly) are discouraged. There should be no cycles in the graph and no orphaned ADUs.

4.2 End-to-End Graph Generation

The overall goal of this strategy is to perform all three tasks in a single step and thus better utilize the large context window of recent LLMs such as GPT-4 Turbo. Bundling them together should make the generation faster due to the reduced number of requests—we only need one request instead of three. At the same time, the costs should also be lower, since we do not need to feed the output of the model back into the system multiple times. As with the pipeline-based approach, here is the procedure: Given the original text of the resource to be annotated, the model is prompted to perform all three tasks simultaneously and return the complete argument graph.

> *Prompt:* The user will provide a long text that contains a set of arguments. Your task is to generate a complete argument graph containing all ADUs, the major claim, and the relations between the ADUs. ADUs shall only be EXTRACTED from the text, not changed. Relations can either be of type support or attack. You shall create a hierarchical graph with the major claim being the root node (i.e., it should have no outgoing relations, only incoming ones). Flat graphs (i.e., all ADUs directly connected to the major claim directly) are discouraged. There should be no cycles in the graph and no orphaned ADUs.

5 Interactive Argument Mining System

With the prompting strategy for the AM tasks in place, we now discuss their integration into a user-friendly annotation interface. Instead of writing a new tool from scratch, we decided to build on an existing tool to manually build argument graphs from a plain text source. When investigating the available options, we set the following three constraints: (i) The tool should be open-source so that our extensions can be used by other researchers, (ii) its graph representation should be compatible with AIF, and (iii) it should be well-maintained to avoid

building on abandoned software. Among options such as Online Visualization of Arguments (OVA) [3] and MonkeyPuzzle [11], we settled on ArgueMapper [17] as the foundation for our AM system. While OVA has a larger user base and is capable of dealing with dialogical arguments, ArgueMapper was easier to extend due to its modern technology stack—for instance, it uses React for state management and TypeScript for type safety. The Argument Buffers (Arguebuf) [17] format used by ArgueMapper is also compatible with AIF, allowing easy integration with existing tools and datasets.

The following modifications were made to ArgueMapper: (i) The *Plus* button to create new elements is replaced with an *Assistant* button allowing the annotator to invoke the four prompts described in Section 4. Clicking on one of them opens a dialog where the user can add custom instructions to the built-in prompt (e.g., to specify the ADU segmentation level). (ii) A field for nodes containing the textual explanation generated by the model. (iii) Settings to provide an API key, select the model (GPT-3.5 Turbo or GPT-4 Turbo), and specify a custom endpoint. (iv) In case an error occurs during the generation, a bottom bar with details for the user (since the model output is probabilistic, the annotator is encouraged to try it again and/or modify the custom instructions). All changes have been merged into the upstream project[4] under the same MIT license to ensure that the community can benefit from them.

Like the rest of ArgueMapper, our assistant features do not require running a backend component on a server—instead, all requests to the LLM are sent directly from the browser. By setting a custom API endpoint, the assistant features can be used with any LLM that offers an OpenAI compatible API (e.g., using `ollama.com`). As such, our system is compatible not only with proprietary ChatGPT, but also with open models such as Llama.

6 User Study

Having presented both our prompting strategy and its integration in ArgueMapper, in the following we evaluate the feasibility of LLM assisted AM through a user study. Returning to our research question formulated in Section 1, there are two main dimensions that we investigate by formulating two hypotheses: speed (H1) and quality (H2). While the former can be evaluated using quantitative measures, the latter one is more difficult to assess because of the inherent subjectivity—two completely different graphs may be of equal quality. We therefore combine quantitative and qualitative measures to check H2.

H1 (Annotation Time)**:** When given access to the ArgueMapper assistant, annotators are faster than not having access to LLM-based generations.

H2 (Annotation Quality)**:** The availability of an LLM assistant does not cause a decrease in the quality of the resulting argument graphs.

