DDC Project Report

Group Members:

Youssef Abida
Thomas Aubourg
Nathan Corr
Edgar Demeude

October 19, 2025

Abstract

This report presents the work conducted as part of the DDC project, which explores automated reasoning and analysis within computational argumentation. The project integrates three complementary components: **(1)** logical reasoning with Assumption-Based Argumentation (ABA and ABA+), **(2)** relation-based argument mining using transformer models such as RoBERTa, and **(3)** visualization of gradual semantics through the weighted h-categorizer framework.

The first part focuses on generating and analyzing structured argumentative frameworks, handling normal and preference-based attacks. The second part introduces a machine learning pipeline for detecting argumentative relations (attack, support) between text segments. The third part extends the reasoning layer with numerical interpretation of argument strengths, visualized as acceptability degree spaces.

Together, these components form an end-to-end system that connects symbolic reasoning, data-driven learning, and semantic visualization, offering a comprehensive view of argument dynamics from both formal and empirical perspectives.

Contents

1 Introduction

The goal of this project is to explore different approaches to automated reasoning and argument analysis through the implementation of three complementary components:

- an **Assumption-Based Argumentation (ABA) framework generator**;
- a **Relation-Based Argument Mining (RBAM)** module;
- a **Gradual Semantics visualizer**.

Each part focuses on a distinct level of argumentation: from formal logical reasoning to data-driven argument extraction and numerical evaluation of argument strength. Together, these components illustrate the link between formal argumentation theory and its computational and data-driven applications.

The project is organized into two repositories:

- **argument-backend** (Python, FastAPI): handles all logical reasoning components of the system. It parses user input, generates the ABA framework, constructs arguments and attacks, applies Dung-style semantics (admissible, preferred, complete, and stable), performs relation-based argument mining, and computes gradual semantics values.
- **argument-frontend** (Next.js, TypeScript + React): provides an intuitive web interface for entering ABA frameworks, submitting them to the backend API, and visualizing argument structures, relations, and gradual evaluations.

The web application is deployed online, with the frontend hosted on Vercel and the backend hosted on Hugging Face Spaces, allowing users to interact with the system directly from a browser:

- Frontend: arguments-visualisation.vercel.app
- Backend: huggingface.co/spaces/Edgar-Demeude/argument-backend

Part 1 – ABA Framework Generator. Following the formal definition of the ABA framework introduced by Toni (2014), the system allows users to define:

- a language of literals;
- a set of inference rules;
- a collection of assumptions;
- a contrary mapping.

From these components, the application automatically constructs all possible arguments, identifies attacks between them, and ensures that the resulting ABA framework is both non-circular and atomic. The corresponding argumentation graph is generated and evaluated under standard Dung-style semantics.

Part 2 – Relation-Based Argument Mining (RBAM). This module applies natural language processing techniques to identify and classify argumentative relations such as *support* and *attack* between textual statements. It bridges the gap between unstructured text and the formal argumentation structures used in ABA, allowing users to extract relational data directly from real-world discourse.

Part 3 – Gradual Semantics Visualization. The final component computes and visualizes weighted gradual semantics, providing a numerical evaluation of each argument's acceptability.

This visualization enables users to explore how argument strength evolves depending on different parameters.

1.1 Setup. To ensure reproducibility and ease of use, the project is divided into two main repositories: one for the backend and one for the frontend. Each repository includes a detailed `README` describing installation steps, dependencies, and environment configuration.

- **Backend installation:**

  ```
  git clone https://github.com/edgar-demeude/argument-backend.git
  ```

  Refer to the backend's documentation for setup instructions and API usage.

- **Frontend installation:**

  ```
  git clone https://github.com/edgar-demeude/argument-frontend.git
  ```

  The frontend can be run locally or connected to the deployed backend.

Users can either run both components locally following the repository guides or use the online version to interact with the project directly from their browser.

2 Part 1: Reasoning with ABA

2.1 Description

Assumption-Based Argumentation (ABA) is a formal reasoning framework designed to model deductive and conflict-based reasoning between sets of assumptions. This section presents the implementation of a complete ABA system in Python, capable of constructing, transforming, and visualizing both standard ABA frameworks and their extended version, **ABA+**, which incorporates preference handling between assumptions.

