# ArgServices: A Microservice-Based Architecture for Argumentation Machines

Mirko Lenz[1 [0000−0002−7720−0436]], Lorik Dumani[1 [0000−0001−9567−1699]], Ralf Schenkel[1 [0000−0001−5379−5191]], and Ralph Bergmann[1,2 [0000−0002−5515−7158]]

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

**Abstract.** Argumentation is ubiquitous, and the development of argumentation machines could greatly assist humans in managing and navigating argumentation. However, the development of such systems is hindered by the lack of common standards and suitable tools, leading to ad-hoc solutions with little reuse value. Towards a more unified approach, we present an extensible microservice-based architecture for argumentation machines. Being built on the established gRPC framework, it provides strongly typed interfaces for the following services: (i) Argument Mining, (ii) Case-Based Reasoning on Arguments, (iii) Argument Retrieval and Ranking, and (iv) Quality Assessment of Arguments. Our system is designed to be extensible, allowing for easy integration of new tasks. We demonstrate the feasibility of our architecture via a proof-of-concept implementation and provide additional supplementary resources, such as a REST API gateway. Our contributions are publicly available on GitHub under the permissive MIT license.

**Keywords:** Argumentation · Argument Graphs · Microservices · gRPC · REST · Natural Language Processing · Open Source

## 1 Introduction

Living in an ever-changing world, we are constantly confronted with new information and, based on it, have to make decisions. With the advent of the Internet, computers have become an integral part of this process. While traditional Web search engines mostly rely on textual similarity, and thus require users to manually extract and analyze relevant information, domain-specific systems could incorporate other sources of knowledge to provide assistance. However, even within a single field—such as argumentation—many competing solutions exist without a common standard or interface. As a result, the development of ad-hoc solutions is often necessary, which are not easily reusable in other contexts. Argumentation machines [30] for example may offer a variety of services like retrieval and validation, but contributions in the domain of Computational Argumentation (CA) focus mainly on one aspect of such a system.

In this paper, we present a microservice-based architecture for argumentation machines designed to be extensible and reusable. The target audience is researchers and developers who aim to build, extend, or only use specific functions of argumentation machines, without reinventing the wheel. Our ultimate vision is to allow other researchers to rewrite one service using their own algorithms and integrate it into the existing architecture—enabling them to evaluate their approach in a larger framework. Our contributions are (i) service descriptions for common tasks in CA based on the established gRPC framework, (ii) a proof-of-concept implementation of the architecture showcasing its feasibility, and (iii) a collection of supplementary resources like a REST API gateway and ready-to use client/server libraries.

The remainder of this paper is structured as follows: In Section 2, we introduce the foundations necessary to understand our architecture, followed by a discussion of related work in Section 3. Section 4 presents the service definitions that are implemented in a proof-of-concept described in Section 5 and supported by supplementary resources introduced in Section 6. Finally, Section 7 discusses current limitations and Section 8 concludes our paper.

## 2 Foundations

Our proposed architecture for an argumentation machine is fundamentally based on argument graphs, so we briefly introduce them in this section. Since dealing with texts is essential in this domain, we also present some common Natural Language Processing (NLP) [5] concepts here. Lastly, we introduce some aspects of microservice-oriented backends like Representational State Transfer (REST) and gRPC.

### 2.1 Theoretical Argumentation and Argument Graphs

An argument is typically composed of a single *claim* that is supported or attacked by one or multiple *premises* [28]—these smallest units of an argument can be subsumed under the term Argumentative Discourse Units (ADUs) [28]. A claim itself may also support another claim and thus additionally act as a premise, making it possible to represent entire conversations. Moreover, an argument typically has one primary/central conclusion, the so-called *major claim*. This inductive structure already forms a directed graph—we call it *argument graph*.

According to Argument Interchange Format (AIF) [13], an argument graph is a tuple $G = (V, E)$ where $V$ is a set of nodes and $E \subseteq V \times V$ is a set of directed edges. The nodes are divided into atom nodes $A \subset V$ representing the ADUs and scheme nodes $S \subset V$ representing the relationships between them: $V = A \cup S$. Edges cannot be drawn between two atom nodes, so we define $E \subseteq V \times V \setminus A \times A$. Any sequential ordering of ADUs originating from the source text is lost in the AIF graph representation. To mitigate this, the annotation software Online Visualization of Arguments (OVA) [8] for example uses additional properties to store the position of each atom node in the text and consequently the order of

the ADUs. The creation of structured argument representations—including but not limited to graphs—is also known as Argument Mining (AM) [20].

To use more detailed semantics when representing the scheme nodes, argumentation schemes introduced by Walton et al. [41] may be used. Each scheme—for instance, *Expert Opinion*—explicitly describes the role of a claim and its fixed set of premises. In this example, the claim would be the conclusion of one premise representing an expert's opinion and the other premise representing the expert's expertise. To check the applicability of such a scheme to a relationship, the authors defined critical questions.

### 2.2 Argument Processing

Argument graphs contain two types of information—structure and semantics. The former refers to the graph-based representation (i.e., the nodes and edges), whereas the latter refers to the text of the nodes. Our system deeply integrated both aspects, so the following section will introduce the necessary concepts.

