The Open Argument Mining Framework

Debela Gemechu[1], Ramon Ruiz-Dolz[1], Kamila Gorska[1], Somaye Moslemnejad[1], Eimear Maguire[1], Dimitra Zografistou[1], Yohan Jo[2], John Lawrence[1], Chris Reed[1]

[1]Centre for Argument Technology, University of Dundee
[2]Graduate School of Data Science, Seoul National University

Correspondence: debela@arg.tech
Demo Video: https://youtu.be/Gtw0ly9QBZw

Abstract

Despite extensive research in Argument Mining (AM), the field faces significant challenges in limited reproducibility, difficulty in comparing systems due to varying task combinations, and a lack of interoperability caused by the heterogeneous nature of argumentation theory. These challenges are further exacerbated by the absence of dedicated tools, with most advancements remaining isolated research outputs rather than reusable systems. The oAMF (Open Argument Mining Framework) addresses these issues by providing an open-source, modular, and scalable platform that unifies diverse AM methods. Initially released with seventeen integrated modules, the oAMF serves as a starting point for researchers and developers to build, experiment with, and deploy AM pipelines while ensuring interoperability and allowing multiple theories of argumentation to co-exist within the same framework. Its flexible design supports integration via Python APIs, drag-and-drop tools, and web interfaces, streamlining AM development for research and industry setup, facilitating method comparison, and reproducibility.

1 Introduction

The automatic recognition of the structure of human reasoning in natural language discourse – argument mining (AM) – is a particularly challenging task in NLP. Various reviews have surveyed the field (Lippi and Torroni, 2016; Stede and Schneider, 2019; Lawrence and Reed, 2019), though more recently surveys have become much harder to assemble, with the ACL anthology returning 7,500 papers for the search "argument* mining" at time of writing. The ACL Workshop on AM is running its twelfth edition in 2025, and the area is set to play an even more lynchpin role in NLP more broadly as interest in the capabilities of large language models to perform reasoning grows rapidly.

Despite a significant pedigree of research, the area of AM suffers from some significant challenges. First of all, as Ruosch et al. (2023) have demonstrated, reproducibility of results in AM is a pressing issue. Secondly, the challenge of reproducibility is compounded by the fact that AM comprises many interdependent tasks, and different studies have focused on different combinations of these subtasks, making it very difficult either to compare between systems or to leverage previous work in tackling different parts of the pipeline. Thirdly, even to the extent that different components might successfully be redeployed and harnessed in combination, interoperability remains a key challenge because basic conceptions and intuitions of argument structure are baked in to ad hoc representation languages which do not support interchange. Finally, there is a lack of AM tools, with most of the advancements in the area remaining isolated research prototypes (Kawarada et al., 2024; Cabessa et al., 2025; Pojoni et al., 2023; Chen et al., 2024; Gorur et al., 2025; Cabessa et al., 2024; Mancini et al., 2024; Habernal et al., 2024; Schaller et al., 2024). Our goal in this work is to address these challenges through Open Argument Mining Framework (oAMF), a solution that standardises and streamlines AM while preserving the ingenuity and creativity that have been hallmarks of research in the area.

The rest of this paper is organised as follows. Section 2 introduces xAIF, the data format enabling seamless communication in oAMF. Section 3 outlines the development, deployment, usage, and the existing modules in oAMF. Section 4 presents AMF-

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), pages 318–328

Compatible Tools, including visualisation and evaluation modules. Section 5 evaluates oAMF, and related works are detailed in Section 6. The release of oAMF is in Section 7, with key contributions and future directions in Section 8.

2 Data Format

The Argument Interchange Format, AIF, is a mature, well-established and widely used standard for computational representation of argumentation (Chesñevar et al., 2006). It provides a formally specified ontology with which to capture basic notions of the structure of arguments as graphs (Rahwan et al., 2011). The AIF (and its extensions to handle argument situated in dialogue) captures propositions (including a special subclass of propositions that refer to discourse events and are referred to as locutions), and relations between propositions (including relations of inference, conflict and rephrase, plus additional relations capturing illocutionary function and protocol-governed transition in dialogical settings). In combination with various parts of the Argument Web ecosystem (Lawrence et al., 2023), the AIF is currently used in representing the largest extant datasets of annotated argumentation (Hautli-Janisz et al., 2022).

