# Highlights

**Explainable Argumentation as a Service**

Nikolaos I. Spanoudakis, Georgios Gligoris, Antonis C. Kakas, Adamos Koumi

- Argumentation-based explainable decision making is provided as a service
- Developers can author, test and use their theories on-line
- Developers are assisted in developing their theories with rule templates
- Multiple scenarios may be used for assisting/automating the testing phase
- The system supports university courses labs and students thesis in three countries

# Explainable Argumentation as a Service

Nikolaos I. Spanoudakis[a,*], Georgios Gligoris[b], Antonis C. Kakas[c], Adamos Koumi[c]

[a]*School of Production Engineering and Management,*
[b]*School of Electrical and Computer Engineering,*
*Technical University of Crete, University Campus, Chania, 73100, Greece*
[c]*Department of Computer Science,*
*University of Cyprus, 75 Kallipoleos Str., P.O. Box 537, Nicosia, CY-1678, Cyprus*

**Abstract**

Gorgias Cloud offers an integrated application development environment that facilitates the development of argumentation-based systems over the internet. Argumentation is offered as a service in a way that this allows application systems to remotely access the argumentation service and utilize the results of the argumentative computation. Moreover, the service results include the explanation of the decision in both human and machine-readable formats. The first is useful for allowing the application validation to be done by experts, while the second is useful for development. It appears that this is the first case where argumentation is offered to developers in such an open and distributed way.

*Keywords:* Argumentation, SaaS, Explainable AI
*2000 MSC:* 68T35, 68U35, 68N19

## 1. Introduction

Argumentation is an emerging technology that has matured enough to produce practical applications for the industry. Start-up companies, like Argument Theory, already have clients using argumentation for decision making, drawing upon expert knowledge.

Although there is a number of frameworks [1] for developing argumentation systems, e.g., Gorgias [2], CaSAPI [3], DeLP [4], and $ASPIC^+$ (TOAST system) [5], web-based argumentation systems upon which applications can be built are rare.

A web-based system, SPINdle [6], allows for web-based development and testing of defeasible logic applications. Its environment, however, consists simply of a text editor for writing logic programming rules. Another recent editor, *Web-Gorgias-B* [7], allows to build argumentation theories on-line based on the Gorgias framework.

Gorgias is a structured argumentation framework where arguments are constructed using a basic (content independent) scheme of *argument rules*. Two types of arguments are constructed within a Gorgias argumentation theory: *object-level* arguments and *priority arguments* expressing a preference, or relative strength, between other arguments. The dialectic argumentation process of Gorgias to determine the acceptability/admissibility of an argument supporting a desirable claim typically occurs between *composite arguments* where priority arguments are included into the composite argument in order to strengthen the arguments currently committed to.

The Gorgias framework was introduced in [8], extended in [2] and has been applied to a variety of real-life application problems [9]. The system of Gorgias was introduced in 2003 and has been used by several research groups to develop prototype real-life applications of argumentation in a variety of application domains, e.g. in portfolio management [10], provision of services in ambient intelligence [11], medical diagnosis [12], product pricing [13], management of firewall policies [14], conflicts resolution in pervasive services [15], [16]).

*Web-Gorgias-B* eliminates the need for logic programming knowledge for application developers, thus, allowing naive users to, for example, define their decision policies as argumentation theories. *Web-Gorgias-B* achieves this task by using a table formalism recently proposed in the literature [9]. According to this, a user can define application scenarios and select the available options in each scenario. However, this approach poses a number of limitations to Gorgias developers preventing, e.g., the use of abducibles, or arguing about beliefs.

In this paper, we propose a web-based Integrated Development Environment (IDE) for applications of argumentation, named Gorgias Cloud, offering an argumentation-based reasoning Application Programming Interface (API), thus allowing us to build systems that use argumentation *as a service* in the context decision making applications. Moreover, this API provides an explanation capability that enhances the utility of the service in the application systems.

The Gorgias Cloud IDE offers three novel features:

1. Assistance for editing argumentation theories in the internal code language of Gorgias using templates for

*Corresponding Author

object level or priority arguments and abducibles
2. Ability to store multiple scenarios to test the behaviour of developed theories
3. REST-compliant web API so that *Gorgias* queries can be executed from any other programming environments (e.g. Java, Python) used for developing applications.

In the following section of materials and methods, we review some background information on the Gorgias argumentation framework along with the main concepts of explainable artificial intelligence. We also present some of the main recent web development techniques that we will be combining with argumentation to build systems. Then, in section 3, we outline the *Gorgias* Cloud system's architecture, and present the main features of the system, i.e. the authoring environment, the explainable output and the API with its usage for building web applications. We discuss how such applications are build through a simplified example of a social media assistant. Finally, we focus on the system evaluation and discussion of our findings before concluding.

## 2. Materials and Methods

### *2.1. The Gorgias Argumentation Framework*

Gorgias is a structured argumentation framework where arguments are constructed using a basic (content independent) argument scheme that associates a set of premises with the claim, or position, of the argument. In this framework we can represent **argument rules**, denoted by **Premises ▷ Claim** linking the Premises, a set of literals, to a literal Claim: we say that the argument rule **supports** the Claim.

The Premises are typically given by a set of conditions describing a scenario. There are cases, however, where the conditions are themselves defeasible. We call them **beliefs**, and they can be argued for or against, i.e. the claim of an argument can also be a literal on a belief predicate. The distinction, then, between beliefs and options, is that options cannot be premises of arguments whose Claim is a belief.

An **argument**, $A$, is a set of **argument rules**. In an argument, $A$, through the successive application of its constituent argument rules several claims including a "final or desired" claim are reached and, hence, supported by $A$. Argument rules have the syntax of Extended Logic Programming, i.e. rules whose conditions and conclusion (claim) are positive or explicit negative atomic statements, but where negation as failure is excluded from the language[1]. The conclusion of an argument rule can be a positive or negative atomic statement. The conflict relation in Gorgias can be expressed either through explicit negation or a complementarity relation between statements in the application language or an explicit statement of conflict between argument rules and any combination of these three ways.

An important element of the Gorgias framework is that it allows a special class of argument rules, namely **priority argument rules** that are used to express a context-sensitive relative strength between (other) argument rules. They have the same syntactic form as arguments rules, i.e. they have the form Premises ▷ Claim, but now the Claim is of a special type, $a_1 > a_2$, where $a_1$ and $a_2$ are (the names of) any two other individual argument rules. Note that $a_1$ and $a_2$ can themselves be priority argument rules, in which case we say that the argument is a **higher-order** priority argument expressing the fact that we prefer one priority over another, in the context of the premises of this higher-order priority. When the claim of an argument is not a priority statement, i.e. it is a literal on some relation in the language, this argument is called an **object-level** argument.

The purpose of **priority arguments**, constructed from priority argument rules, is to represent a relative strength between arguments and thus to provide the defense relation between arguments and sets of arguments. Informally, an argument is allowed to defend against another argument only if it is at least as strong as the argument that it defends against. Since the arguments in the framework are sets of argument rules and the priority relation is defined on the elements of these sets we need to lift this relation onto the sets in order to get a strength relation $\succ$ on the arguments. Furthermore, the dialectic argumentation process to determine the acceptability/admissibility of an argument supporting a desirable claim typically occurs between composite arguments, containing both object and priority argument rules. **Composite arguments** are (minimal and closed) set of arguments, $\Delta = (A_1, A_P)$, where, $A_1$, is a subset of object level argument rules and $A_P$ is a subset of priority argument rules, chosen from the given argumentation theory in an application problem. Then the $\succ$ (and hence defense) relation between two composite arguments is induced from the local strength relation that the priority rules, contained in the respective composite arguments, give to the other arguments contained in them. Informally, a composite argument, $\Delta_1$, defends another composite argument, $\Delta_2$, whenever they are in conflict, and the arguments in $\Delta_1$ are rendered by the priority arguments that it contains at least as strong as the arguments contained in $\Delta_2$. In other words, if the priority arguments in $\Delta_2$ render an argument in it preferred over an individual argument in $\Delta_1$ then so do the priority arguments in $\Delta_1$: a relative weak argument in $\Delta_1$ is balanced by a weak argument in $\Delta_2$.