For the *structural* aspect, we use the wide variety of research on graph-based representations. A relevant field is Process-Oriented Case-Based Reasoning (POCBR), a variant of Case-Based Reasoning (CBR) [2,33] that focuses on graph-based workflows (e.g., business processes). The idea of CBR is to solve new problems by reusing solutions to problems similar to those that have been solved in the past by performing four steps: (i) *Retrieve* a set of similar cases from the so-called *case base*, (ii) *reuse* the found cases by *adapting* them to the new query, (iii) *revise* the adapted cases by checking their validity, and (iv) *retain* the new solution for future use. One central difference between case-based retrieval and Information Retrieval (IR) is the inclusion of structural information—in our case the argument graph.

When assessing the similarity between the atom nodes of two argument graphs in the CBR framework, we need to take into account the *semantic* aspect. Over the past few years, the use of language models—for instance, to compute the semantic similarity between texts via embeddings—has become a common practice in NLP. The basic idea is that words or sentences with similar meanings should be close to each other in a high-dimensional vector space. Using standard measures like the cosine distance, we can assess the similarity between two texts by comparing their embeddings/vectors. For the nodes of the scheme, embeddings could also be computed to assess their similarity, but taxonomy-based measures may be a better fit [40].

### 2.3 System Architectures

When designing a system, there are two common approaches: *monolithic* and *microservice-based* architectures. A monolith is a single codebase that contains all functionality of the system and is deployed as a single unit. On the contrary, a microservice-based architecture is composed of multiple coherent services that are deployed independently of each other. [4] In this paper, we committed to the latter for two main reasons: (i) The specific implementation of a single module

is entirely separated from the rest, meaning it is possible to combine different programming languages (e.g., Java and Python). (ii) We aim at providing an argumentation machine that allows other researchers to quickly swap out a single module with their own implementation and evaluate it in a larger context. Although both can in principle be achieved with a monolithic architecture, the microservice-one was the more natural choice for us, since general-purpose monoliths are not yet widely used (see Section 3). In addition, a microservice architecture makes it easier to scale horizontally, which means that it translates well into production environments. In the following, we briefly introduce this architecture in more detail.

According to Jamshidi et al. [18], a service/module of a microservice-based system offers "access to its internal logic and data through a well-defined network interface"—the so-called Application Programming Interface (API). As of 2024, the most common style of these systems is REST, which uses a fixed set of URL-based endpoints and Hypertext Transfer Protocol (HTTP) operations to access the functionality of a service. There are also other options like Simple Object Access Protocol (SOAP), GraphQL, and gRPC, each having its own set of advantages and drawbacks. For our architecture, we ultimately settled on gRPC—a Remote Procedure Call framework developed by Google on top of the modern HTTP/2 protocol specifically for microservice backends.[3] Compared to the established REST, it has the following differences:

**Stronger typing** gRPC uses Protocol Buffers (Protobuf) for data serialization, which allows for a more strict definition of the data types used in the API. This strong contract between client and server removes some potential sources of bugs (e.g., sending strings instead of integers).

**Code generation** The use of Protobuf to define services provides a code generation tool that creates client and server stubs for most major programming languages. This means that compared to REST, the user does not have to deal with the low-level details of the HTTP protocol.

**Binary data transfer** The messages sent between the client and the server are encoded in a binary format, which is much more compact than the textual JavaScript Object Notation (JSON) format used by REST. Note that this does not have a negative impact on readability, as it only applies to the transfer itself—that is, any Protobuf message can be serialized to a JSON object as well.

These advantages come at the cost of a steeper learning curve—for instance, developers need to learn a new domain-specific language and have to re-run the code generation tool after changes to the service definitions. With its reliance on HTTP/2, gRPC cannot be natively used in browsers and requires proxies to work around this limitation (see Section 6 for our solution). Yet, due to the mentioned advantages, gRPC and Protobuf are already heavily used in Machine Learning (ML)—most prominently, they serve as the backbone for the official Tensorflow API.[4]

[3] `https://grpc.io`

[4] `https://github.com/tensorflow/serving`

## 3 Related Work

To the best of our knowledge, there is almost no work on the architecture of argument machines as we present them. Works up to the mid-2010s often only presented computational models of argument which are more concerned with argumentation theory—that is, the construction of arguments or a whole argumentation. In the following, we consequently not only collected works describing entire argumentation machines, but also works concerned with only one aspect of it (e.g., argument retrieval).

The development of argumentation machines began in the mid-1990s and included work on argumentative dialog planning [29], applications of argument schemas in Artificial Intelligence (AI) [31], and argumentation engines capable of handling a large number of topics [30]. The AIF (see Section 2.1) was later extended to handle dialogical argumentation [32]. In the following years, the argument annotation tools ArgueBlogging [9] and OVA+ [19] were developed. They specialized in constructing discussions about blogs and annotating plain texts with argument graphs, respectively.

Slonim et al. [36] presented Project Debater, an autonomous debating system that is capable of discussing with people a wide variety of topics and taking certain positions. Even before the era of Large Language Models (LLMs), their system was capable of conducting a discussion—that is, understanding users' viewpoints and generating suitable arguments. In their work, they described the architecture of the system and conducted an evaluation that included several debate topics. An alternative architecture for an argumentation machine has been proposed by Bergmann et al. [6] as part of the ReCAP project. In our work, we build on their proposal and present an improved version that has been developed according to best practices in an effort to keep up with the rapidly changing field of CA.