The AIF imposes a number of well-formedness constraints on the data it handles, including that relations must have exactly one consequent and at least one antecedent, that propositions can only be interconnected via relations, and so on (Rahwan et al., 2007). In an environment of incremental processing where parts of an argument structure represented in AIF may be added piecemeal such constraints are too onerous. In addition, it may be appropriate to markup initial discourse with additional intermediate annotation that is not captured by AIF simpliciter. For both of these reasons, basic AIF is extended as "xAIF" which offers a mechanism by which AIF structure can both be underspecified (with respect to constraints) and overspecified (with respect to structural markup), and is made available as a convenient JSON language. xAIF provides the interlingua of the open argument mining framework, acting as the language for both input and output of all the modules. An example of xAIF is available in Figure 1 and a complete documentation is available at https://github.com/arg-tech/xaif/blob/main/docs/tutorial.md.

3 The Open Argument Mining Framework (oAMF)

oAMF is a modular, open-source framework designed to facilitate end-to-end AM by integrating diverse AM modules, fostering interoperability, flexibility, scalability, and ease of use across various AM tasks through multiple interfaces, including drag-and-drop, web-based, and programming APIs. oAMF empowers researchers and developers to create customisable, reproducible, and scalable AM workflows (pipelines) by seamlessly integrating multiple modules within a single framework, thereby simplifying the process of building and experimenting with AM pipelines and enhancing both development and research efficiency.

Currently, the framework includes 17 open-source AM modules (see Table 2), all deployed on the oAMF server and available on GitHub for community contributions. New modules can be added, with each module expected to follow specific input/output formats, implementation guidelines, and configuration requirements (see Section 3.1).

oAMF can be accessed through multiple interfaces. The web interface (3.3.3) allows the selection and execution of pre-built AM pipelines. A drag-and-drop interface (3.3.2) lets users construct AM pipelines based on deployed components on the oAMF server. The oAMF Python library can be installed to deploy modules locally and create AM pipelines using either the locally deployed modules or those on the oAMF server or both, offering a flexible solution for both local and remote execution.

3.1 oAMF Module Development

oAMF allows developers to extend its capabilities by adding new modules, following a structured development approach that ensures interoperability.

Input-Output Format: Each module uses xAIF for input and output to ensure interoperability. To streamline the process, oAMF offers a Python library, which simplifies input and output formatting into the required xAIF structure. As shown in Figure 1, this library simplifies xAIF manipulation, aiding developers in managing argument units and relations. For more details on installation and usage, visit the PyPI package page at https://pypi.org/project/xaif/.

Implementation: Modules are implemented as Flask-based web services to ensure smooth deployment. Each module is containerised to isolate its

```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 = AIF("here is the text.") # Or initialise with raw text
# 1. Adding entries
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 1: `xaif` package to manipulate xAIF data.

dependencies. For detailed information on new module development process, refer to Appendix A. An empty oAMF project that can be cloned and customised to add new modules is available at https://github.com/arg-tech/AMF_NOOP/.

3.2 oAMF Deployment

The release of oAMF includes the open-source Python library, available at https://pypi.org/project/oamf/, which can be installed via `pip install oAMF`. It is used to deploy oAMF modules locally, create and run AM pipelines using either locally deployed or remote modules (see Section 3.3.1). The user provides the GitHub repository link (specified as 'repo') for local deployment or URLs for remote modules (specified as 'ws'), along with the web-service route and tag. The library retrieves the 'repo's and deploys the modules locally as containerised Flask applications, dynamically loading only the specified modules. The script in Figure 2 shows the deployment and loading of the specified modules. Loaded modules are referenced using their tags for pipeline construction.

3.3 oAMF for Creating and Running Pipelines

oAMF offers interfaces for building and executing AM pipelines, supporting all technical levels including API for advanced customisation, a drag-and-drop interface for quick setup, and a web interface for easy execution.