### *2.2. Explainable AI*

Providing explanations for the results of AI systems is today a major requirement for any AI system as we require that systems need not only to perform well in the accuracy of their output but also in the interpretability

[1] Initially, the framework of Gorgias had the name $LPwNF$ : Logic Programming without Negation as failure.

and understandability of their results (see e.g. [17, 18] for recent surveys on **Explainable AI**). Explanations of conclusions drawn from a learned theory or decisions taken by an AI system facilitate their usability by other systems (artificial or human) and contribute significantly towards building a high level of trust towards information systems. They have a role to play both at the level of developing a system and at the level of acceptance of the system in its application environment. The central idea is that explanations can make our systems interpretable and thus usable, by formulating, for the developer and or final user, a "story" line of the production of the output from the input data, using appropriate human-understandable features. Explanations can, thus, help render our systems transparent, accountable and contestable.

The central challenge facing Explainable AI is in the generation of explanations that are both interpretable and complete (i.e. true to the computational model underlying the system). Depending on the complexity of the system this trade-off between precision and interpretability of explanation becomes harder. Nevertheless, there are several properties of good quality explanations that we can follow, stemming mainly from the need for explanations to be cognitively compatible with the users and to be "socially useful" in clarifying the reasons underlying the results of the system that they are explaining [19]. Explanations need to be sensitive to the level of the user and to the purpose that the user needs the result of the system that is explained.

In general, there are three main cognitive and social requirements that form a good quality explanation. An explanation must be **attributive**, i.e. must give the basic and important reasons justifying why the result that it is explaining holds and could be chosen. Equally important is for the explanation to explain why the particular result is a good choice in relation to other possible results, i.e. the explanation must also be **contrastive**. Finally, a good explanation is also one that is **actionable**, i.e. where it is appropriate, it helps us understand what further actions we can take to confirm and to further build or utilize the result.

Systems build based on argumentation naturally lend themselves to being explainable due to the very close and direct link between argumentation and justification. Arguments supporting a claim or conclusion can provide the attributive part of an explanation while the defending arguments, within an acceptable coalition of arguments resulting from the dialectic argumentation process to defend the attributive arguments against their counter-arguments, will provide the contrastive element of the explanation. These attributive and defending arguments also point towards taking (further) actions to confirm or question their premises, particularly when these relate to subjective beliefs or hypotheses.

### *2.3. Web Development*

A modern approach to front-end development for web applications is the Component Based Architecture [20] (CBA), an evolution of the Model-View-Controller [21] (MVC) design pattern.

CBA relies on the Object-Oriented Model. The web application is comprised of components that contain only elements with related functionality and that, following the OOD paradigm are encapsulated and loosely coupled. Each component exposes an API without the need to reveal its internal components or state. Thus, this approach has the following properties:

- **reusability**, a component may be used in multiple places in the application
- **testability**, allowing for testing components independently using their API
- **readability**, the code is easy to read and, thus, maintain

Moreover, and because web development is not only about designing user interfaces (the front-end), the web application needs a back-end and a database.

The Spring Boot [22] framework is a technology for developing a back-end based on the Java programming language and a variety of technologies built on it for business applications development. Examples are the Java beans for encapsulating the business logic or data items and the Hibernate [23] framework for automating relational database generation and communication. For the latter, a cross-platform solution is the MySQL relational database, one of the most popular databases.

Finally, the modern approach to servers development calls for using virtual machines for executing server software, allowing for better load-balancing and for dynamic resources allocation to servers. Docker [24] is an open platform that allows us to separate applications from infrastructure, but also to manage the infrastructure similarly to managing applications. Docker manages multiple virtual machines allowing to specify if and how they are connected.

## 3. The Gorgias Cloud System

In this section we present the Gorgias Cloud system. Specifically, in section 3.1 we provide an overview of the system architecture. Subsequently, in section 3.2, we focus on describing the Gorgias Cloud website as a web-based Integrated Development Environment (IDE), using an example to illustrate it. Finally, we focus on two important and original features of the IDE, i.e., the application level explanation component (section 3.3) and the Application Programming Interface (API, section 3.5) for use by application developers.

### *3.1. System Architecture*

The overall IDE was designed following the modern Component Based Architecture [20] (CBA). The client-side application employs technologies that can run on any standard browser. We used HTML5 and CSS3.

The server side includes the front-end, the back-end and the database in a three layered architecture. For the front-end we used Angular and for the back-end the Spring Boot [22] framework.

The front-end is the website of Gorgias Cloud and allows the user to register for an account, log-in and use the Integrated Development Environment (IDE) for argumentation based decision making, by developing theories to execute using the Gorgias system.

The back-end provides the business logic of Gorgias Cloud through services that support the front-end. Some of these services are also available in the internet as an API, so that a user application can access the Gorgias Cloud for managing a user's projects and files but also for argumentation-based decision making. The API provides the following services:

- **createProject**, to create a new project for a user
- **deleteProject**, to delete a project of a user
- **getUserProjects**, to get a list of the projects of a user
- **getProjectFiles**, to get a list of the available files in a user's project
- **addFile**, to add (upload) a new file inside a user's project
- **deleteFile**, to delete a file inside a user's project
- **getFileContent**, to get the content (text) of a file in a user's project
- **setFileType**, to set the type of a file in a user's project. The files can have one of the following types:
  - **Gorgias File**, storing a decision theory
  - **Scenario File**, the scenario files instantiate facts and beliefs for testing decision theories
  - **Background File**, containing background information for the application domain in Prolog syntax
- **updateFile**, to update (replace) a file in a user's project
- **GorgiasQuery**, to determine if a decision is supported by an admissible argument of the **Gorgias** theory (file) in a user's project. This is the central service of the API.Its input is the user's project and file, along with a query predicate and a list of beliefs/facts that (partially) constitutes the current context of the application. The output consists of whether this query is admissible or not. If it is, the output contains also the query predicate's instantiated variables and an explanation for its admissibility both in machine and human readable formats.

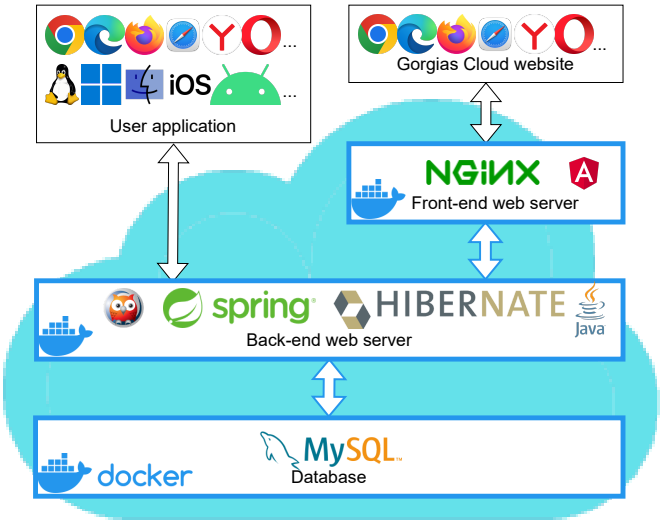


Figure 1: Gorgias Cloud System Architecture Overview

The back-end, through the use of Data Object Java Beans (DAO), connects to the MySQL database where the following information is stored:

- **User Data**, the users' access credentials. Each user is also linked to zero or more:
- **Projects**, that each is linked to zero or more:
- **Files**, each file has a content (code) and type (see the file types above)

The **Prolog Service** is implemented by the Spring-Boot programming framework. To establish a connection between Prolog and Spring Boot we use the Java-Prolog Interface (JPL) library [25].