[4] https://github.com/recap-utr/arguemapper

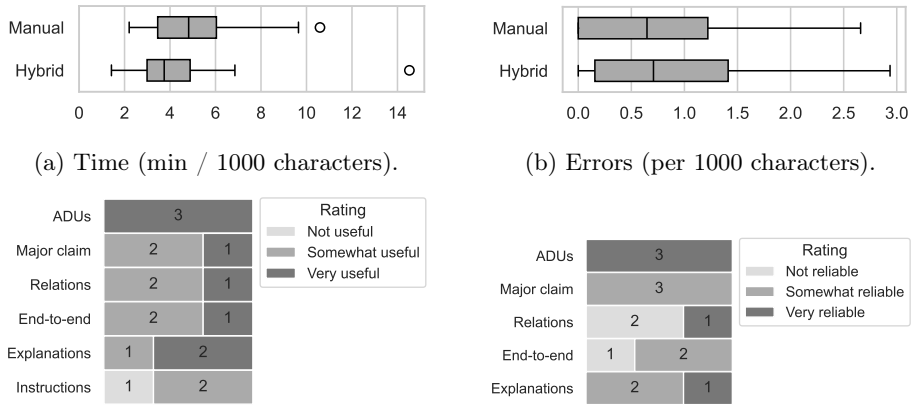
6.1 Experimental Setup

The following section highlights the dataset, the annotation procedure, and the evaluation metrics used in our study. We focus on the pipeline-based approach described in Section 4.1, leaving the end-to-end approach for future work.

Dataset Selection We refrained from re-annotating an existing argument graph corpus, since that could lead the annotators to look up the "reference" solutions and thus biasing the results. Instead, we decided to annotate a new one from scratch. We settled on the *News Articles* dataset compiled by [8] and released under the public domain license CC0 1.0. It consists of 3,824 texts collected between December 2016 and March 2017 from multiple media sources without a specific focus on certain topics. Due to the broad coverage, annotators were not required to have prior knowledge about the dataset. The articles differed greatly w.r.t. the overall length and the dataset exhibited encoding issues with Chinese characters—we therefore removed all articles from "China Daily". To ensure a consistent annotation process covering arguments of comparable length, we also filtered out articles with fewer than 1000 or more than 5000 characters, leaving us with a total of 2335 candidates for annotation.

Annotation Procedure To perform the annotation, we relied on three male computer science students who were familiar with CA and had prior experience in annotating argument graphs using tools such as the aforementioned OVA and ArgueMapper. They had a regular contract with our institution and received a salary above the minimum wage. To get them on the same page about this annotation task, we provided them with guidelines developed by Dumani et al. [12] for a similar type of dataset. Each student annotated a total of 15 articles without the LLMs assistant and 15 with it, selected at random from the preprocessed dataset. Each of them worked with the regular ArgueMapper application for the first 15 articles and only was given access to the assistant for the remaining 15. Two of the students were then asked to identify and fix any errors in the graphs not created by themselves (the articles of the third student were randomly split between the other two), meaning that for each article, two graphs are available in the final corpus. Theoretically, one could compute the Inter-Annotator Agreement (IAA) between the annotators, but the values would be meaningless since the annotators fixed each other's errors instead of creating distinct graphs. Also, computing IAA values for graphs is challenging due to potential differences in the graph structure and potential propagation of errors—for example, if two annotators do not identify the same set of ADUs, the set of relations that can be compared is different as well.

Evaluation Methodology To assess H1, we measure the time from the start of the annotation—that is, loading the article into the tool—to the last edit made to any element of the graph. Furthermore, each annotator recorded the time needed for the annotations in an effort to verify the scores obtained from



(a) Time (min / 1000 characters).

(b) Errors (per 1000 characters).

(c) Usefulness (total votes).

(d) Reliability (total votes).

Fig. 2: Results of the user study, color scheme uses dark shades for favorable and light shades for unfavorable results.

ArgueMapper. To verify H2, we compare the number of errors identified in the second phase of the annotation. We already pointed to the subjective nature of the task, so our initial assumption is that the number of errors does not increase when using the assistant. To further investigate the quality of the annotations, we conducted a survey to gather the annotators' feedback on the assistant features. Among other questions, they were asked to rate *usefulness* and *reliability* of the assistant features on a three-point scale. While the former tries to estimate whether the features were considered helpful in general, the latter assesses the trustworthiness of the results—which, given the tendency of LLMs to confabulate/hallucinate, is a critical aspect. In this context, an example would be the model predicting ADUs that are not present in the original text. They should also assess *perceived impact* of the assistant on speed/quality, the observed response latency, and the preferred ChatGPT model. Lastly, they could provide *free-form feedback* and choose whether they would *prefer using the assistant* in the future or not. After completing the survey individually, we organized a virtual meeting with all annotators to discuss their feedback together.