ABA Visualization

2.2 ABA Implementation

The implementation is structured into several classes, each playing a distinct role in modeling an assumption-based argumentation framework:

- **Literal:** Represents a literal (the basic element of the language), identified by its name.
- **Rule:** Encodes a logical rule of the form $h \leftarrow b_1, \ldots, b_n$, with attributes for the rule name, head literal, and body literals.
- **Contrary:** Represents a contrariness relation between two literals (the contrary and the opposed literal).
- **Attack:** Represents an attack relation between arguments, storing the attacker and target arguments.
- **Argument:** Represents an argument constructed from rules and assumptions, defined by its name, conclusion (claim), and supporting assumptions.
- **ABAFramework:** The main class responsible for building, transforming, and visualizing both ABA and ABA+ frameworks.

2.3 Building the ABA Framework

The ABA framework is generated through the `build_aba_framework()` function. This function parses a structured text file to extract the core components (language, rules, assumptions, contraries, and preferences) and instantiates an `ABAFramework` object. Users can then choose to generate a standard ABA framework or extend it into an ABA+ framework, incorporating preferences between assumptions.

2.4 Framework Transformation

Before generating arguments, the framework must be ensured to be both non-circular and atomic. This transformation is handled by the `transform_aba()` method:

Circularity Detection — `_is_aba_circular()` A dependency graph verifies if distinct derivations yield the same literal label. If a reachability search shows a dependency between distinct rule instances of the same label, the framework is circular.

Cycle Removal — `_make_aba_not_circular()` This method computes $k = |L| - |A|$. For atomic and non-atomic rules, it generates new renamed heads and bodies to remove circular dependencies.

Atomicity Verification — `_is_aba_atomic()` Checks that every rule's body (if non-empty) contains only assumptions.

Atomic Transformation — `_make_aba_atomic()` For each non-assumption literal x, two new assumptions x_d and x_{nd} are introduced. Non-assumptions in rules are replaced by x_d, and new contraries `Contrary(`x_d`, `x_{nd}`)` and `Contrary(`x_{nd}`, `x`)` are added, making all rule bodies assumption-only.