Figure 1 outlines the Gorgias Cloud system architecture. On the top-right we can see the Gorgias Cloud website that the user can access using any web browser. The website is offered by the Front-end web server (an instance of NGINX) that executes server-side javascript written in the Angular framework. The Front-end web server is a docker container. The back-end (also a docker container) is used by the Front-end but also by user applications, be they web sites using any kind of technology (e.g. PHP, javascript, etc), or apps running on smart devices (android or iOS) or desktop operating systems (Linux, MAC-OS, or, Windows).

### *3.2. Authoring environment*

We will now showcase the main features of the Gorgias Cloud system using an application example. The example is about developing a social media assistant that will help its user to see posts, generated by the user's social network, in an adapted form according to the user's wishes (or policy). The idea is for the personalized assistant to help the user manage the information overload that can occur within the social media platforms.

Let us consider a particular user whose policy is the following:

> Normally, give **default** priority to a post. If the topic of a post falls within the user's interests set the priority to **important**, unless the information of the post is negative. Posts that come from the user's manager are **important** regardless of whether they are positive or negative. **Hide**, posts that are on politics unless the post is from the user's manager. **Hide** politics posts from the user's manager when negative, but when positive they are set to be **important**.

After the user logs in the Gorgias Cloud website the user can define a new project and start editing a Gorgias file. The user may also upload a file. The Gorgias Cloud editor, however, can aid the user in writing the theory by defining the following five templates that can help the user write down the theory:

- **Argument Rule**:
  *rule*(*Argument_Name*, *Position*, [*Defeasible_Premises*]):– *Non_Defeasible_Premises*.
  where:
  - *Argument_Name* is a prolog predicate used as the rule's label
  - *Position*, is a prolog predicate representing a Claim. Note that in the rest of the paper a Claim can also be referred to as a Position
  - *Defeasible_Premises* is a comma-separated list of prolog predicates that are beliefs and premises to the Position
  - *Non_Defeasible_Premises* is a comma-separated list of prolog predicates that are non-defeasible premises to the Position
- **Preference Rule**:
  *rule*(*Preference_Name*, *prefer*(*Label_1*, *Label_2*), [*Defeasible_Premises*]):–*Non_Defeasible_Premises*.
  where:
  - *Preference_Name* is a prolog predicate used as the rule's label
  - *Label_1* is a prolog predicate representing a rule's label. The rule with this label is preferred over the rule with *Label_2*
  - *Label_2* is a prolog predicate representing the non-preferred rule's label *Defeasible_Premises* is a comma-separated list of prolog predicates that are beliefs and premises to the *prefer* relation
  - *Non_Defeasible_Premises* is a comma-separated list of prolog predicates that are non-defeasible premises to the *prefer* relation
- **Complement**:
  *complement*(*Option_1*, *Option_2*):–
  *Non_Defeasible_Premises*.
  where:
  - *Option_1* is a prolog predicate representing a Position not compatible with Position *Option_2*. This means that arguments supporting (whose Position is) *Option_1* **attack** any arguments supporting *Option_2*
  - *Option_2* is a prolog predicate representing a Position
  - *Non_Defeasible_Premises* is a comma-separated list of prolog predicates that are non-defeasible premises to the validity of the complementarity relation
- **Conflict**:
  *conflict*(*Label_1*, *Label_2*).
  where:
  - The rule with label *Label_1* attacks the rule with *Label_2* and vice-versa
- **Abducible**:
  *abducible*(*Name*, [*Defeasible_Premises*]):–
  *Non_Defeasible_Premises*.
  where:
  - *Name*, is a prolog predicate representing a belief
  - *Defeasible_Premises* is a comma-separated list of prolog predicates that are beliefs and premises to the validity of the abducible
  - *Non_Defeasible_Premises* is a comma-separated list of prolog predicates that are non-defeasible premises to the validity of the abducible

Using the above five templates, the user can define a decision theory in the Gorgias framework. Figure 2 shows the code editor of the Gorgias Cloud IDE. Whenever saving or uploading a file the user is warned if there are syntax errors or warnings as if using the SWI-Prolog environment.

In Listing 1 we can see the Gorgias code encoding the above personal policy of our social media assistant example. Lines 1-6 define the three positions or options (*default*(*Post*), *hide*(*Post*) and *important*(*Post*)) as mutually exclusive using the *complement* template.

The **Argument Rules** in lines 7-9 state that all positions are possible for any given *Post*. Then lines 10-25 define the **Preference Rules** for the user's policy. Specifically, lines 10-11 give default preference to rule with *Argument_Name* : *r1*(*Post*), i.e. the default priority for any given *Post*. If the post is in the user's topics of interest then the priority is important (this is achieved by Preference Rules 12-14), unless the information of the post is negative (this is achieved in lines 15-16). If the post comes