```python
from oamf import oAMF
oamf = oAMF() # Initialise the library
# Specify the URL, module type ('repo' or 'ws'), route, and tag. Use
      multiple tags to use the same module multiple times.
modules_to_load = [
    ("https://github.com/arg-tech/default_turninator.git", "repo",
          "turninator-01", "turninator"),
    ("https://github.com/arg-tech/default_segmenter.git", "repo",
          "segmenter-01", "segmenter"),
    ("https://github.com/arg-tech/default_segmenter.git", "repo",
          "segmenter-01", "segmenter2"),
    ("http://bert-te.amfws.arg.tech/bert-te", "ws", "bert-te", "bert-te")
    ]
oamf.load_modules(modules_to_load) # Load and deploy the modules
```

Figure 2: Install and load modules with oAMF API.

3.3.1 Programming API

The programming API allows defining a pipeline as a directed graph by specifying and connecting modules through their associated tags. The pipeline can be executed by providing an input file, typically in xAIF format. The script shown in Figure 3, shows how to build and execute a pipeline using both local and remote modules. See the Jupyter Notebook

```python

from oamf import oAMF
# Initialize the library
oamf = oAMF()
# Define pipeline as a graph
pipeline_graph = [
        ("turninator", "segmenter"),
        ("turninator", "segmenter2"),
        ("segmenter", "bert-te"),
        ("segmenter2", "bert-te2")
        ]
oamf.pipelineExecutor(pipeline_graph, "input_file.json")
```

Figure 3: Create and execute pipeline with oAMF API.

for a step-by-step guide on using deployed web services to construct and execute pipelines here, and a Python script for deploying modules locally and building an AP pipeline here.

3.3.2 Drag-and-Drop Interface

oAMF integrates with n8n, an open-source workflow automation tool (https://n8n.io), available at https://n8n.oamf.arg.tech/[1], 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 8n8 workflow.

[1]Login with email: oamf-user@arg.tech; Password: Password1

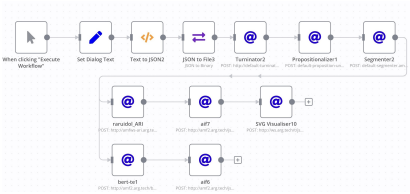

Figure 4: Drag-and-drop interface in n8n.

3.3.3 Web Interface

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.

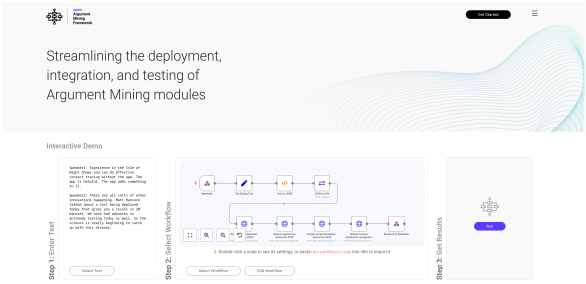

Figure 5: Web interface.

3.4 Modules

The oAMF comes with a series of modules covering basic functionalities for natural language argument analysis, including argument segmentation, classification and relation identification, among others. Argumentation is a theoretically rich topic, with multiple ways of representing similar concepts. The oAMF allows for different modules based on different theories of argumentation to co-exist and work together. These variations are reflected in the module description provided below, in which it can be observed how fundamental concepts such as the boundaries of an argumentative span (e.g., proposition, component, unit) or the relations between them (e.g., attack, support, conflict, inference, rephrase) differ between modules. The oAMF, therefore, makes it possible to create and evaluate pipelines in which modules designed based on different theories of argumentation work together.

default-turn-separator–Gemechu-2025 (DTSG). This module addresses the task of segmenting unstructured text into turns of speech. These turns include the complete speech transcripts divided by speaker interventions in the case of dialogue argumentation, or a unique segment in the case of monologue argumentation.

default-segmenter–Gemechu-2025 (DSG). This module takes unstructured text or text segmented into turns of speech as its input, and produces a structured segmented output. The process involves dividing the complete text transcripts into sequences of smaller units of locutions related with transition relations that capture the flow of discourse.

targer-segmenter–Chernodub-2019 (TARGER). The TARGER (Chernodub et al., 2019) module addresses the task of discourse segmentation. It therefore processes unstructured text into segmented argumentative discourse units.

deepseek-segmenter–Gemechu-2025 (DSS). This module utilises the deepseek-r1.1.5b model to segment argumentative text into discourse units. Using a few-shot prompting approach, it segments unstructured text into argumentative discourse units.

default-anaphora-resolver–Jo-2019 (DARJ). Given an xAIF document containing segmented locutions, this module addresses the task of resolving anaphora in co-references completing the locutions containing pronouns with the missing information as described in (Jo et al., 2019).