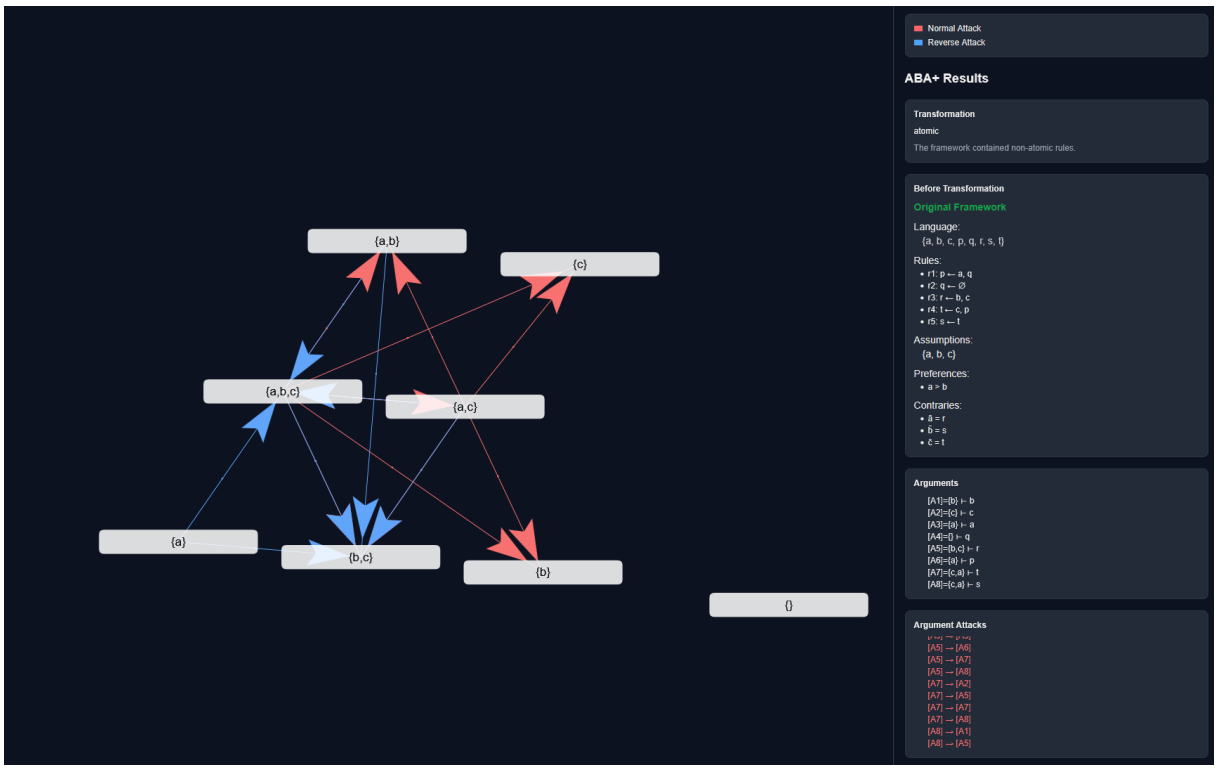
2.5 Argument and Attack Generation

Argument Generation — `generate_arguments()` All possible arguments are constructed from the rules and assumptions using a breadth-first (BFS) strategy: elementary arguments are created and recursively combined using applicable rules.

Attack Generation — `generate_attacks()` Attacks are derived from contrariness relations: A_1 attacks A_2 if A_1 concludes a literal that is contrary to one of A_2's assumptions.

2.6 ABA+ Extension: Handling Preferences

The ABA+ extension introduces preference management between assumptions.



ABA+ Implementation

Preference Integration — `make_aba_plus()` This method generates all combinations of assumptions and computes both normal and reverse attacks, resulting in an enriched ABA+ framework.

Normal and Reverse Attacks — `_generate_normal_reverse_attacks()`

- **Normal attack:** X attacks Y if an argument derived from X attacks one derived from Y, and no assumption in Y is preferred over those in X.
- **Reverse attack:** Y reverse attacks X if an argument of X would normally attack Y, but a Y assumption is strictly preferred over one in X.

2.7 Visualization

The framework includes visualization utilities: `plot_aba_graph()` for standard ABA and `plot_aba_plus_graph` for ABA+, which uses solid black lines for normal attacks and dashed red lines for reversed attacks.

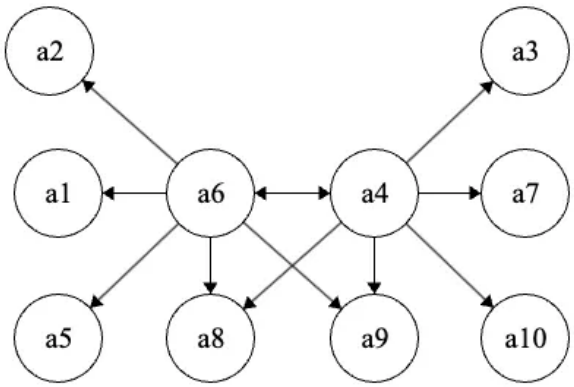
2.8 Results: ABA and ABA$^+$ Frameworks

Here are two examples with their respective visualizations compared to the expected outputs, illustrating the correct construction of ABA and ABA$^+$ frameworks and the proper handling of normal and reverse attacks.

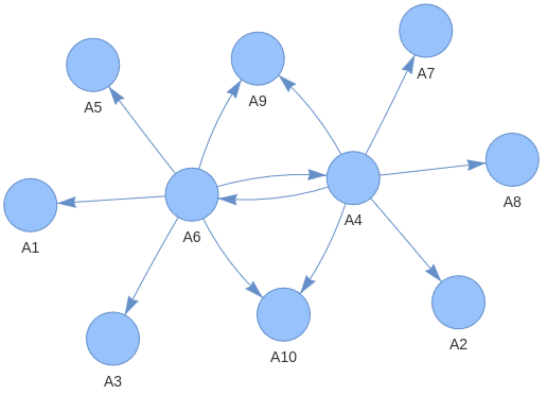
Example 1: Circular and Non-Atomic ABA Framework