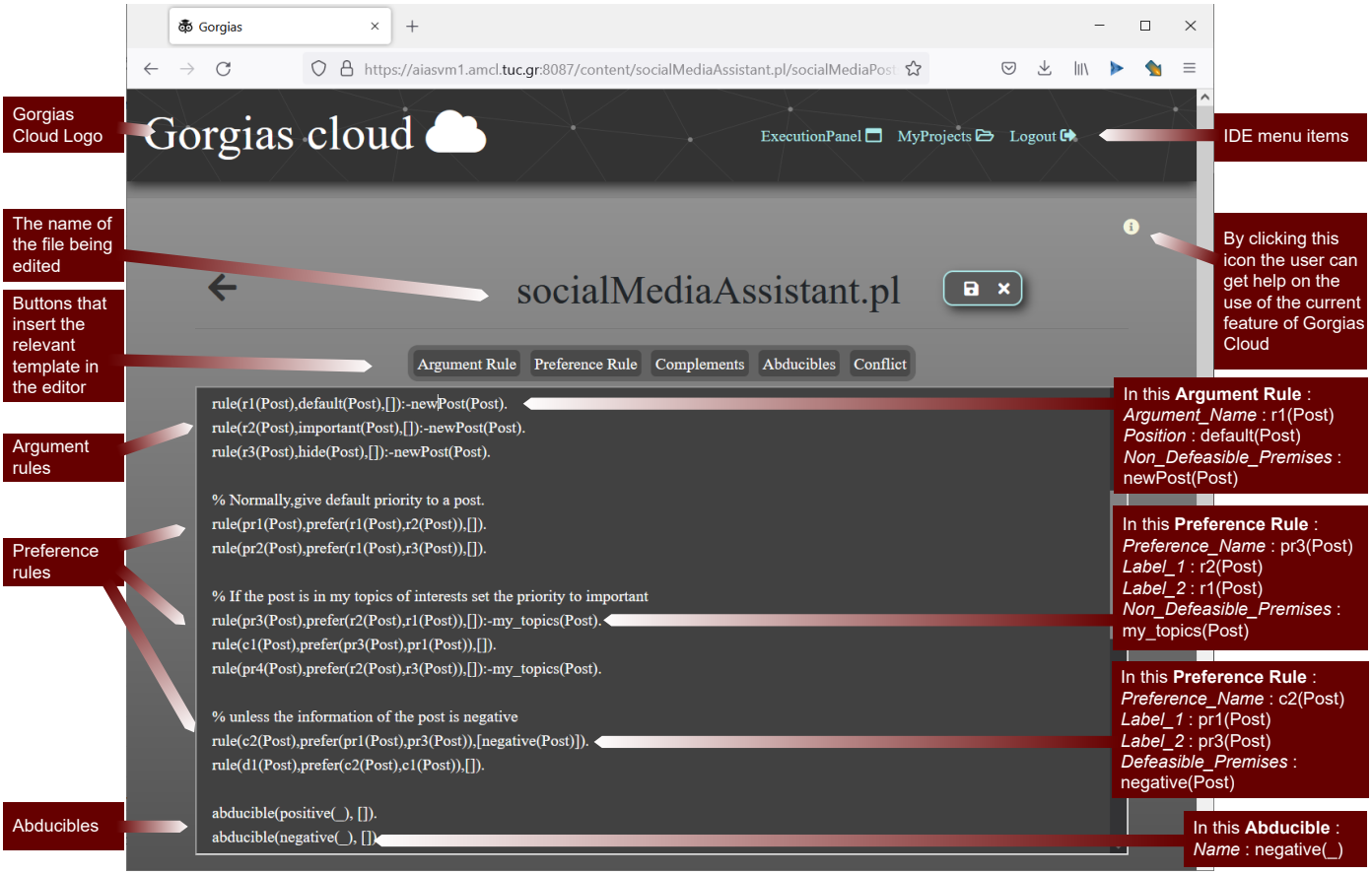


Figure 2: Authoring the Gorgias theory

from the user's manager the priority is set to important, regardless if it is positive or negative (lines 17-19). Posts that are on politics are hidden (lines 20-23), unless the post is from the user's manager. In that case the post must be hidden, if it is negative, or marked as important, if it is positive (lines 24-25). Finally, each post can be assumed to be positive or negative if its status cannot be determined (note the **Abducibles** in lines 26-27). It is easy to see that this Gorgias code can be developed directly from the policy requirements expressed in a structured natural language form as that used in the policy for the example.

### *3.3. Explainable output*

In this section we will present the way that Gorgias Cloud aids the developer to validate or debug the authored policy. The user can define testing scenarios and save them to files and s/he can use these pre-defined files for testing or compose scenarios dynamically using the Gorgias Cloud IDE.

For our running example, we can have a number of scenarios to test. One of them, "Test4.pl", is presented in Listing 2. In line 4 the reader can see a defeasible knowledge item, i.e. that the post with id *p1* is negative. The method to characterize a post as positive or negative, or assign a topic to it is out of the scope of this paper. Usually, it is done using clustering, or, supervised learning methods, see, e.g. [26, 27]. Clustering is a machine learning task that involves finding natural grouping in data. It is also called "topic modeling" when clustering is applied on text files [27]. The important observation is that the Gorgias system (and in fact other argumentation systems) can operate on top of such modules.

A screenshot of the Gorgias Cloud IDE is shown in Figure 3. "Test4.pl" has been loaded and the query for the Position *hide(Post)* has been "Run" by the user (green button at the bottom of the screen). In this we see the internal ode Explanation supporting the admissibility of the position to hide the post as well as an application level explanation that is build from the internal explanation.

For the argumentation framework of Gorgias this correspondence between admissible coalitions of arguments and associated explanations at the cognitive level of the application can be constructed naturally from the internal Explanation, $E$, returned by the Gorgias system. We can automatically use the information (i.e. argument rule names) in $E$ to construct an explanation, at the application level, that exhibits the desired characteristics of being *attributive, contrastive and actionable* as follows:

- **Attributive:** Extracted from the object-level argument rules in $E$.
- **Contrastive:** Extracted from the priority argument

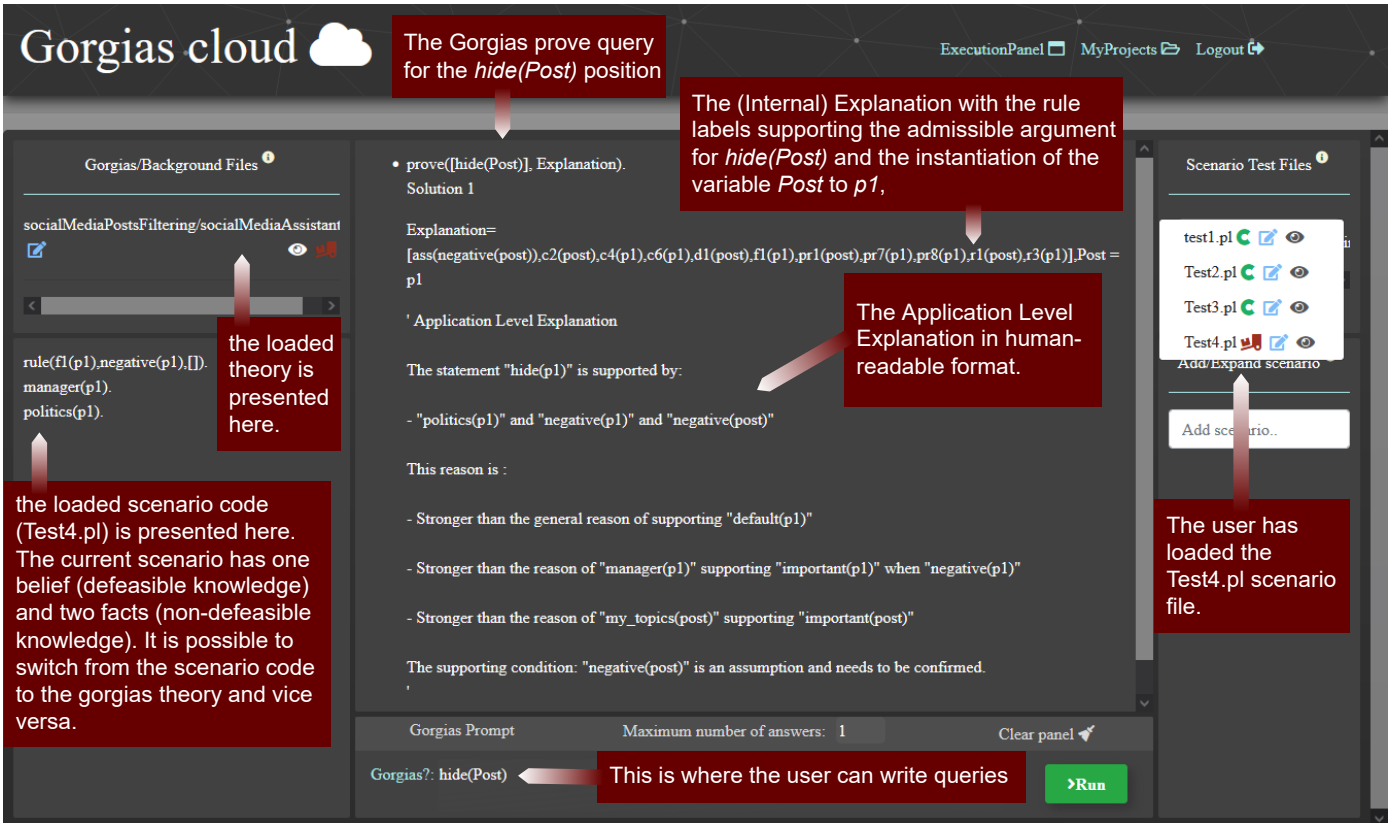


Figure 3: Execution Panel of Gorgias Cloud, the theory on the left and the results of the query in the center.

rules in $E$.

