Practical Solutions to Practical Problems in Developing Argument Mining Systems

Debela Gemechu[1], Ramon Ruiz-Dolz[1], John Lawrence[1], Chris Reed[1]

[1]Centre for Argument Technology, University of Dundee

Abstract

The Open Argument Mining Framework (oAMF) addresses key challenges in argument mining research which still persist despite the field's impressive growth. Researchers often face difficulties with cross-system comparisons, incompatible representation languages, and limited access to reusable tools. The oAMF introduces a standardised yet flexible architecture that enables seamless component benchmarking, rapid pipeline prototyping using elements from diverse research traditions, and unified evaluation methodologies that preserve theoretical compatibility. By reducing technical overhead, the framework allows researchers to focus on advancing core argument mining capabilities rather than reimplementing infrastructure, fostering greater collaboration at a time when computational reasoning is increasingly vital in the era of large language models.

1 Introduction

Argument Mining (AM) represents one of the most intellectually stimulating frontiers in computational linguistics today. However, for researchers and practitioners in the AM community, several pain points have become increasingly apparent. First, the inherently modular nature of argument mining – comprising multiple interdependent subtasks – creates substantial difficulties when attempting to compare systems or integrate components from different research efforts. Second, even when technical integration is possible, conceptual interoperability is hindered by divergent representation languages that encode different theoretical assumptions about argument structure. Finally, despite the wealth of research publications, there is a notable scarcity of accessible, reusable tools, with many innovations remaining as isolated research prototypes (Chen et al., 2024; Habernal et al., 2024; Kawarada et al., 2024; Cabessa et al., 2025; Gorur et al., 2025a).

These challenges directly impact the daily work of argument mining researchers: How does one fairly compare a newly developed component against existing approaches? How can a task-specific module be efficiently integrated into an end-to-end system? What is the most effective way to evaluate and visualise results across different argument representations?

The Open Argument Mining Framework (oAMF) addresses these fundamental concerns by providing a standardised yet flexible architecture that facilitates module comparison, system integration, and consistent evaluation methodologies. Unlike previous approaches that have prioritised specialised solutions for narrow use cases, the oAMF creates an ecosystem where researchers can:

- Seamlessly benchmark new components against established baselines using standardised interfaces
- Rapidly prototype end-to-end argument mining pipelines by mixing components from different research traditions
- Visualise and evaluate results through unified representation formats that maintain theoretical compatibility

By reducing the technical overhead associated with these common development scenarios, the oAMF aims to accelerate innovation while preserving the theoretical diversity that has been a hallmark of argument mining research. The framework empowers researchers to focus on advancing core argument mining challenges rather than reimplementing infrastructure components, ultimately fostering greater collaboration across the community.

Currently, the framework includes 17 widely used AM modules, all available on GitHub for community contributions. New modules can also be added, with each module expected to follow specific input/output formats, implementation guidelines, and configuration requirements (see Section 4).

2 Practical Problems faced by AM system developers

The deployment of applied argument mining systems is still in its infancy. Despite more than a decade of research in argument mining (ultimately stretching back to (Moens et al., 2007), but now eleven years with the dedicated forum provided by the Argument Mining Workshop) there have been few live systems deployed beyond laboratory settings. The most high-profile is the work at IBM (Slonim et al., 2021) which has to a large extent been rolled in to watsonx as part of their commercial offering. In addition, there are more modest examples such as args.me (Wachsmuth et al., 2017) and the Evidence Toolkit (Visser et al., 2020) amongst others. But the engineering and deployment of systems that involve AM remains a rarity.

With the revolution in NLP ushered in by LLMs, the ability to handle reasoning in language is becoming paramount, and as a result the handling of structures of argumentation is of rapidly increasing importance, as evidenced by the dramatic uptick in papers on the topic in the ACL anthology which returns 7,500 papers for the search "`argument* mining`" at time of writing.