simple-propositionaliser–Gemechu-2025 (SPG). The goal of this module is to extract argumentative propositions from the locutions identified in the discourse. It therefore takes a text input segmented into locution nodes and analyses them extracting the argumentative propositions into information nodes. Finally, the model anchors information and locution nodes with illocutionary acts, forming a complete graph representation of the discourse.

cascade-propositionaliser–Jo-2019 (CPJ). This module extracts argument propositions using a cascaded approach with seven sequential steps. Starting from a set of utterances, it resolves anaphora, then extracts the locutions and performs a series of checks (such as whether it contains reported speech). The subject is then reconstructed, and with a final revision the argument proposition is extracted (Jo et al., 2019).

decompositional-argument-miner–Gemechu-2019 (DAMG). Given a text segmented into argument components, this module identifies inference and conflict relations between these components (Gemechu and Reed, 2019).

default-textual-entailment-recogniser–Gemechu-2025 (DTERG). Starting from an unstructured set of argument propositions, this module pre-trained on textual entailment tasks identifies positive and negative entailment between proposition pairs.

simple-argument-relation-identifier–Moslemnejad-2025 (SARIM). This module uses Support Vector Machine trained to identify attack and support relations given a set of argument propositions from an already segmented text input.

argument-relation-identifier–RuizDolz-2021 (ARIR). This module implements a fine-tuned RoBERTa encoder that performs a sentence-pair 4-class classification task, identifying non-related, inference, conflict, and rephrase relations between pairs of argument propositions from a set of already segmented text (Ruiz-Dolz et al., 2021).

decoder-relation-identifier–Gemechu-2024 (DRIG). This model is the implementation of Gemechu et al. (2024) decoder-only architecture, which is fine-tuned in classifying argument relations into 4-classes using sequence pair classification setup.

targer-AM–Chernodub-2019 (TARGER-AM). The TARGER (Chernodub et al., 2019) module classifies the argument relation between a pair of argument units into support, attack and none.

deepseek-relation-miner–Gemechu-2025 (DSRM). Given a pair of argument units, this module employs the deepseek-r1.1.5b model with few-shot prompting to classify their relationship as support, attack, or none, capturing argumentative connections between discourse components.

pragma-dialectics-scheme-classifier–Zografistou-2025 (PDSCZ). The aim of this module is to identify the three pragma-dialectical argumentation schemes of comparison, symptomatic, and causal argumentation. Taking the set of already segmented argument propositions and the inference relations between them as its input, this module classifies the existing inference relations into one of the three scheme classes.

walton-scheme-classifier–RuizDolz-2025 (WSCR). Given a set of already identified inference relations between argument propositions, this module classifies the inference into one group of Walton's argumentation schemes such as case-based, or practical reasoning arguments among others (Walton and Macagno, 2015), thus providing further insight about the argumentative structures identified in the discourse.

proposition-type-classifier–RuizDolz-2025 (PTCR). Starting from a set of already segmented argument propositions, this module, consisting of a fine-tuned RoBERTa encoder, classifies them into three possible classes depending on their argumentative role: value, fact, or policy.

4 AMF-Compatible Tools

Within the oAMF ecosystem, several tools are available to facilitate the management input, output visualisation, and evaluation.

whisper-speech-to-text-2025 (WSTT): This module converts spoken language into text using the Whisper model (Radford et al., 2023). It enables transcription of speech, supporting AM tasks that involve processing speech input data.

svg-visualiser-2025 (SV): This module is used to convert the xAIF output from each oAMF module into SVG format, enabling easy visualisation of the argument structure produced by oAMF modules.

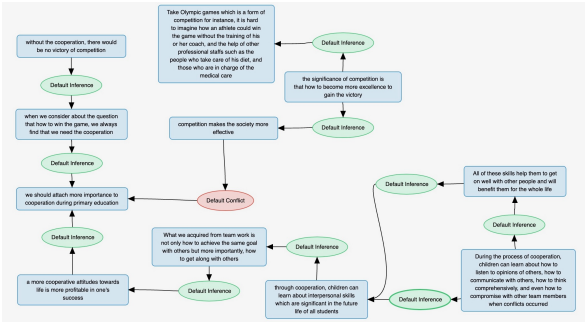

Figure 6: An argument map generated by the visualiser.