- **Actionable:** Extracted from the hypothetical or abducible arguments in $E$.

These characteristics in the human-readable "Application Level Explanation" part of the output of the Gorgias Cloud IDE can be seen for our example in Listing 3. The attributive part of the explanation is in lines 8-9, while the contrastive part is in lines 10-12. In this example we do not have an actionable part.

If we consider the test file shown in Listing 4 the results will be the ones shown in Listing 5. Here, besides the attributive (lines 8-9) and the contrastive (lines 10-13) parts we also find the actionable part in line 14. The supporting condition *negative(p2)* is provided by the abducible in line 28 of the theory (Listing 1). As the abducible represents an assumption, the action to verify it is suggested by the "Application Level Explanation" result.

### *3.4. Generating Application Level Explanations*

Let us summarize informally the automatic process of extracting an Application Level Explanation from the internal code Explanation that the Gorgias system returns. Such internal explanations contain the rule *labels* that were used to admissibly support the conclusion. For example, in Listing 3 the "Explanation" result is the $Explanation = [c4(p1), c6(p1), f1(p1), pr7(p1), pr8(p1), r3(p1)]$, representing the (composite) admissible argument for the given Position.

Initially, the priority rule labels, if any, are isolated in the *Explanation* list. Moreover, for each priority rule of the list, the priority level is calculated. Next, the list is iterated and overlapping rules are deleted. That is, the lower priority rules in the rule priority hierarchy are deleted. Only high priority rules, from which object level arguments are derived, are kept in the list.

Then the list is sorted in an ascending order. For each high level priority rule that is in the list: We recursively iterate the priority rules that lead to the weaker option, of the specific high level priority rule. The recursive iteration ends when the weaker object level rule is reached. During the recursive iteration of the high level priority list, a new list is created which contains:

- The weaker option for each high level priority rule,
- The weaker option supporting facts,
- The stronger conclusion supporting facts,
- The high level priority rule, supporting arguments, that hold against the weaker option,

Finally, the supported Position is presented, along with its supporting arguments. Furthermore, for each overridden Position, its supporting information is presented, along

Listing 1: The "socialMediaAssistant.pl" code.

```
complement(hide(Post),default(Post)).
complement(default(Post),hide(Post)).
complement(hide(Post),important(Post)).
complement(important(Post),hide(Post)).
complement(default(Post),important(Post)).
complement(important(Post),default(Post)).
rule(r1(Post),default(Post),[]):-newPost(Post).
rule(r2(Post),important(Post),[]):-newPost(Post).
rule(r3(Post),hide(Post),[]):-newPost(Post).
rule(pr1(Post),prefer(r1(Post),r2(Post)),[]).
rule(pr2(Post),prefer(r1(Post),r3(Post)),[]).
rule(pr3(Post),prefer(r2(Post),r1(Post)),[]):-my_topics(Post).
rule(c1(Post),prefer(pr3(Post),pr1(Post)),[]).
rule(pr4(Post),prefer(r2(Post),r3(Post)),[]):-my_topics(Post).
rule(c2(Post),prefer(pr1(Post),pr3(Post)),[negative(Post)]).
rule(d1(Post),prefer(c2(Post),c1(Post)),[]).
rule(pr5(Post),prefer(r2(Post),r1(Post)),[]):-manager(Post).
rule(c3(Post),prefer(pr5(Post),pr1(Post)),[]).
rule(pr6(Post),prefer(r2(Post),r3(Post)),[]):-manager(Post).
rule(pr7(Post),prefer(r3(Post),r1(Post)),[]):-politics(Post).
rule(c4(Post),prefer(pr7(Post),pr2(Post)),[]).
rule(pr8(Post),prefer(r3(Post),r2(Post)),[]):-politics(Post).
rule(c5(Post),prefer(pr8(Post),pr4(Post)),[]).
rule(c6(Post),prefer(pr8(Post),pr6(Post)),[negative(Post)]).
rule(c7(Post),prefer(pr6(Post),pr8(Post)),[positive(Post)]).
abducible(positive(_), []).
abducible(negative(_), []).
```

Listing 2: The "Test4.pl" testing file code.

```
newPost(p1).
manager(p1).
politics(p1).
rule(f1(p1),negative(p1),[]).
```

with the information that caused the supported Position to be preferred over the overridden one.

Algorithm 1 describes the generation of the information we need to compose the application level explanation. To write the algorithm we have used the same terminology as earlier for presenting the code templates. In the algorithm we do not distinguish between defeasible and non-defeasible premises as the difference does not interest the user. The output of the algorithm populates the following data structures:

- **Position**: the supported position predicate.
- **actions**: the abducibles that have been employed for the decision. This is the actionable part of the explanation.
- **attacks**: each attack contains two lists, one for the supporting information for the preferred position and the other the supporting information for the overridden position. These are then put together to produce the contrastive part of the explanation as statements of the relative strength of the supporting information against the attacking information.

Listing 3: The hide(Post) query execution results for "Test4.pl".

```
prove([hide(Post)], Explanation).

Solution 1

Explanation=[c4(p1),c6(p1),f1(p1),pr7(p1),pr8(p1),r3(p1)],Post = p1

Application Level Explanation
The statement "hide(p1)" is supported by:
 - "newPost(p1)" and "politics(p1)" and "negative(p1)"
This reason is :
 - Stronger than the reason of "newPost(p1)" supporting "default(p1)"
 - Stronger than the reason of "newPost(p1)" and "manager(p1)" supporting "important(p1)" when "negative(p1)"
```

Listing 4: The test1.pl testing file code.

```
newPost(p2).
my_topics(p2).
manager(p2).
politics(p2).
```

An important part of the algorithm is the *addSupport* recursive function that iterates through the rules from top to bottom and adds the relevant supporting information either to the preferred or to the non-preferred list. Note that when a rule is non-preferred we reverse the order of the $p$ and $np$ parameters so as to have the information at the next level be inserted at the right list (listed in Algorithm 2).

### *3.5. Argumentation as a Service*

In this section we are going to illustrate how we can offer the Gorgias decision making functionality as a Service through the Gorgias Cloud system. So let us assume that the user has developed the desired decision policy and tested it through the execution panel. S/he then wishes to use this functionality from applications deployed anywhere - from mobile phones to websites.

To do this, the user will employ the services offered by the Gorgias Cloud Back-end (see section 3.1). The only requirement is that the user has registered and successfully logged-in at least once into the Gorgias Cloud website. For example, the user of our working social media assistant example has deployed a website for helping users categorize their friends posts as shown in the screenshot in Figure 4. The reader should can see that the human-readable explanation is not the one returned by the service (the one

Listing 5: The hide(Post) query execution results for "test1.pl".

```
prove([hide(Post)], Explanation).

Solution 1

Explanation=[ass(negative(p2)),c4(p2),c5(p2),c6(p2),pr7(p2),pr8(p2),r3(p2)], Post = p2

Application Level Explanation
The statement "hide(p2)" is supported by:
 - "newPost(p2)" and "politics(p2)" and "negative(p2)"
This reason is :
 - Stronger than the reason of "newPost(p2)" supporting "default(p2)"
 - Stronger than the reason of "newPost(p2)" and "my_topics(p2)" supporting "important(p2)"
 - Stronger than the reason of "newPost(p2)" and "manager(p2)" supporting "important(p2)" when "negative(p2)"
The supporting condition: "negative(p2)" is an assumption and needs to be confirmed.
```

presented in Listing 3). The web developer preferred to use the explanation returned by the API to construct an explanation customized to the specific application domain.