Yet there are several fundamental methodological challenges that face both researchers and developers setting out to build argument mining algorithms and systems.

The all-or-nothing challenge. Reliably extracting the structure of reasoning expressed in natural language remains one of the most challenging open problems in NLP today. Many different architectures and approaches have been applied, and though monolithic end-to-end systems are rather rare (Eger et al., 2017), the more modular approaches very typically have to either engineer end-to-end system componentry, or else release systems that make major I/O assumptions (such as the availability of reliably segmented input data, or the availability of directionality labelling subsystems). Building an entire application system that exploits argument mining is therefore an all-or-nothing affair, requiring system building from user input to user output.

The reusability challenge. Part of the reason for the all-or-nothing challenge is that tools and algorithms released by the community are typically rather shortlived and idiosyncratic, making their reuse difficult in the short term, and all but impossible over the course of a few years. As a result, progress is rarely able to make use of previous work, reusing, for example, techniques for segmentation, where that is not the focus of current work.

The interoperability challenge. The other part of the reason for the all-or-nothing challenge is the lack of well-defined modularity or the ability for subsystems to exchange data in a common representation language. Freeman (Freeman, 1991) has become one of the most expressive underlying reference argumentation theories because of its ability to integrate approaches such as Toulmin (Toulmin, 1958) with simpler pro-con models (Gorur et al., 2025b), and as a result is used in data efforts ranging from monological lab-constructed data of the microtext corpus (Peldszus and Stede, 2016) through to some of the largest manually annotated dialogical corpora (Hautli-Janisz et al., 2022) currently available. Several shared tasks have focused on exploiting this representational adequacy (Ruiz-Dolz et al., 2024), but the sheer creative diversity that has characterised argument mining for more than a decade has also created a rich array of different approaches that effectively stymie interoperability between them.

The evaluation challenge. In both academic and commercial environments, providing unbiased evaluation of techniques and systems is critical, yet reliable measures of different aspects of argument mining performance are difficult to establish beyond the bounds of controlled shared tasks, because of the lack of interoperability and standardisation. Even the very measures that are deployed vary widely from, for example, κ, which fails to account for textual variation, through γ, which is difficult to interpret.

3 Solving challenges in the development of AM work

3.1 Compare against other modules

The oAMF allows for easy comparison of approaches on individual argument mining tasks by creating two or more workflows that are identical other than the specific task under consideration. A variety of modules can then be tested on this task,

for example comparing a newly developed module against existing state-of-the-art approaches. Importantly, the framework is not bound to any single argumentation model or theory. It is designed to support heterogeneous models and allows components grounded in different frameworks, theories, or representational schemes to interoperate seamlessly. This flexibility ensures that researchers can easily adapt the framework to suit their preferred models or experiment with multiple ones in parallel.

3.2 Fit a single part in to an end-to-end workflow

Although there are an increasing number of AM works which take an 'end-to-end' view (Stab and Gurevych, 2017; Persing and Ng, 2016; Potash et al., 2017), it is still common to focus on specific individual tasks from the identification of argument components, through levels of increasing complexity; considering the role of individual components, considering argumentative relations, and considering more complex argumentative relationships, such as an instance of an argumentation scheme.

The oAMF allows for different implementations to be selected for each of these tasks, creating a unified end-to-end approach that uses the best techniques available along each step of the process.

3.3 Evaluation and Visualisation

Individual oAMF modules and AM pipelines composed of oAMF modules—each responsible for a specific subtask—can be executed, with the output visualised using an oAMF-compatible visualisation tool (see an example of the argument graph visualisation in Figure 1). Additionally, the output can be assessed for performance using another oAMF-compatible module, CASS. The CASS module evaluates oAMF output based on metrics such as Macro F1, Accuracy, Text Similarity, Kappa, and U-Alpha. For example, a pipeline might start with a module for dialog turn segmentation, followed by a module for segmenting text into argumentative discourse units, a module for pre-processing the discourse segments, and finally, a module for argument relation identification. For a list of the modules currently available in oAMF, see Section 4.3.