CASS-Moslemnejad-2025 (CASS): This tool compares two xAIF files with a Combined Argument Similarity Score (Duthie et al., 2016). Originally part of the Argument Analytics suite (Lawrence et al., 2016), it is now available as an oAMF module. CASS combines scores for multiple aspects of AM, providing a comprehensive assessment of AM systems. It now also reports individual metrics comparing the argument graphs (Macro F1, Accuracy, Text Similarity, Kappa, U-Alpha).

5 Evaluation

We evaluate oAMF by configuring three AM pipelines and comparing their performance against SOTA methods, pipelines **A**, **B** and **C**:

A: DTSG → DSG → SPG → DTERG.

B: DTSG → TARGER → CPJ → DRIG.

C: DTSG → TARGER → CPJ → DTREG → DRIG.

Following the comparison approaches, the pipelines are evaluated on Argument Component Identification (ACI) and Argument Relation Identification (ARI) tasks. These pipelines are evaluated on the AAEC dataset (Stab and Gurevych, 2017), and compared with end-to-end AM approaches (Eger et al., 2017; Morio et al., 2022; Bao et al., 2022) on ACI and ARI. Additionally, the pipelines are compared with individual models that have achieved SOTA results in cross-dataset evaluations for the ARI task (ARI*) (Ruiz-Dolz et al., 2025).

Pipeline	ACI	ARI	ARI*
Pipeline A	54.85	24.76	-
Pipeline B	56.32	32.65	-
Pipeline C	56.32	-	**47.37**
Ruiz-Dolz et al. (2025)	-		42.00
Eger et al. (2017)	66.21	29.56	-
Morio et al. (2022)	**76.55**	**54.66**	-
Bao et al. (2022)	75.94	50.08	-

Table 1: oAMF evaluation and comparison works.

Evaluation Metrics. For ACI, we treat it as a span detection task and compute the F1 score for exact matches, while for ARI, we compute the F1 score for classification of argument pairs.

Result. The pipelines match SOTA performance while offering a simplified drag-and-drop process for AM tasks. oAMF models were not trained on the AAEC dataset used for this evaluation, yet achieve comparable performance. Notably, oAMF modules outperform the cross-dataset performance of SOTA models on ARI. LLM-based modules are slower; e.g. DSRM takes 19.461s for a single sample on the ARI task, whereas DTERG completes it in 0.345s.

6 Related work

In the broader NLP landscape, tools like AllenNLP (Gardner et al., 2018) offer modularity for general-purpose NLP tasks while specialised tools, like TweetNLP (Camacho-Collados et al., 2022), focus on specific tasks like sentiment analysis. Aside from some tools addressing specific AM tasks, like TARGER (Chernodub et al., 2019), there is no tool offering a complete and robust AM solution.

There have been recent advancements in research that propose end-to-end approaches for AM. These approaches combine standard tasks, such as ACI and ARI, into unified workflows. For instance, Eger et al. (2017) frame AM as a token-based sequence tagging task, classifying tokens into argument components (premise, conclusion) and their respective relations (support, attack) using the BIO tagging approach. Morio et al. (2022) propose an end-to-end cross-corpus training strategy, while Bao et al. (2022) introduce a generative framework leveraging a constrained pointer mechanism and reconstructed positional encoding. However, these remain research prototypes, rather than fully developed tools ready for deployment by end users. oAMF emerges as a solution to address these gaps, offering a unified platform that orchestrates various AM modules, providing a comprehensive and scalable tool for diverse AM tasks with easy-to-use interfaces for both local and remote execution.

7 Release

oAMF is released as an open-source framework under the LGPLv3 license. All resources, including links to source code, APIs, web applications, and documentation, are available through the web page at https://oAMF.arg.tech. The Github page is at https://github.com/arg-tech/oAMF. The oAMF Python package is on PyPI: https://pypi.org/project/oAMF/. The drag-and-drop interface is available at https://n8n.oamf.arg.tech/. The xAIF library is available at https://libraries.io/pypi/xaif. Complete documentation of oAMF is available at https://github.com/arg-tech/oAMF/blob/main/docs/tutorial.md.

8 Conclusion

The oAMF presents a significant advancement in the field of AM by providing a modular, scalable, and interoperable platform. By integrating several AM modules, oAMF enables researchers and developers to construct, experiment with, and deploy AM pipelines with minimal effort. Its flexible interfaces, including Python APIs and visual tools, cater to both technical and non-technical users. With its open-source nature, scalability, and user-friendly design, oAMF promotes collaboration and advances AM research and applications.