To illustrate the way that the API is used we include a code snipet from the PHP file that invokes the Gorgias query on the Gorgias Cloud API. This is shown in Listing 6. The reader can see the function *executeGorgias* taking as input the data that is known for the post's context. In line 3 the project folder along with the specific theory are used to configure the service request. Then, in lines 4-23 the various found defeasible and non-defeasible pieces of information are added in the *factsList* variable. In line 24 this variable is used as a parameter to the service request and in line 25 the service is configured to return only the first result (admissible argument). Line 26 defines a data structure to store whether the different options have admissible arguments and the relevant explanations. In lines 27-28 the goal of the inference is set and line 29 invokes the remote service. Lines 30-32 check the result status and store the data, before querying for the next options.

The architecture of the Social Media Assistant is shown in Figure 5. We note that the Gorgias Cloud API documentation provides some of the relevant code for developers of such systems to implement this architecture. A user can use free tools in the internet, such as the Swagger editor to generate a client in any programming language. This is how the developer of this App generated the *Gorgias Cloud API* for the PHP language. In the architecture figure, this is shown as the bottom component in the yellow compartment that connects the PHP code of the developer to the Gorgias Cloud API.

```
1  Function explain(List Δ, Theory T):
2      Set attacks = ∅
3      List actions = ∅
4      Set HPRules = ∅
5      for each label in Δ do
6          rule ← getRule(label)
7          if rule is Argument Rule then
8              Position ← rule.Position
9          else if rule is Preference Rule then
10             found ← false
11             for each label2 in Δ do
12                 rule2 ← getRule(label2)
13                 if rule2 is Preference Rule ∧
                    (rule2.Label_1 = label ∨
                    rule2.Label_2 = label) then
14                     found ← true
15             if found = false then
16                 HPRules ← HPRules ∪ rule
17         else if rule is Abducible then
18             actions.add(rule.Name)
19     for each rule in HPRules do
20         List attack = ∅
21         List p = ∅
22         p.add(rule.Premises)
23         List np = ∅
24         attack.add(p)
25         attack.add(np)
26         attacks.add(attack)
27         addSupport(rule, p, np)
28 Function getRule(String label, Theory T):
29     for each rule in T do
30         if rule is Argument Rule ∧
            rule.Argument_Name = label then
31             return rule
32         else if rule is Preference Rule ∧
            rule.Preference_Name = label then
33             return rule
```

**Algorithm 1:** Algorithm for generating the information needed for the "Application Level Explanation" feature,

## 4. Results

Gorgias Cloud has been used, over the last couple of years, in undergraduate and graduate courses at the University of Cyprus and the University of Paris City. This is used by students both to learn about argumentation and to carry out their cognitive assistant projects, implemented as services over the web similarly but at a larger and more realistic scale as the example of the social media assistant in this paper. In general, the students find the use of the

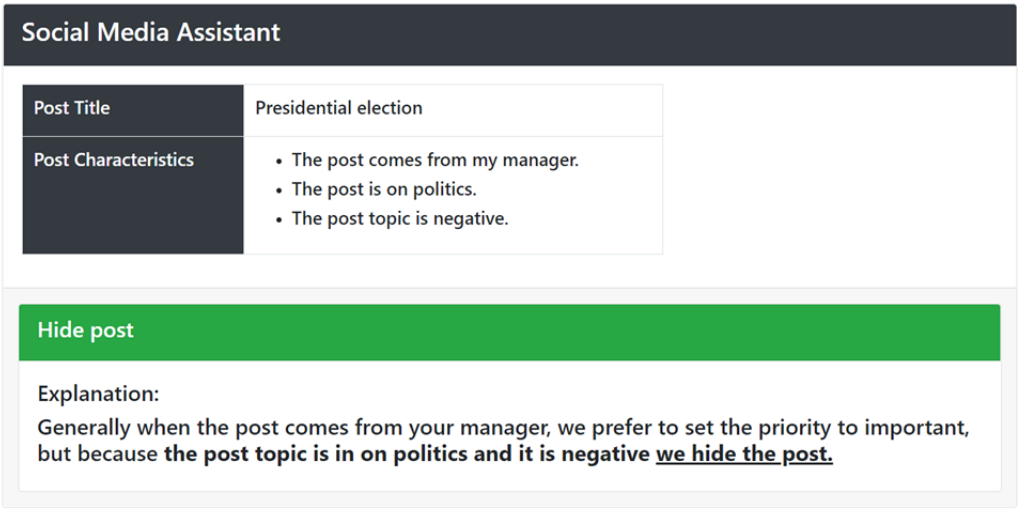


Figure 4: A screenshot of the web application for filtering posts from social media.

```
1  Function addSupport(Rule r, List p, List np):
2      if r is Preference Rule then
3          pRule ← getRule(rule.Label_1)
4          p.add(pRule.Premises)
5          addSupport(pRule, p, np)
6          npRule ← getRule(rule.Label_2)
7          np.add(npRule.Premises)
8          addSupport(npRule, np, p)
9      else
10         p.add(r.Premises)
11         p.add(r.Position)
12         return
```

**Algorithm 2:** The *addSupport* function.

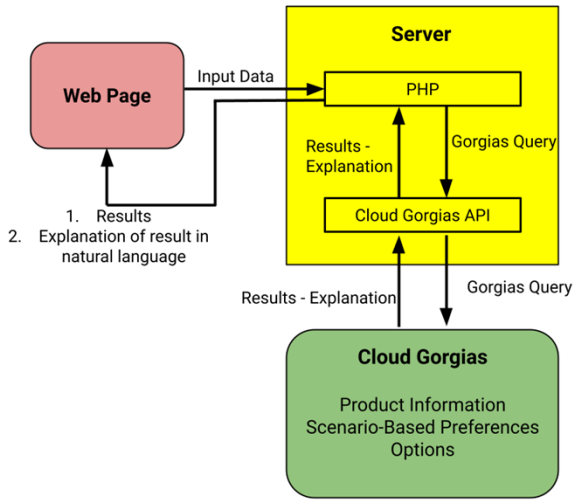


Figure 5: Social Media App Architecture Overview

system easy and very helpful for carrying out their group project. They appreciate the fact that they can build a realistic system based on the advanced technology of argumentation (most of them using the PHP and Python languages for development of their applications).

It is also used by students for their diploma thesis. One of our students in the Technical University of Crete defined and applied energy management polices into smart buildings [28]. The case study was applied on the building of Environmental Engineers of the Technical University of Crete where we used Internet of Things enabled sensors for measuring basic parameters-data, such as temperature, lighting and $CO_2$ levels. The result of the work was the implementation of energy policies that help to reduce energy consumption as well as the effective management of situations where there is a conflict of decisions [29]. The Gorgias Cloud API was invoked by a web server for the Python programming language using a Python client application.

Another student in the National Technical University of Athens delivered an explainable AI Model for ICU admission prediction of COVID19 patients [30]. The model was based on the patients' symptoms and lab results. Firstly, the student trained models with a variety of different algorithms using existing COVID19 patients' data. Then, she selected the model with the highest accuracy and created a sum of the most important rules written in the form of Gorgias arguments. Finally she developed a framework in the Java language accessing the Gorgias Cloud API using a Java client application. Similarly, a student at the University of Cyprus has used Gorgias Cloud in her machine learning project to learn an argumentation theory to predict the possibility of a gynecological cancer from real-life data on endometrial tumor images. The explanations of the resulting argumentation theory enabled the study to separate the data into groups of typical cases of malignant or benign tumours.