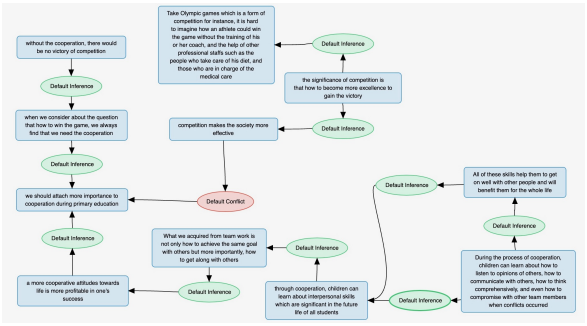

Figure 1: An argument map generated by the visualiser.

```python
from xaif import AIF
# Sample xAIF JSON with 2 L nodes and 2 I nodes
aif_data = {"AIF": {"nodes": [
     {"nodeID": 0, "text": "Example L node 1", "type": "L"},
     {"nodeID": 1, "text": "Example L node 2", "type": "L"},
     {"nodeID": 2, "text": "Example I node 1", "type": "I"},
     {"nodeID": 3, "text": "Example I node 2", "type": "I"},
     {"nodeID": 4, "text": "Default Inference", "type": "RA"}
   ],
   "edges": [
     {"edgeID": 0, "fromID": 0, "toID": 2},
     {"edgeID": 1, "fromID": 1, "toID": 3},
     {"edgeID": 2, "fromID": 2, "toID": 4},
     {"edgeID": 4, "fromID": 2, "toID": 3}
   ],
   "locutions": [{"nodeID": 0, "personID": 0}],
   "participants": [{"firstname": "Speaker", "participantID": 0,
         "surname": "Name"}]
 },
  "dialog": True
  }

aif = AIF(aif_data) # Initialise the AIF object with xAIF data
aif.add_component(component_type = "locution", text = "Example L node
      3.", speaker = "Another Speake") # The next ID (5) is assigned
aif.add_component(component_type = "proposition", Lnode_ID = 5,
      proposition = "Example I node 3.") # The L-NodeID is required
aif.add_component(component_type = "argument_relation", relation_type =
      "RA", iNode_ID2=3, iNode_ID1=6) # Requires I-Node IDs and AR type
print(aif.xaif) # Print the generated xAIF data
print(aif.get_csv("argument-relation")) # Exports to tabular format
```

Figure 2: `xaif` package to manipulate xAIF data.

4 Practical Solutions

4.1 How to create an oAMF module

The oAMF allows developers to create new argument mining modules and integrate them with others, simplifying interoperability and reproducibility of AM systems. This section describes the process of creating a new module, covering I/O format constraints, its implementation, the project structure, how to configure the metadata file, the Flask application routes, and a summary of steps for developing oAMF-compatible AM modules.

Input-Output Format: Each module uses xAIF for input and output to ensure interoperability. The *xaif* library provided by the oAMF simplifies xAIF file manipulation (see Figure 2), helping developers in managing argumentative discourse structure in a consistent format. The documentation can be accessed at https://github.com/arg-tech/xaif/blob/main/docs/tutorial.md.

Implementation: oAMF modules are implemented as a dockerised web service to ensure portability and scalability. They are implemented using the Flask framework, a lightweight Python web framework for creating RESTful services. A set of endpoints are exposed, allowing users to interact with the module through HTTP requests. Each module takes xAIF as input/output. Developers can build new modules by cloning a template project, updating metadata, implementing module logic, and configuring the service for containerisation. A template to help develop custom modules is available at: https://github.com/arg-tech/AMF_NOOP/. A step-by-step summary of module development is provided in Appendix A.