Apart from this, we are only aware of work on stand-alone systems, which we present in the following. Wachsmuth et al. [39] proposed the first argument search engine (Args) known to us, which introduced a system that reads any free text user queries to search for arguments, and then presents relevant arguments from a pool of almost 300k previously mined and indexed arguments from five debate portals (i.e., Web content) in a ranking based on BM25F. Other projects used their system as a starting point for their own research—for instance, by reimplementing its most important properties or using their dataset for tasks like ranking [3,11]. However, despite all merits, their system does not cover the entire argumentation machine as we envision it.

Among others, Bondarenko et al. [11] have been setting up the CLEF lab Touché every year since 2020, where they used the Args dataset until 2022 and ClueWeb22 since 2023. The systems submitted by the participants can be uploaded to TIRA [16] to reproduce the results. Like Args, ArgumenText [37] is an argument search engine. First, it finds relevant documents in a large set of heterogeneous arbitrary Web sources using ElasticSearch, then it identifies relevant premises in them using Keras, assigns them stances by applying BiL-

STM, and ranks the premises by their classifier's confidence score. Its evaluation showed a high recall of 89% and a rather moderate precision of 47%.

Beyond the retrieval of arguments, Eden et al. [15] presented insights on the creation of a Key Point Analysis (KPA) system and highlighted some of the main challenges. KPA is concerned with extracting the main points from a collection of opinions, a service that may in the future also be incorporated into our proposed architecture. Romberg [34] tackles the problem that argumentation is often subjective and annotations are summarized with average or majority vote, resulting in minorities being ignored when learning. Therefore, she introduced PerspectifyMe, a method that combines subjective points of view by complementing an aggregated label with a subjectivity score. Heinisch et al. [17] addressed the subjectivity issue in annotation processes and found that classifiers incorporating relations between different annotators are beneficial even for predicting single-annotator labels. Building models that are aware of potentially subjective annotations is a crucial aspect in CA, so we plan to include this aspect in future iterations of our architecture.

## 4 Microservices for Argumentation

As mentioned in Section 1 and seen in Section 3, argumentation machines can vary greatly w.r.t. their functionality. Consequently, the main goal of our service definitions is to be easily extensible for tasks not envisioned by us. As a starting point, we identified the following tasks as common in CA: (i) argument mining, (ii) case-based reasoning on arguments, (iii) retrieval and ranking of arguments, and (iv) quality assessment of arguments. Please note the difference between (ii) and (iii): while the former integrates the structural information of entire graphs (see Section 2.2), the latter considers ADUs or claim-premise pairs. An overview of the different modules is given in Figure 1. There are two special services in our proposed architecture that were not part of the aforementioned list: (i) The argumentation base in the middle and (ii) the NLP service to the bottom right. All services either consume argument graphs or produce them, so we created a dedicated module to serve them. Some of the services also require NLP functionality, leading to the creation of a separate service for this task. All services are designed to work independently of each other (with the exception of the NLP service), but may be combined to form a complete argumentation machine. Further aspects of their orchestration are discussed in Section 5.

All services are defined using Protobuf and gRPC with their definitions publicly available on GitHub under the permissive MIT license.[5] These service definitions are also available from the Buf Schema Registry, which makes it possible to add them as dependencies in other gRPC-based project and provides users with a nicely formatted and up-to-date documentation.[6] We acknowledge that there may be varying requirements or the need for additional data depending on

[5] `https://github.com/recap-utr/arg-services`
[6] `https://buf.build/recap/arg-services`

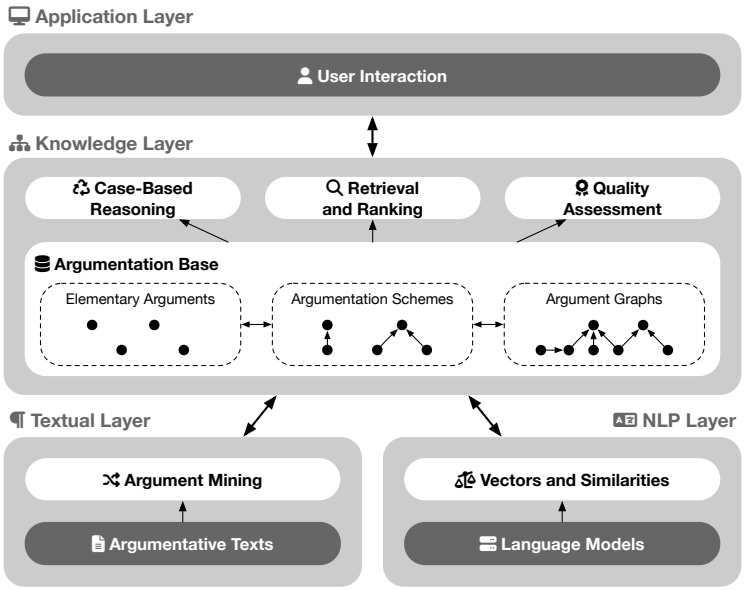


**Fig. 1.** Overview of our proposed architecture for microservice-based argumentation machines. Light gray boxes represent the different layers, dark gray ones external resources, and white ones the services exposed via our API.

the context. Consequently, each function described here allows arbitrary JSON-encoded data to be encoded as an optional parameter called *extras*.

In the following section, we introduce each service in detail and provide a list of all included functions. We also highlight some of the most important options/parameters that can be used to customize the behavior of the services. Due to their tight integration with the other services, we start with two core modules: the argumentation base (middle) and NLP service (bottom right). Subsequently, the remaining services mentioned above are introduced.