All in all, more than 240 students and/or researchers have used the Gorgias Cloud system over the last two years

Listing 6: An extract of the PHP file that invokes the Gorgias Cloud service.

```
public function executeGorgias($postTitle,
    $myTopicsOfInterestsPost,$managerPost,
    $politicsPost,$negativePost,$positivePost)
    {
  $gorgias_query = new GorgiasQuery();
  $gorgias_query->setGorgiasFiles(array("
      socialMediaPostsFiltering/
      socialMediaAssistant.pl"));
  $factsList=array();
  if ($myTopicsOfInterestsPost) {
    $fact="my_topics(" . $postTitle . ")";
    $factsList[]=$fact;
  }
  if ($managerPost) {
    $fact="manager(" . $postTitle . ")";
    $factsList[]=$fact;
  }
  if ($politicsPost) {
    $fact="politics(" . $postTitle . ")";
    $factsList[]=$fact;
  }
  if($negativePost){
    $fact="rule(f1(".$postTitle.
        "),negative(" . $postTitle . "),[])";
    $factsList[]=$fact;
  }elseif($positivePost){
    $fact="rule(f1(".$postTitle.
        "),positive(" . $postTitle . "),[])";
    $factsList[]=$fact;
  }
  $gorgias_query->setFacts($factsList);
  $gorgias_query->setResultSize(1);
  $gorgiasResult=array("hide"=>false, "
      hideDelta"=>array(), "important"=>false,
      "importantDelta"=>array(), "default"=>
      false,"defaultDelta"=>array());
  $query = "hide(".$postTitle.")";
  $gorgias_query->setQuery($query);
  $gorgiasQueryResult = $apiInstance->
      executeQueryUsingPOST($gorgias_query);
  if(!$gorgiasQueryResult->getHasError()&&
      $gorgiasQueryResult->getHasResult()){
    $queryResult=$gorgiasQueryResult->getResult
        ();
    $gorgiasResult["hide"]=true;
```

for their studies or research.

Gorgias Cloud, has also been used for Proof of Concepts (PoC) development. For example, in 2020, at the Technical University of Crete, we employed Gorgias Cloud to develop a Proof of Concept for automated argumentation-based PDF documents annotation (ELKE-82300 project). The PoC was successful and now the system is in production, since the start of 2022, by the Argument Theory start-up company in Paris, France.

## 5. Discussion

One of the existing systems for argumentation on the web is that of the SPINdle defeasible logic reasoner. This includes a text editor where the user writes a defeasible logic theory and can execute it online to get onclusions. The response of the service is the set of goals that are definite provable or not and defeasible provable or not.

Gorgias Cloud goes beyond this and offers an integrated application environment that facilitates the development of argumentation-based systems over the internet. Argumentation is offered as a service in a way that this allows application systems to remotely access the argumentation service and utilize the results of the argumentative computation. It appears that this is the first case where argumentation is offered to developers in an open and distributed way.

An important feature of Gorgias Cloud is the provision of Explanations for its results. These can be at the internal code level of Gorgias but also at the level of the application language that the user/programmer has used in encoding the argumentation knowledge of their problem. The latter form of explanation can be very helpful not only in understanding the results of the computation but also in evaluating the operational behavior of the Gorgias program during its development. They give a high-level view of how the results are justified at the level of the declarative understanding of the requirements of the problem by the developer. Thus, this gives a quick and cognitively easy first idea of how the program is behaving that can be very useful in evaluating the current quality of the program. Furthermore, once the development has finished and the program is utilized within an application system the application level explanations can be utilized to offer more complete and informed services to the users.

Gorgias Cloud frees the developers from writing and deploying argumentation code locally. Nevertheless, developers are still required to write and develop their argumentation theories at the internal Gorgias code level. This does not present a major difficulty with developers who have some familiarity with computing technology, but it excludes the possibility for the application domain experts to be actively involved at least partly in the development of their systems. To facilitate this, we would need high-level authoring tools that allow the direct elicitation of the argumentation knowledge of the application domain and an automatic translation of this into Gorgias code. Such tools, e.g. *Gorgias-B* [31], *Web-Gorgias-B* [7], together with a high-level methodology for software development based on argumentation [32] already exist. The next step in the development of Gorgias Cloud would be to integrate these tools as front-ends, thus providing to the developers an environment where they can work directly at the level of the systems requirements. Also the recently formed company Argument Theory operates using such an authoring tool, called *rAIson*, integrated with Gorgias as a production-level SaaS service, allowing the speedy development of argumentation-based decision modules for applications.

Gorgias Cloud also forms a step towards supporting argumentation based interaction among Web services, a feature that has already been foreseen in the semantic web community since 2007 [33]. Existing on-line argumenta-

tion systems are mostly serving the Social Semantic Web, e.g. in four types of social media: forums, wikis, blogs, and microblogs [34, 35]. According to Schneider et al. [35] arguments in the semantic web must be (1) identified, (2) resolved, (3) represented and (4) stored, (5) queried, and (6) presented to users. In our work (1)-(3) are done manually, however, (4)-(6) can be automated based on the Gorgias Cloud API.

## 6. Conclusion

We have presented the Gorgias Cloud system and its innovative features. These include an API that facilitates the development of argumentation-based systems on-line that operate using expressive explanations that are:

- **Attributive**: Explain why the supported Position is supported.
- **Contrastive**: Explain why the supported Position is preferred over others that would have been possible in the given situation
- **Actionable**: Propose that some pieces of information that have been assumed to hold so that the supported Position is preferred over others be further explored for their validity

Developers and users can use these application level explanations to evaluate the behaviour of their system with respect to the specification requirements under which the system is developed. This is made easy as these explanations are at the same high cognitive and language level as that of the application domain of the system.

Future work lies in the direction of integrating the Gorgias Cloud system with the *Web-Gorgias-B* system that allows users that are not familiar with argumentation to develop their decision policies. It is known that most users do not have formal training in logic or argumentation [34]. Through this integration naive users will be able not only to develop their theories but also to deploy them through the Gorgias Cloud API for use by their applications. Independent developers are already looking forward to this integration [36].

We would also like to integrate semantic web capabilities to the Gorgias Cloud service in such a way that background knowledge can come from ontologies or knowledge available on the web. Moreover, we could link the decision making capabilities in the Gorgias Cloud system to the semantic web so that whenever a web user wants to take a decision a relevant service can be invoked. This calls for more extensions to Gorgias Cloud, e.g. the capability to use a service without logging in by, e.g. automatically registering a new user through the API.

## References


[1] F. Cerutti, S. A. Gaggl, M. Thimm, J. P. Wallner, Foundations of implementations for formal argumentation, The IfCoLog Journal of Logics and their Applications; Special Issue Formal Argumentation 4 (8) (September 2017).

[2] A. C. Kakas, P. Moraitis, Argumentation based decision making for autonomous agents, in: The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings, ACM, 2003, pp. 883–890. doi:10.1145/860575.860717.

[3] D. Gaertner, F. Toni, Computing arguments and attacks in assumption-based argumentation, IEEE Intelligent Systems 22 (6) (2007) 24–33. doi:10.1109/MIS.2007.105.