6.2 Results and Discussion

We now present the results obtained through our user study, starting with the analysis of H1 and then moving on to the evaluation of H2. As part of the process, the annotators identified two articles that did not contain any argumentative information and were therefore not annotated: a list of Grammy winners and a list of Donald Trump's most memorable lines. The corpus containing the remaining 88 argument graphs is publicly available[5] under the attribution license

[5] https://github.com/recap-utr/news-articles-corpus

CC BY 4.0 for future research. On average, the graphs created manually contain 9.61 atom nodes and 8.64 scheme nodes, while the assistant-based ones contain 9.93 and 8.93, respectively. Combined, the published corpus contains a total of 860 atom nodes and 773 scheme nodes. The annotator names have been redacted from the published version and replaced with unique identifiers to ensure their privacy. We also make available versions of the graphs with the errors identified and corrected by the annotators in the second phase of the annotation.

Annotation Time Using the created/edited timestamps of the graphs proved to be unreliable: If an annotator paused the annotation and resumed it later, the break would count towards the total time. We therefore used the manually recorded times for our analysis. Since each annotator worked on a different set of articles, we computed the required time per 1,000 characters to allow for a fair comparison. Analyzing the average duration as shown in Figure 2a, we find that the annotators were indeed faster when using the assistant: The mean duration decreased from five to four minutes—a reduction of $\sim$ 20%. In case of the assistant-based approach, there is one outlier that took more than 14 minutes per 1,000 characters, while the remaining cases took less than 8 minutes per 1,000 characters. The annotator did not provide any additional information about the outlier, but possible reasons include difficulties in understanding the article and/or unwanted predictions by the LLM that needed to be corrected.

Given the ability to test both GPT-3.5 Turbo and GPT-4 Turbo, all three students preferred the latter. Although the more advanced model was found to be slower, the additional time was worth it due to the better results. The two models also differ in terms of costs: GPT-4 Turbo is about 20 times more expensive than GPT-3.5 Turbo. With most requests using GPT-4 Turbo, the total cost for the 45 cases was $\sim$ 4.50 USD (or $\sim$ 0.10 USD per article).

In addition to the raw times, we also asked the annotators about the *perceived impact* of the assistant on the annotation speed: All three reported feeling faster when using the LLM functionality. They also unanimously replied that the observed latency of the responses was fast enough for interactive use. Given that both quantitative and qualitative results support H1, we conclude that the assistant-based approach is indeed faster than the manual one.

Annotation Quality To assess H2, we need to verify that the availability of the LLM features had no negative impact on the quality of the annotated argument graphs. An indicator of quality is the number of errors identified by the annotators in the second phase of the annotation—shown in Figure 2b. We find that the average number of errors per 1,000 characters increased from 0.79 to 0.90 (about 14%). In both cases, we observe less than one error per 1,000 characters, which we consider to be a good result given the complexity of the task. There are no outliers in both boxplots and the maximum number of errors only increased slightly from 2.7 to 2.9. Closer investigation revealed that most errors were related to missing ADUs and wrong relation types.

Additional insights can be gained from the user study shown in Figures 2c and 2d. The *ADU extraction* was rated both as "useful" and as "reliable" by the three annotators. The *major claim identification* and *relation prediction* were rated at least "somewhat useful", but differed in terms of reliability. The former was found to be "somewhat reliable" by all participants, but the latter was rated as "not reliable" by two of them. Since these features are successive steps in a pipeline, we conclude that the more complex the task, the less reliable the results. This interpretation is consistent with the feedback received for the *end-to-end graph generation*: They found it to be at least "somewhat useful", but one student rated it as "not reliable" and the others as "somewhat reliable". The explanations generated by the LLM were found to be "useful" by most annotators, but were considered "not/somewhat reliable". *Custom instructions* were the only feature not found to be "useful" by at least one participant, there was even a vote for "not useful". We did not measure its reliability of the feature as it changes the behavior of the LLM and does not generate any output on its own.

Overall, the assistant features are mostly considered to be useful. Reliability ratings are more mixed, with the relation prediction and end-to-end approach being the most problematic. When asked about the assistant's *perceived impact* on the quality of the argument graphs, two annotators reported no change, while one even found that the assistant had a positive impact. Although there were some issues with the assistant, they found it to be a valuable tool for the annotation of argument graphs, especially when dealing with long texts, and would prefer to use it again in the future. Although the results of the user study suggest that the assistant features do not negatively impact the quality of the annotated argument graphs, the error analysis shows that the number of errors increased slightly. We therefore reject H2, but note that the increase in errors may be mitigated in the future with improvements in the LLM technology, optimized prompts that better guide the model towards the desired result, and special training of the annotators to better understand the model's behavior.