### 4.1 Argumentation Base

The core of our argumentation base is our own argument serialization format Argument Buffers (ArgueBuf) [21] first introduced at COMMA 2022. Given that it has been designed as a first-class citizen of Protobuf, the integration into our architecture is straightforward. Its formal semantics are based on the established AIF standard (see Section 2.1), but the storage format is designed to be more uniform and extensible at the same time. With regards to the *graph structure*, the following five main differences exist compared to argument graphs serialized to AIF: (i) The graph itself and each of its nodes/edges allow arbitrary key-value pairs to be stored. (ii) The sets containing the nodes and edges are represented as dictionaries with their respective IDs as keys to enforce uniqueness. (iii) Original textual resources and participants of a conversation can be

stored together with the graph. (iv) The major claim is explicitly marked as such. (v) Information about analysts and the creation/modification date can be stored. *Atom nodes* can not only store the text of the ADU, but also the position in the original text and the participant who made the statement. The link to the original text also allows for reconstruction of the sequential ordering of the argument as found in the source text. *Scheme nodes* do no longer use a free-text field to store the scheme name, but instead refer to a scheme from a predefined list through an enumeration. This decision makes parsing and serialization easier and more reliable, with the drawback that new schemes first need to be added to the Protobuf definition. All the mentioned changes are additions to the format, so an existing AIF graph can easily be converted to ArguebufGraph. An area where our format is currently lacking is the representation of dialogical argumentation, which is planned to be added in the future. This service offers a single function:

**Casebase** Given a list of filter criteria (expressed as regular expressions), return a list of argument graphs. The type of filters available depends on the implementation—thus the use of generic regexes—but may include factors like the corpus name, the serialization format, or the inclusion of schemes.

We have chosen to stick to the CBR terminology here to be consistent with some of our other services. Currently, only regex-based filtering is supported, but more advanced filtering options could be added in the future if the need arises.

### 4.2 Natural Language Processing

As outlined earlier, the use of language models—for instance, to compute the semantic similarity between texts via embeddings—has become a common practice in NLP. At the same time, these models are getting larger and larger, making it harder to use them on a regular computer. When combined with a microservice architecture, another challenge is that each service would need to load the model into memory, which is a waste of resources. We therefore chose to add a dedicated service for these needs with a central NlpConfig message that can be passed between individual services. It encodes (i) the language of the processing pipeline, (ii) the choice of the language model (multiple are also possible), (iii) the similarity measure to use, and (iv) the pooling function for plain word embeddings. Currently, this service focuses on determining semantic similarity between texts through embeddings. More general NLP tasks like named entity recognition or dependency parsing are not yet supported but could be added in the future if the need arises. To mitigate this restriction, we added a function to process texts with the Python library spaCy [25] and return the result as a binary representation. Please note that the goal of this service is to save resources by loading common base models once instead of requiring each service to load them individually. That also means that custom models (e.g., for classification) are not part of this service and need to be handled by the respective service implementations. It is also beyond the scope of this service to add generative models, as there already exist well-established interfaces like the OpenAI API for this purpose. The corresponding service offers the following functions:

**Vectors** Given a list of texts, it returns their embeddings of $n$ dimensions.
**Similarity** Given a list of text pairs, it returns their similarity score between 0 and 1.
**Spacy Document** Give a list of texts, return the corresponding spaCy documents. This is an optional service that is usable only with Python servers and clients.

### 4.3 Argument Mining

Having introduced the two cornerstones of our architecture, we now present the service responsible for building our argumentation base through AM. The set of functions is derived from an end-to-end pipline [23] for transforming plain texts into argument graphs consisting of multiple successive steps with an additional function for transforming a text to a graph without any intermediate steps.

**Segment Text** Given a natural language text, return the list of Elementary Discourse Units (EDUs).
**Classify ADUs** Given a list of EDUs, return the list of ADUs.
**Predict Major Claim** Given a list of ADUs, return a ranking of major claim candidates.
**Predict Polarity/Entailment** Given a list of ADUs, compute the cross product to generate claim-premise pairs and predict the polarities (i.e., support, attack, or neutral) between them.
**Construct Graph** Given all ADUs, the major claim, and the predicted polarities, construct the resulting graph using some heuristic.
**End-to-End Pipeline** Given a natural language text, return an argument graph.

With the advent of generative language models, we plan to add a function to the service that allows the generation of textual arguments and/or argument graphs from a given prompt. Although even the present functions can already be implemented using LLMs, the current set of features is more focused on the extraction of arguments from existing texts. As such, the envisioned generation function would enable the synthesis of new arguments.

### 4.4 Case-Based Reasoning on Arguments

With the methods in place to build the argumentation base, we have now reached the knowledge layer and can take advantage of them in our services. The retrieval functionality is motivated by our paper published at FLAIRS 2019 [7] that proposes a combination of semantic and structural similarity measures for CBR with argument graphs. The underlying paper for the adaptation service has been published at ICCBR 2023 [22] and proposes a hybrid approach combining WordNet with LLMs to adapt retrieved arguments. For all methods offered by the service, the cases and the user-provided are represented as argument graphs—enabling the user to specify the structure and the content of the desired argument. It offers the following functions:

**Retrieve** Given a collection of argument graphs (i.e., the casebase) and a user-defined query (that is also a graph), perform a search for the most similar argument in the casebase.

**Adapt** Given one retrieved graph and the user-defined query, perform a keyword-based adaptation with the goal of making the retrieved one more similar to the query. The function allows passing one or multiple rules to influence the process. One can decide to restrict the adaptation process to pure generalization, pure specialization, or a combination of both.