[4] A. J. García, G. R. Simari, Defeasible logic programming: An argumentative approach, TPLP 4 (1-2) (2004) 95–138. doi:10.1017/S1471068403001674.

[5] M. Snaith, C. Reed, TOAST: online aspic$^+$ implementation, in: Computational Models of Argument - Proceedings of COMMA 2012, Vienna, Austria, September 10-12, 2012, 2012, pp. 509–510. doi:10.3233/978-1-61499-111-3-509.

[6] H.-P. Lam, G. Governatori, The Making of SPINdle, in: A. Paschke, G. Governatori, J. Hall (Eds.), Proceedings of the 2009 International Symposium on Rule Interchange and Applications (RuleML 2009), Springer-Verlag, Las Vegas, Nevada, USA, 2009, pp. 315–322. doi:10.1007/978-3-642-04985-9_29.

[7] N. Spanoudakis., K. Kostis., K. Mania., Web-gorgias-b: Argumentation for all, in: Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART,, INSTICC, SciTePress, 2021, pp. 286–297. doi:10.5220/0010269402860297.

[8] A. C. Kakas, P. Mancarella, P. M. Dung, The acceptability semantics for logic programs, in: Proc. of 11th Int. Conf. on Logic Programming, 1994, pp. 504–519.

[9] A. C. Kakas, P. Moraitis, N. I. Spanoudakis, *GORGIAS*: Applying argumentation, Argument & Computation 10 (1) (2019) 55–81. doi:10.3233/AAC-181006.

[10] K. Pendaraki, N. Spanoudakis, Portfolio performance and risk-based assessment of the portrait tool, Operational Research 15 (3) (2015) 359–378. doi:10.1007/s12351-014-0162-9.

[11] P. Moraitis, N. I. Spanoudakis, Argumentation-based agent interaction in an ambient-intelligence context, IEEE Intelligent Systems 22 (6) (2007) 84–93. doi:10.1109/MIS.2007.101.

[12] I. A. Letia, M. Acalovschi, Achieving competence by argumentation on rules for roles, in: Engineering Societies in the Agents World V, 5th Int. Workshop, ESAW 2004, Toulouse, France, October 20-22, 2004, Revised Selected and Invited Papers, 2004, pp. 45–59. doi:10.1007/11423355_4.

[13] N. Spanoudakis, P. Moriaitis, Engineering an agent-based system for product pricing automation, Engineering Intelligent Systems 17 (2) (2009) 139.

[14] A. K. Bandara, A. C. Kakas, E. C. Lupu, A. Russo, Using argumentation logic for firewall configuration management, in: Integrated Network Management, IM 2009. 11th IFIP/IEEE International Symposium on Integrated Network Management, Hofstra University, Long Island, NY, USA, June 1-5, 2009, 2009, pp. 180–187. doi:10.1109/INM.2009.5188808.

[15] Y. Benazzouz, D. Boyle, Negotiation and Argumentation in Multi-Agent Systems: Fundamentals, Theories, Systems and Applications, Bentham Science Publisher, 2014, Ch. Argumentation-Based Conflict Resolution in Pervasive Services, pp. 399–419. doi:10.2174/9781608058242114010 1.

[16] Y. Benazzouz, N. Sabouret, B. Chikhaoui, Dynamic service composition in ambient intelligence environment, in: 2009 IEEE International Conference on Services Computing (SCC 2009), 21-25 September 2009, Bangalore, India, 2009, pp. 411–418. doi:10.1109/SCC.2009.16.

[17] K. Čyras, A. Rago, E. Albini, P. Baroni, F. Toni, Argumentative XAI: A survey, in: Z.-H. Zhou (Ed.), Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21, International Joint Conferences on Artificial Intelligence Organization, 2021, pp. 4392–4399. doi:10.24963/ijcai.2021/600.

[18] A. Vassiliades, N. Bassiliades, T. Patkos, Argumentation and explainable artificial intelligence: a survey, The Knowledge Engineering Review 36 (2021).
[19] T. Miller, Explanation in artificial intelligence: Insights from the social sciences, Artificial Intelligence 267 (2019) 1–38. doi:10.1016/j.artint.2018.07.007.
[20] G. Kunz, Mastering Angular Components: Build component-based user interfaces using Angular, Packt Publishing Ltd, 2018.
[21] A. Leff, J. T. Rayfield, Web-Application Development Using the Model/View/Controller Design Pattern, in: 5th International Enterprise Distributed Object Computing Conference (EDOC 2001), 4-7 September 2001, Seattle, WA, USA, Proceedings, IEEE, 2001, pp. 118–127. doi:10.1109/EDOC.2001.950428.
[22] C. Walls, Spring Boot in action, Simon and Schuster, 2015.
[23] J. B. Ottinger, J. Linwood, D. Minter, Beginning Hibernate: For Hibernate 5, Apress, 2016.
[24] C. Anderson, Docker [software engineering], IEEE Software 32 (3) (2015) 102–105. doi:10.1109/MS.2015.62.
[25] T. Ali, Z. Najem, M. Sapiyan, Jpl : Implementation of a prolog system supporting incremental tabulation, in: Sixth International conference on Computer Science and Information Technology (CCSIT 2016), Zurich, Switzerland, January 02-03, 2016, 2016, pp. 323–338. doi:10.5121/csit.2016.60127.
[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Advances in neural information processing systems 30 (2017).
[27] P. Kherwa, P. Bansal, Topic modeling: a comprehensive review, EAI Endorsed transactions on scalable information systems 7 (24) (2020).
[28] P. Krinakis, Development of an intelligent system for decision policy into smart buildings with the use of hierarchical argumentation theory, Master's thesis, Technical University of Crete (2021). doi:10.26233/heallink.tuc.89787.
[29] N. Bassiliades, N. I. Spanoudakis, A. C. Kakas, Towards multipolicy argumentation, in: Proceedings of the 10th Hellenic Conference on Artificial Intelligence, SETN '18, Association for Computing Machinery, New York, NY, USA, 2018. doi:10.1145/3200947.3201032.
[30] E. Dazea, An explainable ai model for icu admission prediction of covid19 patients, Master's thesis, National Technical University of Athens (2021). doi:10.26240/heal.ntua.21812.
[31] N. I. Spanoudakis, A. C. Kakas, P. Moraitis, Gorgias-b: Argumentation in practice, in: P. Baroni, T. F. Gordon, T. Scheffler, M. Stede (Eds.), Computational Models of Argument - Proceedings of COMMA 2016, Potsdam, Germany, 12-16 September, 2016, Vol. 287 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2016, pp. 477–478. doi:10.3233/978-1-61499-686-6-477.
[32] N. I. Spanoudakis, A. C. Kakas, P. Moraitis, Applications of argumentation: The soda methodology., in: ECAI, 2016, pp. 1722–1723.
[33] P. Torroni, M. Gavanelli, F. Chesani, Argumentation in the semantic web, IEEE Intelligent Systems 22 (6) (2007) 66–74.
[34] J. Schneider, A. Passant, T. Groza, J. G. Breslin, Argumentation 3.0: how semantic web technologies can improve argumentation modeling in web 2.0 environments, in: Computational Models of Argument, IOS Press, 2010, pp. 439–446.
[35] J. Schneider, T. Groza, A. Passant, A review of argumentation for the social semantic web, Semantic Web 4 (2) (2013) 159–218.
[36] S. Almpani, Y. Kiouvrekis, P. Stefaneas, P. Frangos, Computational argumentation for medical device regulatory classification, International Journal on Artificial Intelligence Tools 31 (01) (2022) 2250005.