Free-Form Feedback Besides answering multiple questions, the annotators also provided free-form feedback. We summarize the main points in the following, starting with general aspects. The participants reported multiple instances where the LLM response could not be parsed properly by the application, meaning that they had to try again and wait for a new response. Such errors may occur if invalid JSON objects are generated by the LLM, meaning that OpenAI does not strictly enforce the provided schema in all cases. The probabilistic nature of the assistant was received with mixed feelings: It allowed the annotators to obtain different results by running the same generation twice (as anticipated), but these inconsistent results also led to some confusion as to which one to keep. Finally, the annotators wanted to have a best-practice guide on how to use the assistant features effectively.

The discussion of individual features mainly confirmed the findings obtained from Figures 2c and 2d. The *ADU extraction* was praised by all participants and even helped one of them better understand the structure of the text. They

also reported that this step should rather extract too much than too little, as it is easier to remove unwanted ADUs than adding missing ones. A problem of this step was that the model sometimes rephrased ADUs—requiring the human to fix the text manually. The *major claim identification* was found to be useful if the text actually contained a statement that could be considered the main conclusion. As an improvement, they suggested adding a feature to automatically synthesize a major claim from the original text—for instance, through summarization. The *relation prediction* was found to be more effective in shorter texts and less so in longer ones. The annotators reported multiple instances of isolated subgraphs and/or circular relations. The generation of graphs using the *end-to-end* approach was not extensively used, and instead the annotators preferred to fix issues between the individual steps manually. The *explanations* were found to be consistent with the generated predictions; their usefulness, however, was somewhat limited: ADU that were deemed non-argumentative by the annotator often provided contained a summary of its content instead of proper reasoning—indicating that the model should not have extracted them in the first place. *Custom instructions* were seldom used: In cases where the model output was not satisfactory, it was faster for them to do it manually instead of trying to fix the assistant's behavior. Instead, participants wished to modify the temperature of the model so that they could decide between more creative and more conservative responses depending on the current article.

Limitations Using LLMs for the graph construction task may introduce biases towards certain structures which the annotators may accept without question. The assistant tends to generate more hierarchical graphs than annotators would manually create in certain cases. In addition, predicting relations and/or complete graphs may lead to semantically invalid results, which the annotator has to fix manually (e.g., circular relations). LLMs are probabilistic models with results that vary greatly in different runs even when using the same input—potentially leading to confusion for the annotator. At the same time, this variation allows us to incorporate the notion of subjectivity into our interactive use case: The annotator may choose to run the same generation twice to obtain different results and decide which one to keep based on their own judgement.

Our qualitative results are based on feedback from three annotators and one dataset, which may not be representative of real-world scenarios. The students also likely knew what we were trying to show, which may have influenced their votes on the usefulness and reliability of the assistant features. To determine the speed improvement, we analyzed the annotation time for the entire graph creation process without breaking it down into individual tasks. Consequently, it may be the case that the speedup is caused by one of the tasks, whereas others may actually be slower.

7 Conclusion and Future Work

In this paper, we have presented an interactive system for the annotation of argument graphs that integrates LLMs to support the annotators. Our prompting strategy covers both an end-to-end technique and a pipeline-based one, allowing annotators to choose the most suitable approach for the task at hand. Our user study demonstrated that the modified version of ArgueMapper in fact decreases the annotation time of the argument graphs, while only having a slight impact on the resulting quality. In addition to making our ArgueMapper assistant publicly available, we also release the argument graph corpus created as part of our study. The speed improvements of the interactive system may contribute to the availability of more argument graph corpora, which in turn could positively affect the retrieval of arguments.

In future work, we plan to extend the functionality offered by the assistant—for instance, by adding a way to synthesize a major claim from the original text. Furthermore, we see the potential of using the LLM features to onboard new annotators: In addition to written guidelines with a static selection of examples, they could utilize LLM predictions to receive dynamic feedback for their task.