### 4.5 Argument Retrieval and Ranking

Argument Retrieval contains a wide range of IR tasks, including ranking and clustering—the former being at the heart of every IR system. At SIGIR 2021 [27] we presented an argument search system that ranks premises to queries according to the principle of TF-IDF (i.e., the more frequent premises of claims that are (more) similar to the query occur, the higher the score), as well as by the three (main) quality dimensions of cogency, reasonableness, and effectiveness [38]. At CIKM 2021 [14] we presented a work on fine granular clustering of arguments, as clustering is an essential part of our ranking approaches. The service offers the following functions:

**Statistical Ranking** Given a query and a list of ADUs, return a ranking of the given arguments based on frequency and specificity.

**Quality-Based Ranking** Given a query and a list of ADUs, return a ranking of the given arguments based on scores derived from a set of quality dimensions.

**Fine-Granular Clustering** Given a query and a list of ADUs, predict a set of scores used to assign them to fine-granular clusters.

### 4.6 Quality Assessment of Arguments

As implied in the previous section, we used argument quality in our work, which is why we also offer a dedicated service for this task. A work presented at CIKM 2023 [12] introduced a User Interface (UI) that takes two premises for a claim and not only decides for these two, which is more convincing for all 15 argument quality dimensions [38], but also provides an additional explanation together with the individual scores justifying why the particular decisions were made. Another work published at the ArgMining workshop at COLING 2022 [10] presented an end-to-end tool that reads any plain text and returns the so-called qualia structures (which express the meaning of lexical items from four viewpoints). With validation being a central part of argument quality—arguments containing disinformation have lower quality—we also provide a further service that is based on a work presented at the TMG workshop at ICCBR 2023 [26] able to predict the suitability of experts when cited for emphasizing statements. The quality service has the following functions:

**Quality Explanation** Given a claim and two premises, determine and explain which one is more convincing for all available quality dimensions as well as globally across all dimensions.

**Qualia Annotations** Given a text and a list of qualia patterns, compute the constituency tree and return the qualia role for each pattern.

**Expert Suitability** Given a premise and (optionally) the Google Scholar ID of a researcher, predict whether they are an expert on the given topic.

## 5 Proof-of-Concept

With all the necessary tools in place, we created a proof-of-concept implementation of our architecture that includes almost all the services and functions described in Section 4. One of the central goals of our work has been to create a machine that allows other researchers to reimplement a single module and gain the ability to perform experiments in a larger system. Consequently, some of the services are written in Python, while others use Java. The code is based on existing implementations originally written for the corresponding paper or was newly created for this work. Some services are even implemented through a LLM-based prompting strategy to showcase the flexibility of our architecture in keeping up with the latest trends in NLP. The code and additional instructions are available on GitHub under the permissive MIT license.[7] In the following section, we present individual service implementations and discuss their orchestration. To wrap up, we also introduce an evaluation framework for the CBR services that demonstrates the client side of our architecture.

**Argumentation Base** The argumentation base is provided by our ArgueBuf Python library (see Section 6 for more details). It can serve argument graphs from a local directory or a remote server and allows to filter them with regular expressions. Our implementation expects that filter criteria are stored in directory names using the pattern `<property1>=<value1>,<property2>=<value2>,...` and therefore allows the use of arbitrary properties for filtering.

To get started more easily, we provide a public collection of argument graphs called ArgueBase[8] that adheres to the naming convention mentioned of the directory. It contains a diverse set of publicly argument graph corpora in various formats like AIF or ArgueBuf and includes links to the original sources and licenses.

**Natural Language Processing** Our NLP service implementation is written in Python and built on the popular spaCy library—including a specialized client to simplify the consumption of the service. Besides the embedding models offered by spaCy, our services provide an integration with SentenceTransformers[9] that allows to use a wide range of pre-trained models for computing contextualized embeddings and includes support for CUDA acceleration. For regular word embeddings, multiple pooling methods are supported in addition to the default

[7] `https://github.com/recap-utr/arg-services-poc`
[8] `https://github.com/recap-utr/arguebase`
[9] `https://www.sbert.net/`

pooling of the mean, including the generalized power mean [35]. Instead of applying cosine similarity to pooled vectors, max-pooling can be used to determine the similarity between two texts [42]. Finally, multiple models can be selected at the same time with their individual embeddings concatenated to a larger vector.

**Argument Mining** Based on a prompting strategy created for another project (currently in development), we implemented an LLM-based AM service in Python. The service allows to run the stages individually or as an end-to-end pipeline and makes use of the *function calling* feature of OpenAI's ChatGPT to enforce a JSON schema for the predictions. It demonstrates that even in light of generative models getting better at many tasks, our architecture can act as a *translation layer* between new and existing systems.

**Case-Based Reasoning on Arguments** The retrieval functionality is implemented in the Python application ArgueQuery and uses both semantic and structural similarity measures for CBR with argument graphs. The semantic part is handled by comparing embeddings, while the structural part involves an A* search algorithm to find the best mapping between the user query and the graphs in the case base.

Adaptation is possible through ArgueGen and uses a combination of WordNet [24,1] and LLMs to adapt retrieved arguments. For each argument to be adapted, the Python-based service first identifies the central keywords, prompts a generative language model for suitable replacements, verifies the response using the WordNet database, and applies all valid ones to the argument graph. In addition to this hybrid approach, it is also possible to perform the adaptation solely based on LLMs or WordNet.