While oAMF achieves comparable performance to state-of-the-art results despite not being trained on the evaluation dataset, some modules lag behind models trained and tested on the same data. This highlights the platform's strong generalisability while pointing to opportunities for targeted improvements. Future work will extend evaluations to additional datasets, improve component accuracy, and foster collaborations to encourage broader adoption and incorporate user feedback. These efforts aim to further establish oAMF as a versatile and effective tool for advancing argument mining research and applications.

Acknowledgements

This work has been supported in part by: the Swiss National Science Foundation under grant 10001FM200857, "Mining argumentative patterns in context"; the European Media Information Fund under grant 268755; Volkswagen Stiftung Foundation under grant 98 543, "Deliberation Laboratory"; 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.

Limitations

This work has several limitations. First, the evaluation is currently based on a single dataset, providing an initial but limited indication of oAMF's robustness across different AM scenarios. Second, although some modules achieve comparable performance to state-of-the-art models despite not being trained on the evaluation dataset, they still trail behind models trained and tested on the same data, highlighting room for targeted improvements. Third, the platform's real-world adoption and usability remain to be validated through broader collaborations and user studies. Addressing these limitations will be a priority in future work to enhance oAMF's effectiveness and applicability.

References

Jianzhu Bao, Yuhang He, Yang Sun, Bin Liang, Jiachen Du, Bing Qin, Min Yang, and Ruifeng Xu. 2022. A generative model for end-to-end argument mining with reconstructed positional encoding and constrained pointer mechanism. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 10437–10449.

Jérémie Cabessa, Hugo Hernault, and Umer Mushtaq. 2024. In-context learning and fine-tuning gpt for argument mining. *arXiv preprint arXiv:2406.06699*.

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.

Jose Camacho-Collados, Kiamehr Rezaee, Talayeh Riahi, Asahi Ushio, Daniel Loureiro, Dimosthenis Antypas, Joanne Boisson, Luis Espinosa Anke, Fangyu Liu, and Eugenio Martínez-Cámara. 2022. Tweetnlp: Cutting-edge natural language processing for social media. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–49.

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.

C. Chesñevar, J. McGinnis, S. Modgil, I. Rahwan, C. Reed, G. Simari, M South, G. Vreeswijk, and S. Willmott. 2006. Towards an argument interchange format. *The Knowledge Engineering Review*, 21(4):293–316.

Rory Duthie, John Lawrence, Katarzyna Budzynska, and Chris Reed. 2016. The cass technique for evaluating the performance of argument mining. In *Proceedings of the Third Workshop on Argument Mining (ArgMining2016)*, pages 40–49.

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

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F Liu, Matthew E Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. Allennlp: A deep semantic natural language processing platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 1–6.

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. 2025. Can large language models perform relation-based argument mining? In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 8518–8534.

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 Thirteenth Language Resources and Evaluation Conference*, pages 3291–3300, Marseille, France. European Language Resources Association.

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.

John Lawrence, Rory Duthie, Katarzyna Budzynska, and Chris Reed. 2016. Argument Analytics. In *6th International Conference on Computational Models of Argument, COMMA 2016*, volume 287 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

John Lawrence and Chris Reed. 2019. Argument mining: A survey. *Computational Linguistics*, 45(4):765–818.

John Lawrence, Jacky Visser, and Chris Reed. 2023. Translational argument technology: Engineering a step change in the argument web. *Journal of Web Semantics*, 77:100786.

Marco Lippi and Paolo Torroni. 2016. Argumentation mining: State of the art and emerging trends. *ACM Trans. Internet Technol.*, 16(2).