4.2 How to create an oAMF pipeline

The oAMF offers different interfaces for building and executing AM pipelines i.e., multiple modules working sequentially. These components seamlessly integrate by using xAIF as a standardised format for both input and output, ensuring smooth data exchange throughout the pipeline. The available interfaces cover all different levels of technical knowledge, including an API for advanced customisation, a drag-and-drop interface for quick setup, and a web interface for easy execution.

4.2.1 Programming API

The programming API allows defining a pipeline by specifying and connecting modules through their associated tags. The pipeline can then be executed by providing an input file. The script shown in Figure 3 depicts how to build and execute an oAMF pipeline.

```
from oamf import oAMF
# Initialize the library
oamf = oAMF()
# Define pipeline as a graph


pipeline_graph = [
       ("turninator", "segmenter"), # 'turninator' is a module that
              segments dialogue into turns; 'segmenter' segments
              discourse into ADUs
       ("turninator", "segmenter2"), # another segmenter instance for
              parallel processing
       ("segmenter", "bert-te"),   # 'bert-te' is a BERT-based
              inference identifier module
       ("segmenter2", "bert-te2")  # another inference module instance
       ]
oamf.pipelineExecutor(pipeline_graph, "input_file.json")
```

Figure 3: Create and execute pipeline with the oAMF API.

4.2.2 Drag-and-Drop Interface

The oAMF integrates with n8n, an open-source workflow automation tool[1], available at https://n8n.oamf.arg.tech/, offering a visual, intuitive interface for constructing pipelines. Users can easily drag and drop modules and establish connections. Pipelines can be executed using (1) the n8n interface with user-provided input or (2) the oAMF library by downloading workflow JSON files and running `oamf.pipelineExecutor(pipeline_graph, "input_file.json", "workflow.json")`, where `pipeline_graph` can be an empty list, `input_file.json` holds xAIF input data, and `workflow.json` is the n8n workflow.

4.2.3 Web Interface

The oAMF provides a web interface for quickly running AM pipelines, which can be accessed at https://oAMF.arg.tech. Users can upload input data (e.g., text or xAIF files), select pre-built pipelines using the n8n interface, and execute them directly on the oAMF server—removing the need for manual pipeline construction.

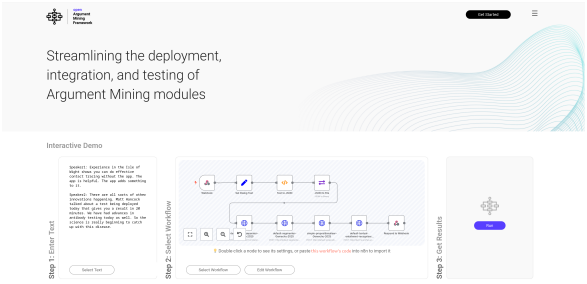

Figure 4: Web interface of oAMF for uploading input data and running pre-built AM pipelines.

4.3 List extant functionality of the oAMF

In addition to providing a complete toolkit to implement new argument mining modules, and different interfaces to connect them and execute pipelines, the oAMF also comes with a set of pre-implemented modules covering basic AM operations such as segmentation, classification, or relation identification. Some of these pre-implemented modules are based on previous AM work. For example, a segmenter based on TARGER (Chernodub et al., 2019), a cascade propositionaliser (Jo et al., 2019), a transformer-based argument relation identifier (Ruiz-Dolz et al., 2021), a decoder-only model

[1] https://n8n.io

based on DialoGPT for argument relation identification (Gemechu et al., 2024) or decompositional argument mining (Gemechu and Reed, 2019). By providing this set of pre-implemented modules covering basic AM operations, we make argument mining accessible to a non-technical audience. A list of available modules can be accessed here: https://github.com/arg-tech/oAMF.