Let us consider the following circular and non-atomic ABA framework $(L, R, A, \bar{\cdot})$:

- $L = \{a, b, x, y, z\}$
- $R = \{y \leftarrow b;\ y \leftarrow y;\ x \leftarrow x;\ x \leftarrow a;\ z \leftarrow x, y\}$
- $A = \{a, b\}$
- $\overline{a} = y,\ \overline{b} = x$



Expected output

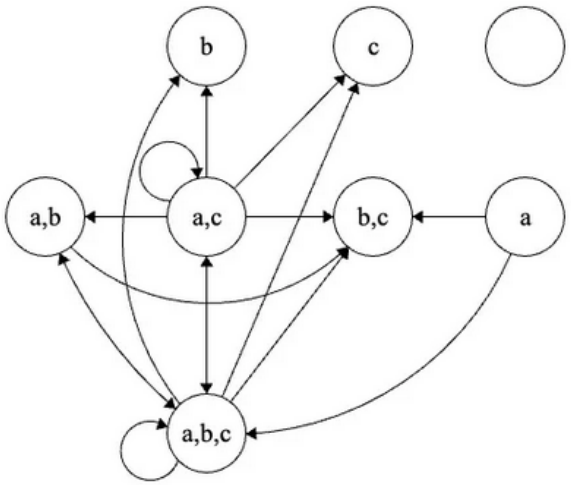


Generated output

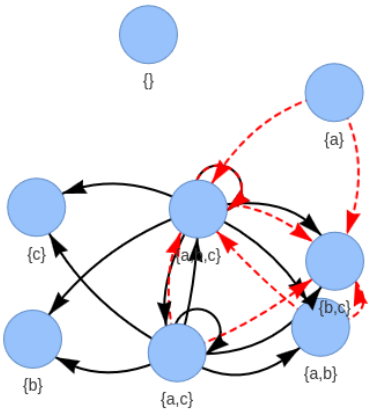
Example 2: ABA$^+$ Framework

Let us consider the following ABA$^+$ framework as a tuple $(L, R, A, \bar{\cdot}, \leq)$:

- $L = \{a, b, c, q, p, r, s, t\}$
- $R = \{p \leftarrow q, a;\ q \leftarrow i;\ r \leftarrow b, c;\ t \leftarrow p, c;\ s \leftarrow t\}$
- $A = \{a, b, c\}$
- $\overline{a} = r,\ \overline{b} = s,\ \overline{c} = t$
- $a > b$



Expected output



Generated output

Red dotted arrows = reverse attacks
Solid black arrows = normal attacks

2.9 Conclusion

This modular implementation enables:

- Dynamic construction of ABA frameworks from structured text input;
- Enforcement of structural correctness (non-circularity, atomicity);
- Extension into ABA+ frameworks with preference handling;
- Graphical visualization of argumentative interactions using PyVis.

Together, these features provide a solid foundation for automated reasoning and argumentation analysis.

3 Part 2: Relation-Based Argument Mining

3.1 Data Collection

The dataset was constructed by scraping debates from the Kialo platform, where arguments are structured as propositions connected by explicit *support* or *attack* relations. For each argument, the following fields were extracted:

- **Parent text**: the content of the source argument;
- **Child argument**: the target proposition of the relation;
- **Relation type**: either *support* or *attack*.

The resulting dataset comprises multiple debates, totaling thousands of arguments and relations. For experimental flexibility, subsets of varying sizes (10k, 25k, and 50k examples) were generated and are available in the `data/kialo/` directory of the GitHub project's notebooks.

3.2 Data Exploration

Initial exploration showed a balanced class distribution (*support* vs *attack*), arguments between 20 and 150 words, and required the removal of non-English debates.

3.3 Classification approach

3.3.1 Preprocessing

All textual data was preprocessed through lowercasing, removal of special characters, tokenization (WordPiece or simple), and truncation/padding to a maximum sequence length of 512 tokens.