Acknowledgments. This work has been funded by *German Academic Scholarship Foundation (Studienstiftung des deutschen Volkes)*.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Al Zubaer, A., Granitzer, M., Mitrović, J.: Performance analysis of large language models in the domain of legal argument mining. Frontiers in Artificial Intelligence (2023)
2. Allen, J.F.: Natural language processing. In: Encyclopedia of Computer Science, pp. 1218–1222. John Wiley and Sons Ltd. (2003)
3. Bex, F., Lawrence, J., Snaith, M., Reed, C.: Implementing the argument web. Communications of the ACM pp. 66–73 (2013)
4. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language Models are Few-Shot Learners. In: Advances in Neural Information Processing Systems. pp. 1877–1901. Curran Associates, Inc. (2020)
5. Cabrio, E., Villata, S.: Five years of argument mining: A data-driven analysis. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence. pp. 5427–5433. AAAI Press (2018)
6. Chen, G., Cheng, L., Tuan, L.A., Bing, L.: Exploring the Potential of Large Language Models in Computational Argumentation (2023)
7. Chesñevar, C.I., McGinnis, J., Modgil, S., Rahwan, I., Reed, C., Simari, G.R., South, M., Vreeswijk, G., Willmott, S.: Towards an argument interchange format. The Knowledge Enigneering Review p. 293 (2006)
8. Dai, T.: News Articles (2017). https://doi.org/10.7910/DVN/GMFCTR

9. Das, N., Choudhary, V., Saradhi, V.V., Anand, A.: End-to-End Argument Mining as Augmented Natural Language Generation (2024)
10. de Wynter, A., Yuan, T.: "I'd Like to Have an Argument, Please": Argumentative Reasoning in Large Language Models (2024)
11. Douglas, J., Wells, S.: Monkeypuzzle - Towards Next Generation, Free & Open-Source, Argument Analysis Tools. In: CMNA@ICAIL (2017)
12. Dumani, L., Biertz, M., Witry, A., Ludwig, A.K., Lenz, M., Ollinger, S., Bergmann, R., Schenkel, R.: The ReCAP Corpus: A Corpus of Complex Argument Graphs on German Education Politics. In: IEEE Proceedings of the 15th International Conference on Semantic Computing (ICSC). pp. 248–255 (2021)
13. Gilardi, F., Alizadeh, M., Kubli, M.: ChatGPT outperforms crowd workers for text-annotation tasks. Proceedings of the National Academy of Sciences p. e2305016120 (2023)
14. Goddu, G.C.: Against Making the Linked-Convergent Distinction. In: Pondering on Problems of Argumentation: Twenty Essays on Theoretical Issues, pp. 181–189. Springer Netherlands (2009)
15. Kawarada, M., Hirao, T., Uchida, W., Nagata, M.: Argument Mining as a Text-to-Text Generation Task. In: Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 2002–2014. Association for Computational Linguistics (2024)
16. Lawrence, J., Reed, C.: Argument Mining: A Survey. Computational Linguistics pp. 765–818 (2019)
17. Lenz, M., Bergmann, R.: User-Centric Argument Mining with ArgueMapper and Arguebuf. In: Computational Models of Argument. pp. 367–368. IOS Press (2022)
18. Lenz, M., Sahitaj, P., Kallenberg, S., Coors, C., Dumani, L., Schenkel, R., Bergmann, R.: Towards an Argument Mining Pipeline Transforming Texts to Argument Graphs. In: Proceedings of the 8th International Conference on Computational Models of Argument. pp. 263–270. IOS Press (2020)
19. Nguyen, H.V., Litman, D.J.: Argument Mining for Improving the Automated Scoring of Persuasive Essays. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
20. Peldszus, A., Stede, M.: From Argument Diagrams to Argumentation Mining in Texts - A Survey. IJCINI pp. 1–31 (2013)
21. Rocha, G., Lopes Cardoso, H., Belouadi, J., Eger, S.: Cross-genre argument mining: Can language models automatically fill in missing discourse markers? Argument & Computation pp. 1–41 (2024)
22. Stab, C., Gurevych, I.: Identifying Argumentative Discourse Structures in Persuasive Essays. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 46–56. Association for Computational Linguistics (2014)
23. Stab, C., Gurevych, I.: Parsing Argumentation Structures in Persuasive Essays. Computational Linguistics pp. 619–659 (2017)
24. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., ukasz Kaiser, Ł., Polosukhin, I.: Attention is All you Need. In: Advances in Neural Information Processing Systems. Curran Associates, Inc. (2017)
25. Wang, J., Liang, Y., Meng, F., Sun, Z., Shi, H., Li, Z., Xu, J., Qu, J., Zhou, J.: Is ChatGPT a Good NLG Evaluator? A Preliminary Study. In: Proceedings of the 4th New Frontiers in Summarization Workshop. pp. 1–11. Association for Computational Linguistics (2023)
26. Zhu, Y., Zhang, P., Haq, E.U., Hui, P., Tyson, G.: Can ChatGPT Reproduce Human-Generated Labels? A Study of Social Computing Tasks (2023)