Both of these services make use of the NLP service of our architecture to compute the embeddings and extract other linguistic features from the texts. They show the power of the NlpConfig message: Each service receives an NLP configuration object containing parameters like the model to use, and then uses it themselves to perform requests to the NLP service. In this way, it is possible to share a single object between all services but also use different configurations for certain services if needed.

**Argument Retrieval and Ranking** For the two ranking functions, a Java-based application built on Apache Lucene[10] is available. Given a user-provided textual query, this argument search engine returns a ranked list of representatives from premise clusters. It first searches an inverted index for claims that are similar to the query, identifies all linked premises (pre-clustered according to their semantics), and then ranks these clusters at runtime using frequencies or argument quality. For the fine-granular clustering function, we developed a prompt-based strategy leveraging OpenAI's ChatGPT to provide responses (similar to the argument mining service).

[10] `https://lucene.apache.org/`

**Quality Assessment of Arguments** The same LLM-based approach was used to provide a prototype of our quality explanation function. To determine the qualia annotations, a text and a list of patterns consisting of sequences of POS tags are expected. The Java-based system then creates the constituency trees of the text and searches for the patterns. If these are found, the qualia role and the qualia query that match a pattern are output for each match. The expert suitability function is the only functionality of our proposed architecture that is not part of this proof-of-concept.

**Orchestration of Services** The ultimate goal of our architecture is to provide an integrated argumentation machine that exposes a set of services to the user. To simplify the deployment and orchestration of these services, we provide Docker-based containers for many of the services described in this section. These containers can be managed jointly using Docker Compose, for which we provide a configuration template as part of our proof-of-concept implementation. For our two central services—that is, the argumentation base and NLP service—we also provide pre-built images that can be pulled directly from the GitHub Container Registry. For all argument mining and CBR services, we created ready-to-use Docker files that can be built locally and integrated into the Docker Compose configuration. The services for ranking and quality assessment of arguments (see Sections 4.5 and 4.6) are mostly written in Java and require custom binary files, so their deployment is more involved. We plan to provide Docker images for the in the future as well.

**Evaluation Client** To evaluate our CBR services, we created a client called ArgueLauncher[11] that can be used to compare the results of the argumentation machine to a gold standard. It is written in Python, can be used to evaluate the retrieval and adaptation services, and contains an abstract interface for evaluations that can be easily extended to other services in the future. With this application using only the client libraries of the services, it can also be used by other developers as a starting point for integrating our architecture into their own systems.

## 6 Supplementary Resources

Besides the service descriptions and their accompanying Protobuf definitions, we also provide a few additional tools and resources to simplify the development of argumentation machines. When designing these, we strived to follow the best practices of software engineering. For instance, all libraries strictly adhere to the semantic versioning scheme and provide a changelog for each release. We also make extensive use of the package manager *Nix*[12] to manage our dependencies and provide reproducible environments. New releases are published through a

[11] `https://github.com/recap-utr/arguelauncher`
[12] `https://nixos.org`

Continuous Integration (CI) pipeline that leverages our Nix setup in GitHub actions. In the next section, we highlight what we believe are the most useful resources for other researchers and developers in the domain of CA.

**Ready-to-Use Client and Server Libraries** As stated in Section 2.3, Protobuf offers easy code generation of native libraries for most programming languages, but this additional step may already be a hurdle for some developers. To lower the barrier of entry, we provide ready-to-use client and server libraries for Python, TypeScript/JavaScript, and Java.[13] They can be installed via their native package manager—that is, `pip`, `npm`, and `maven`.

Creating a new argument graph format for our microservices allows first-class support without the need for numerous format conversions. However, it also means that existing and commonly used formats like AIF cannot be used out-of-the-box. To remedy this, we provide *supercharged* libraries for Python and JavaScript/TypeScript that make it easy to import graphs in AIF, Argdown, Kialo, OVA3, SADFace, and xAIF and export them to AIF and xAIF.[14] They also contain optimized graph representations that abstract some Protobuf-specific details away and make it easier to work with argument graphs in these languages. The Python version is additionally integrated with NetworkX and can render images of graphs using D2 and Graphviz.

**REST API Gateway** Even though gRPC provides major advantages over REST and we try to reduce the burden of using it as much as possible, it is still not as widely used as REST. It may also be the case that a developer wants to integrate our services into an existing system that already uses REST APIs. To combine the best of both worlds, we created a proxy[15] that allows REST clients to access any gRPC service. It is based on the popular Envoy proxy[16] and is provided as a Docker image and a binary file for all three major operating systems: Windows, Linux, and macOS. Additionally, it also supports the conversion between gRPC-Web requests and regular gRPC requests (which is needed for browser-based clients).

**Argument Mapping Interface** First introduced at COMMA 2022 [21], our tool ArgueMapper [21] is a web-based interface for creating argument graphs.[17] Compared to established solutions such as OVA, it uses a modern development stack (TypeScript and React) and is thus easier to extend and maintain. ArgueMapper natively support our Protobuf-based serialization format Arguebuf and can be used to build the argumentation base for our microservices.

[13] `https://github.com/recap-utr/arg-services`
[14] `https://github.com/recap-utr/arguebuf`
[15] `https://github.com/mirkolenz/grpc-proxy`
[16] `https://www.envoyproxy.io/`
[17] `https://github.com/recap-utr/arguemapper`

## 7 Limitations

While we have tried to create an extensible architecture that can be used for a wide range of tasks in CA, there are still some limitations to our approach. First, the reliance on gRPC and Protobuf may be a hurdle for some developers: Although they provide a strong contract between the client and the server, they are not as widely used as REST—especially in the context of research projects. We try in part to mitigate this through our REST API gateway, but it is still an additional layer that needs to be managed.