Eleonora Mancini, Federico Ruggeri, Paolo Torroni, et al. 2024. Multimodal fallacy classification in political debates. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 170–178. Association for Computational Linguistics.

Gaku Morio, Hiroaki Ozaki, Terufumi Morishita, and Kohsuke Yanai. 2022. End-to-end argument mining with cross-corpora multi-task learning. *Transactions of the Association for Computational Linguistics*, 10:639–658.

Mircea-Luchian Pojoni, Lorik Dumani, and Ralf Schenkel. 2023. Argument-mining from podcasts using chatgpt. In *ICCBR Workshops*, pages 129–144.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2023. Robust speech recognition via large-scale weak supervision. In *International conference on machine learning*, pages 28492–28518. PMLR.

Iyad Rahwan, Bita Banihashemi, Chris Reed, Douglas Walton, and Sherief Abdallah. 2011. Representing and classifying arguments on the semantic web. *The Knowledge Engineering Review*, 26(4):487–511.

Iyad Rahwan, Fouad Zablith, and Chris Reed. 2007. Laying the foundations for a world wide argument web. *Artificial Intelligence*, 171(10):897–921.

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, Debela Gemechu, Zlata Kikteva, and Chris Reed. 2025. Looking at the unseen: Effective sampling of non-related propositions for argument mining. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 2131–2143. Association for Computational Linguistics.

Florian Ruosch, Cristina Sarasua, and Abraham Bernstein. 2023. DREAM: Deployment of recombination and ensembles in argument mining. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5277–5290, Singapore. Association for Computational Linguistics.

Nils-Jonathan Schaller, Andrea Horbach, Lars Ingver Höft, Yuning Ding, Jan Luca Bahr, Jennifer Meyer, and Thorben Jansen. 2024. Darius: A comprehensive learner corpus for argument mining in german-language essays. In *Proceedings of the 2024 joint international conference on computational linguistics, language resources and evaluation (LREC-COLING 2024)*, pages 4356–4367.

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

Manfred Stede and Jodi Schneider. 2019. *Argumentation Mining*. Morgan Claypool.

Douglas Walton and Fabrizio Macagno. 2015. A classification system for argumentation schemes. *Argument & Computation*, 6(3):219–245.

A New Module Development

The oAMF module is a web service that is dockerised to ensure portability and scalability. It is built using the Flask framework, which is a lightweight Python web framework for creating RESTful services. The module takes and outputs xAIF data.

- Webservice: The application exposes a set of endpoints that allow users to interact with the module through HTTP requests.
- Dockerised: The module is encapsulated in a Docker container for easy deployment. The container is configured using `Dockerfile` and `docker-compose.yaml`.

A.1 Project Structure

The project follows a standard web application structure with the following components:

- `config/metadata.yaml`: Contains metadata information about the module (See A.2).
- `project_src_dir/`: The directory containing the application code, including Flask routes and logic.
- `boot.sh`: A shell script to activate the virtual environment and launch the application.
- `docker-compose.yaml`: Defines the Docker service and how the application is built and run.
- `Dockerfile`: Specifies the Docker image, environment, and installation of dependencies.
- `requirements.txt`: Lists the Python dependencies required by the project.

A.2 Metadata Configuration (`config/metadata.yaml`)

The metadata file provides essential information about the module, including:

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

A.3 Flask Application Routes

- Index Route (`/`): Displays the contents of the `README.md` file, serving as documentation route.
- AMF Module Route: Any route name can be used.
  - The POST requests are typically used to upload xAIF file, apply the module logic. The response is then returned as a JSON object containing the updated xAIF data.
  - The GET request is used to provide the documentation and the metadata.

A.4 Summary of Steps to Develop an oAMF Module

To create a custom oAMF module, follow these steps:

1. Clone the NOOP template from the repository: https://github.com/arg-tech/AMF_NOOP.
2. Modify Metadata: Update `metadata.yaml` with module details.
3. Implement Core Logic: Modify `routes.py` to add module functionality.
4. Integrate with xAIF: Use `xaif` library to manipulate xAIF data.