5 Conclusion

This paper addresses a significant challenge impacting the AM community: the use, development, and evaluation of practical solutions. For that purpose, we provide a complete analysis of the practical problems faced by the AM community, grouped into four major categories. As a solution to these problems, we introduce the oAMF: a set of libraries, modules, and interfaces aimed at making AM accessible to users with different technical backgrounds. The oAMF allows, at the same time, developers to create new interoperable modules from scratch that can be connected with the existing ones, programmers to implement different AM pipelines connecting different modules and evaluate them fairly, and non-technical users to analyse text in search for argument structures with a codeless interface.

Acknowledgments

This work is funded in part by: the 'AI for Citizen Intelligence Coaching against Disinformation (TITAN)' project, funded by the EU Horizon 2020 research and innovation programme under grant agreement 101070658, and by UK Research and innovation under the UK governments Horizon funding guarantee grant numbers 10040483 and 10055990; the 'Artificial Intelligence for Institutionalised, Multimodal, Gamified, Mass Democratic Deliberations' project, funded by the EU Horizon Europe Framework Programme (HORIZON) under grant agreement 101178806; the 'CLARUS' project, funded by the EU Horizon Europe Framework Programme (HORIZON) under grant agreement 101121182; Volkswagen Stiftung Foundation under grant 98 543, "Deliberation Laboratory"; and the Swiss National Science Foundation under grant 10001FM_200857, "Mining argumentative patterns in context".

References

Jérémie Cabessa, Hugo Hernault, and Umer Mushtaq. 2025. Argument mining with fine-tuned large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 6624–6635.

Guizhen Chen, Liying Cheng, Luu Anh Tuan, and Lidong Bing. 2024. Exploring the potential of large language models in computational argumentation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2309–2330.

Artem Chernodub, Oleksiy Oliynyk, Philipp Heidenreich, Alexander Bondarenko, Matthias Hagen, Chris Biemann, and Alexander Panchenko. 2019. Targer: Neural argument mining at your fingertips. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 195–200.

Steffen Eger, Johannes Daxenberger, and Iryna Gurevych. 2017. Neural end-to-end learning for computational argumentation mining. *arXiv preprint arXiv:1704.06104*.

James B. Freeman. 1991. *Dialectics and the Macrostructure of Arguments*. Foris/de Gruyter, Berlin-New York.

Debela Gemechu and Chris Reed. 2019. Decompositional argument mining: A general purpose approach for argument graph construction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 516–526.

Debela Gemechu, Ramon Ruiz-Dolz, and Chris Reed. 2024. Aries: A general benchmark for argument relation identification. In *11th Workshop on Argument Mining, ArgMining 2024*, pages 1–14. Association for Computational Linguistics (ACL).

Deniz Gorur, Antonio Rago, and Francesca Toni. 2025a. Can large language models perform relation-based argument mining? In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 8518–8534.

Deniz Gorur, Antonio Rago, and Francesca Toni. 2025b. Can large language models perform relation-based argument mining? In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 8518–8534, Abu Dhabi, UAE. Association for Computational Linguistics.

Ivan Habernal, Daniel Faber, Nicola Recchia, Sebastian Bretthauer, Iryna Gurevych, Indra Spiecker genannt Döhmann, and Christoph Burchard. 2024. Mining legal arguments in court decisions. *Artificial Intelligence and Law*, 32(3):1–38.

Annette Hautli-Janisz, Zlata Kikteva, Wassiliki Siskou, Kamila Gorska, Ray Becker, and Chris Reed. 2022.

Qt30: A corpus of argument and conflict in broadcast debate. In *Proceedings of the 13th Language Resources and Evaluation Conference*, pages 3291–3300. European Language Resources Association (ELRA).

Yohan Jo, Jacky Visser, Chris Reed, and Eduard Hovy. 2019. A cascade model for proposition extraction in argumentation. In *Proceedings of the 6th Workshop on Argument Mining*, pages 11–24, Florence, Italy. Association for Computational Linguistics.

Masayuki Kawarada, Tsutomu Hirao, Wataru Uchida, and Masaaki Nagata. 2024. 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)*, pages 2002–2014.