The representation of arguments through graphs is backed deeply into our architecture and may not be suitable for all argumentative domains. For example, some texts may contain arguments that are only loosely connected or where the relations between them are implicit—like in news editorials. In such cases, the strict graph structure may not be the best choice for representing arguments. In addition, our ArgueBuf format does not yet support dialogical argumentation, limiting its applicability in this domain.

Lastly, the evaluation of our architecture is still ongoing, and we have not yet tested it in a real-world scenario. We assume that there exist some features that are missing and/or incomplete when coming to new domains, but we are open for feedback and contributions from the community to improve our architecture—which is possible due to the forward- and backward-compatibility of Protobuf. An example of such missing features is our NLP service that is currently focused only on the extraction of linguistic features and the computation of semantic similarity. In the future, it may be desirable to extend its scope and integrate functions to serve custom models to other services.

## 8 Conclusion & Future Work

In this paper, we presented an extensible microservice-based architecture for argumentation machines based on gRPC and Protobuf. We also presented a proof-of-concept implementation of our architecture and a ready-to-use evaluation framework for CBR tasks. Finally, we introduced a set of supplementary resources that we believe are useful to other researchers and developers in the domain of CA. The architecture is the culmination of our work over the past years, and we hope to contribute to the standardization of argumentation machines. We also await feedback from the community to improve our architecture and resources—everything is hosted on GitHub and open to any kind of contribution.

Software is never a finished product, so there are many potential avenues for future work. First, we will implement the remaining services mentioned in Section 4 so that the evaluation framework is no longer restricted to CBR. Subsequently, we will add more services to our architecture to support more tasks in CA. To appeal to a wider audience, we plan to develop an intuitive UI that allows both lay persons and experts to use these services. Another starting point for future work is the creation of a low-code solution for defining new services on the fly—enabling the fast adoption of new ideas and trends in the field CA.

**Acknowledgements** This work has been funded by the Deutsche Forschungsgemeinschaft (DFG) within the projects *ReCAP* and *ReCAP-II* (№ 375342983, 2018–2024) as part of the priority program *RATIO* (Robust Argumentation Machines, SPP-1999) as well as the *Studienstiftung*.

## References

1. WordNet: An Electronic Lexical Database (1998)
2. Aamodt, A., Plaza, E.: Case-Based Reasoning - Foundational Issues, Methodological Variations, and System Approaches. AI Commun. (1994)
3. Ajjour, Y., Wachsmuth, H., Kiesel, J., Potthast, M., Hagen, M., Stein, B.: Data Acquisition for Argument Search: The args.me Corpus. In: KI 2019: Advances in Artificial Intelligence. pp. 48–59 (2019)
4. Al-Debagy, O., Martinek, P.: A Comparative Review of Microservices and Monolithic Architectures. In: 2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI). pp. 000149–000154 (2018)
5. Allen, J.F.: Natural language processing. In: Encyclopedia of Computer Science, pp. 1218–1222 (2003)
6. Bergmann, R., Biertz, M., Dumani, L., Lenz, M., Ludwig, A.K., Neumann, P.J., Ollinger, S., Sahitaj, P., Schenkel, R., Witry, A.: The ReCAP Project. Datenbank Spektrum pp. 93–98 (2020)
7. Bergmann, R., Lenz, M., Ollinger, S., Pfister, M.: Similarity Measures for Case-Based Retrieval of Natural Language Argument Graphs in Argumentation Machines. In: Proceedings of the Thirty-Second International Florida Artificial Intelligence Research Society Conference. pp. 329–334 (2019-05-19/2019-05-22)
8. Bex, F., Lawrence, J., Snaith, M., Reed, C.: Implementing the argument web. Commun. ACM pp. 66–73 (2013)
9. Bex, F., Snaith, M., Lawrence, J., Reed, C.: ArguBlogging: An application for the Argument Web. Journal of Web Semantics pp. 9–15 (2014)
10. Biertz, M., Dumani, L., Nilles, M., Metzler, B., Schenkel, R.: QualiAssistant: Extracting Qualia Structures from Texts. In: Proceedings of the 9th Workshop on Argument Mining. pp. 199–208 (2022)
11. Bondarenko, A., Fröbe, M., Kiesel, J., Schlatt, F., Barriere, V., Ravenet, B., Hemamou, L., Luck, S., Reimer, J.H., Stein, B., et al.: Overview of Touché 2023: Argument and Causal Retrieval. In: Advances in Information Retrieval. pp. 527–535 (2023)
12. Britner, S., Dumani, L., Schenkel, R.: AQUAPLANE: The Argument Quality Explainer App. In: Proceedings of the 32nd ACM International Conference on Information and Knowledge Management. pp. 5015–5020 (2023)
13. 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)
14. Dumani, L., Wiesenfeldt, T., Schenkel, R.: Fine and Coarse Granular Argument Classification before Clustering. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management. pp. 422–432 (2021)
15. Eden, L., Kantor, Y., Orbach, M., Katz, Y., Slonim, N., Bar-Haim, R.: Welcome to the Real World: Efficient, Incremental and Scalable Key Point Analysis. In: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track. pp. 483–491 (2023)