5. Configure Docker: Ensure `Dockerfile` and `docker-compose.yaml` are set up.
6. Documentation: Update the `README.md` for usage instructions.

Module	Input	Output	Web-Service URL	Repo URL
DTSG	Unsegmented text and no structure.	Text segmented into turns (i.e. contiguous text from one speaker in the case of dialogue; NOOP in the case of monologue).	http://default-turninator.amfws.arg.tech/turninator-01	https://github.com/arg-tech/default_turninator
DSG	Unsegmented text; no structure.	Segmented text; structure containing L-nodes with IDs crossreferring to those in SPAN tags.	http://default-segmenter.amfws.arg.tech/segmenter-01	https://github.com/arg-tech/default_segmenter
TARGER	Unsegmented text; no structure.	Segmented text; structure containing L-nodes with IDs crossreferring to those in SPAN tags.	http://targer.amfws.arg.tech/targer-segmenter	https://github.com/arg-tech/targer
DSS	Unsegmented text; no structure.	Segmented text; structure containing L-nodes with IDs crossreferring to those in SPAN tags.	http://amf-llm.amfws.staging.arg.tech/segmenter	https://github.com/arg-tech/oamf_llm
DARJ	Segmented locutions.	Resolve co-references in locution nodes.	cascading-propositionUnitiser.amfws.arg.tech/anaphora-01	https://github.com/arg-tech/cascading_propositionaliser
SPG	Segmented text; structure containing L-nodes.	Segmented text segmented; structure containing L-nodes anchoring YA-nodes connected to I-nodes.	http://default-proposition-unitiser.amfws.arg.tech/propositionUnitizer-01	https://github.com/arg-tech/proposition-unitizer
CPJ	Segmented text ; structure containing L-nodes.	Segmented text; structure containing L-nodes anchoring YA-nodes connected to I-nodes.	http://cascading-propositionUnitiser.amfws.arg.tech/propositionaliser-cascading	https://github.com/arg-tech/cascading_propositionaliser
DAMG	Segmented text; with I-nodes.	Segmented text; with I-nodes connected with RA and CA nodes.	http://dam.amfws.arg.tech/dam-03	https://github.com/arg-tech/dam
DTERG	Segmented text; with I-nodes.	Segmented text; structure with I-nodes connected with RA nodes.	http://bert-te.amfws.arg.tech/bert-te	https://github.com/arg-tech/bert-te
PDSCZ	Segmented text; structure with I-nodes connected with RA nodes.	Segmented text; structure with I-nodes connected with RA nodes specified by pragma-dialectical scheme type.	http://amfws-schemeclassifier.arg.tech/schemes_clsf	https://github.com/arg-tech/AMF_Scheme_Classifier2
SARIM	xAIF file with the I-nodes.	xAIF file with I-Nodes and relations nodes.	http://amfws-rp.arg.tech/somaye	https://github.com/arg-tech/AMF-RP
ARIR	xAIF file containing propositional argumentative nodes.	xAIF file with the complete propositional argument graph covering three argumentative relation (RA, CA, or MA)	http://amfws-ari.arg.tech/	https://github.com/arg-tech/AMF_ARI
TARGER-AM	xAIF file containing propositional argumentative nodes.	xAIF file with the complete propositional argument graph covering three argumentative relation (RA, CA, or MA)	http://targer.amfws.arg.tech/targer-am	https://github.com/arg-tech/targer/
DRIG	xAIF file containing the I nodes.	Segmented text; structure with I-nodes connected with RA,MA and CA nodes.	http://vanilla-dialogpt-am.amfws.arg.tech/caasra	https://github.com/arg-tech/dialogpt-am-vanila
DSRM	xAIF file containing the I nodes.	Segmented text; structure with I-nodes connected with RA,MA and CA nodes.	http://amf-llm.amfws.staging.arg.tech/relation_identifier	https://github.com/arg-tech/oamf_llm
WSCR	xAIF file containing I nodes and the RA between them.	xAIF file where the "Default Inference" have been replaced by argumentation scheme (e.g., "Argument From Analogy").	http://amf-schemes.amfws.arg.tech	https://github.com/arg-tech/AMF_SchemeClassifier
PTCR	xAIF file with I-Nodes.	xAIF file with the "propositionClassifier" key containing I-Nodes with one of "Value", "Policy", or "Fact" categories.	http://amf-ptc.amfws.arg.tech	https://github.com/arg-tech/AMF_PTC_VFP
†CASS	Two xAIF with the same text	CASS, Macro F1, Accuracy, Text Similarity, Kappa, U-Alpha	http://amf-evaluation-metrics.amfws.arg.tech	https://github.com/arg-tech/amf-evaluation-metrics
†WSTT	Audio Input	xAIF with the text field populated with transcription	realtime-backend.amfws.arg.tech/transcribe_whisper-0	https://github.com/arg-tech/realtime-backend
†SV	xAIF	SVG	http://svg.amfws.arg.tech	https://github.com/arg-tech/svg-visualiser

Table 2: Summary of the oAMF modules and related tools (the latter modules marked by †).