Marie-Francine Moens, Erik Boiy, Raquel M. Palau, and Chris Reed. 2007. Automatic detection of arguments in legal texts. In *Proceedings of the 11th international conference on Artificial intelligence and law*, pages 225–230, Stanford, CA. ACM.

Andreas Peldszus and Manfred Stede. 2016. An annotated corpus of argumentative microtexts. In *Argumentation and Reasoned Action: Proceedings of the 1st European Conference on Argumentation, Lisbon 2015 / Vol. 2*, pages 801–815, London. College Publications.

Isaac Persing and Vincent Ng. 2016. End-to-end argumentation mining in student essays. In *Proceedings of NAACL-HLT*, pages 1384–1394, San Diego, CA.

Peter Potash, Alexey Romanov, and Anna Rumshisky. 2017. Here's my point: Joint pointer architecture for argument mining. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1364–1373, Copenhagen, Denmark. Association for Computational Linguistics.

Ramon Ruiz-Dolz, Jose Alemany, Stella M Heras Barberá, and Ana García-Fornes. 2021. Transformer-based models for automatic identification of argument relations: A cross-domain evaluation. *IEEE Intelligent Systems*, 36(6):62–70.

Ramon Ruiz-Dolz, John Lawrence, Ella Schad, and Chris Reed. 2024. Overview of dialam-2024: Argument mining in natural language dialogues. In *11th Workshop on Argument Mining, ArgMining 2024*, pages 83–92. Association for Computational Linguistics (ACL).

N. Slonim, Y. Bilu, and C. Alzate. 2021. An autonomous debating system. *Nature*, 591:397–384.

Christian Stab and Iryna Gurevych. 2017. Parsing argumentation structures in persuasive essays. *Computational Linguistics*, 43(3):619–659.

Stephen E. Toulmin. 1958. *The Uses of Argument*. Cambridge University Press.

Jacky Visser, John Lawrence, and Chris Reed. 2020. Reason-checking fake news. *Communications of the ACM*, 63(11):38–40.

Henning Wachsmuth, Martin Potthast, Khalid Al-Khatib, Yamen Ajjour, Jana Puschmann, Jiani Qu, Jonas Dorsch, Viorel Morari, Janek Bevendorff, and Benno Stein. 2017. Building an argument search engine for the web. In *Proceedings of the 4th Workshop on Argument Mining*, pages 49–59, Copenhagen, Denmark. Association for Computational Linguistics.

A Implementation Details

Project Structure The project structure contains the following key components:

- `config/metadata.yaml`: Contains metadata information about the module.
- `project_src_dir/`: Directory with the application code, including Flask routes and logic.
- `boot.sh`: Shell script to activate the environment and launch the app.
- `docker-compose.yaml`: Defines the Docker service and its setup.
- `Dockerfile`: Specifies image configuration and dependencies.
- `requirements.txt`: Python dependencies list.

Metadata Configuration The `metadata.yaml` file provides essential module details:

```
Name: "Name of the Module"
Date: "2024-10-01"
Originator: "Author"
License: "Your License"
AMF_Tag: Your_tag_name
Domain: "Dialog"
Training Data: "Annotated corpus X"
Citation: ""
Variants:
  - name: 0 version: null
  - name: 1 version: null
Requires: text
Outputs: segments
```

Flask Application Routes

- **Index Route (`/`)**: Displays the contents of the `README.md` file.
- **Module Route (customisable)**:
  - `POST` requests process xAIF input and return modified output.
  - `GET` requests return module metadata and documentation.

Steps to Develop a Module

1. Clone the NOOP template from the repository: https://github.com/arg-tech/AMF_NOOP/
2. Modify `metadata.yaml` with your module's details.
3. Implement core logic in `routes.py`.
4. Use the `xAIF` library to manipulate xAIF data.
5. Set up `Dockerfile` and `docker-compose.yaml`.
6. Update the `README.md` with documentation.