16. Fröbe, M., Wiegmann, M., Kolyada, N., Grahm, B., Elstner, T., Loebe, F., Hagen, M., Stein, B., Potthast, M.: Continuous Integration for Reproducible Shared Tasks with TIRA.io. In: Advances in Information Retrieval. pp. 236–241 (2023)
17. Heinisch, P., Orlikowski, M., Romberg, J., Cimiano, P.: Architectural Sweet Spots for Modeling Human Label Variation by the Example of Argument Quality: It's Best to Relate Perspectives! In: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. pp. 11138–11154 (2023)
18. Jamshidi, P., Pahl, C., Mendonça, N.C., Lewis, J., Tilkov, S.: Microservices: The Journey So Far and Challenges Ahead. IEEE Software pp. 24–35 (2018)
19. Janier, M., Lawrence, J., Reed, C.: OVA+: An Argument Analysis Interface. In: Computational Models of Argument - Proceedings of COMMA 2014, Atholl Palace Hotel, Scottish Highlands, UK, September 9-12, 2014. pp. 463–464 (2014)
20. Lawrence, J., Reed, C.: Argument Mining: A Survey. Computational Linguistics pp. 765–818 (2019)
21. Lenz, M., Bergmann, R.: User-Centric Argument Mining with ArgueMapper and Arguebuf. In: Computational Models of Argument. pp. 367–368 (2022)
22. Lenz, M., Bergmann, R.: Case-Based Adaptation of Argument Graphs with WordNet and Large Language Models. In: Case-Based Reasoning Research and Development. pp. 263–278 (2023)
23. 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 (2020)
24. Miller, G.A., Beckwith, R., Fellbaum, C., Gross, D., Miller, K.: WordNet: An online lexical database. International Journal of Lexicography pp. 235–244 (1990)
25. Montani, I., Honnibal, M., Boyd, A., Landeghem, S.V., Peters, H., McCann, P.O., geovedi, j., O'Regan, J., Samsonov, M., de Kok, D., et al.: spaCy: Industrial-strength Natural Language Processing (NLP) in Python. Zenodo (2023)
26. Nilles, M., Dumani, L., Metzler, B., Schenkel, R.: Trust me, I am an Expert: Predicting the Credibility of Experts for Statements. In: Proceedings of the Workshops at the 31st International Conference on Case-Based Reasoning (ICCBR-WS 2023). pp. 114–128 (2023)
27. Nilles, M., Dumani, L., Schenkel, R.: QuARk: A GUI for Quality-Aware Ranking of Arguments. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 2546–2549 (2021)
28. Peldszus, A., Stede, M.: From Argument Diagrams to Argumentation Mining in Texts - A Survey. IJCINI pp. 1–31 (2013)
29. Reed, C., Long, D., Fox, M.: An architecture for argumentative dialogue planning. In: Practical Reasoning. pp. 555–566 (1996)
30. Reed, C., Norman, T.J.: A Roadmap of Research in Argument and Computation. In: Argumentation Machines: New Frontiers in Argument and Computation, pp. 1–13 (2004)
31. Reed, C., Walton, D.: Applications of Argumentation Schemes. OSSA Conference Archive (2001)
32. Reed, C., Wells, S., Devereux, J., Rowe, G.: AIF+: Dialogue in the argument interchange format. In: Computational Models of Argument: Proceedings of COMMA 2008. pp. 311–323 (2008)
33. Richter, M.M., Weber, R.: Case-Based Reasoning: A Textbook (2013)
34. Romberg, J.: Is Your Perspective Also My Perspective? Enriching Prediction with Subjectivity. In: Proceedings of the 9th Workshop on Argument Mining. pp. 115–125 (2022)

35. Rücklé, A., Eger, S., Peyrard, M., Gurevych, I.: Concatenated Power Mean Word Embeddings as Universal Cross-Lingual Sentence Representations. arXiv:1803.01400 [cs] (2018)
36. Slonim, N., Bilu, Y., Alzate, C., Bar-Haim, R., Bogin, B., Bonin, F., Choshen, L., Cohen-Karlik, E., Dankin, L., Edelstein, L., et al.: An autonomous debating system. Nature pp. 379–384 (2021)
37. Stab, C., Daxenberger, J., Stahlhut, C., Miller, T., Schiller, B., Tauchmann, C., Eger, S., Gurevych, I.: ArgumenText: Searching for Arguments in Heterogeneous Sources. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations. pp. 21–25 (2018)
38. Wachsmuth, H., Naderi, N., Habernal, I., Hou, Y., Hirst, G., Gurevych, I., Stein, B.: Argumentation Quality Assessment: Theory vs. Practice. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). pp. 250–255 (2017)
39. Wachsmuth, H., Potthast, M., Al Khatib, K., Ajjour, Y., Puschmann, J., Qu, J., Dorsch, J., Morari, V., Bevendorff, J., Stein, B.: Building an Argument Search Engine for the Web. In: Proceedings of the 4th Workshop on Argument Mining. pp. 49–59 (2017)
40. Walton, D., Macagno, F.: A classification system for argumentation schemes. Argument & Computation pp. 219–245 (2016)
41. Walton, D., Reed, C., Macagno, F.: Argumentation Schemes (2008)
42. Zhelezniak, V., Savkov, A., Shen, A., Moramarco, F., Flann, J., Hammerla, N.Y.: Don't Settle for Average, Go for the Max: Fuzzy Sets and Max-Pooled Word Vectors. arXiv:1904.13264 [cs] (2019)