3.3.2 Model Training

We applied three main classification strategies:

1. **Embedding + MLP:** Sentence embeddings (*all-mpnet-base-v2*) fed into a multi-layer perceptron (25 epochs).
2. **BERT Fine-Tuning:** Fine-tuning *roberta-base* for classification (3 epochs).
3. **LLMs:** Testing small LLMs for text classification in a few-shot setup, leveraging instruction-based learning.

3.3.3 Evaluation Metrics

Models were evaluated using **Accuracy (Acc)**, **F1-Score (F1)** (harmonic mean of precision and recall), **Precision (Prec)** and **Recall (Rec)**.

3.4 Classification Results

We compare several model architectures applied to the argument relation classification task, evaluated on datasets of increasing sizes (10k, 25k, and 50k pairs), all trained for 3 epochs.

3.4.1 Models Evaluated

- **Embeddings + MLP:** A lightweight baseline model using pre-trained sentence embeddings as input features to a multi-layer perceptron classifier. It provides a fast and scalable approach requiring minimal training resources.
- **DistilBERT-base:** A distilled version of BERT, offering a balance between performance and efficiency through reduced model size while preserving contextual understanding.
- **RoBERTa-base:** A robust transformer encoder achieving state-of-the-art performance on various NLP benchmarks. It serves as a stronger baseline for fine-tuning on relational classification.
- **Mistral-7B (4-bit):** A large language model evaluated in a zero-shot or few-shot setting using prompt-based inference and 4-bit quantization to reduce memory footprint.
- **Qwen-3 4B (Thinking):** A reasoning-oriented LLM tested under the same prompt-based approach, designed for efficient inference and strong instruction-following capabilities.

3.4.2 Fine-tuned Transformer and Embedding Models

Table 1 reports the performance of models trained on progressively larger datasets. As expected, performance improves with data size for all models.

The results indicate that the **Embeddings + MLP** model provides stable and fast performance, making it suitable for rapid experimentation and large-scale training. However, transformer-based encoders outperform it in accuracy and F1-score, especially **RoBERTa-base**, which reaches 0.82 F1 on the largest dataset. The main limitation is the substantial increase in training time and computational cost as model size grows.

3.4.3 Prompt-based Zero-shot Evaluation with LLMs

To assess the capabilities of large language models without fine-tuning, we conducted zero-shot and few-shot evaluations using prompt engineering (see Table 2). Following prior research, a fixed "primer" composed of four RBAM (Relation-Based Argument Mapping) examples was prepended to each prompt, followed by a test pair of arguments.

Despite the absence of fine-tuning, both models achieve fairly decent performance compared to smaller supervised models on the 10k dataset. **Qwen-3 4B (Thinking)** slightly outperforms **Mistral-7B (4-bit)** across all metrics while offering faster inference, likely due to its optimized reasoning pipeline. These results indicate that prompt-based LLMs can generalize relational patterns effectively. However, their performance might not be fully optimal due to limited prompt engineering, time constraints, and restricted computational resources. Fine-tuned models still tend to outperform them when sufficient labeled data and training time are available.

Table 1: Performance of Fine-tuned Models Across Different Dataset Sizes

Metric	Embeddings + MLP	DistilBERT-base	RoBERTa-base
Dataset: 10k			
F1-Score	0.72	0.69	**0.79**
Accuracy	0.71	0.69	**0.79**
Precision	0.70	0.69	**0.79**
Recall	0.73	0.69	**0.79**
Training Time	2 min	5 min	9 min
Dataset: 25k			
F1-Score	0.73	0.70	**0.80**
Accuracy	0.72	0.70	**0.80**
Precision	0.72	0.70	**0.80**
Recall	0.73	0.70	**0.80**
Training Time	6 min	11 min	21 min
Dataset: 50k			
F1-Score	0.74	0.73	**0.82**
Accuracy	0.73	0.73	**0.82**
Precision	0.73	0.73	**0.82**
Recall	0.74	0.73	**0.82**
Training Time	12 min	23 min	43 min

Table 2: Zero-shot / Few-shot LLM Classification Results (Prompt-based)

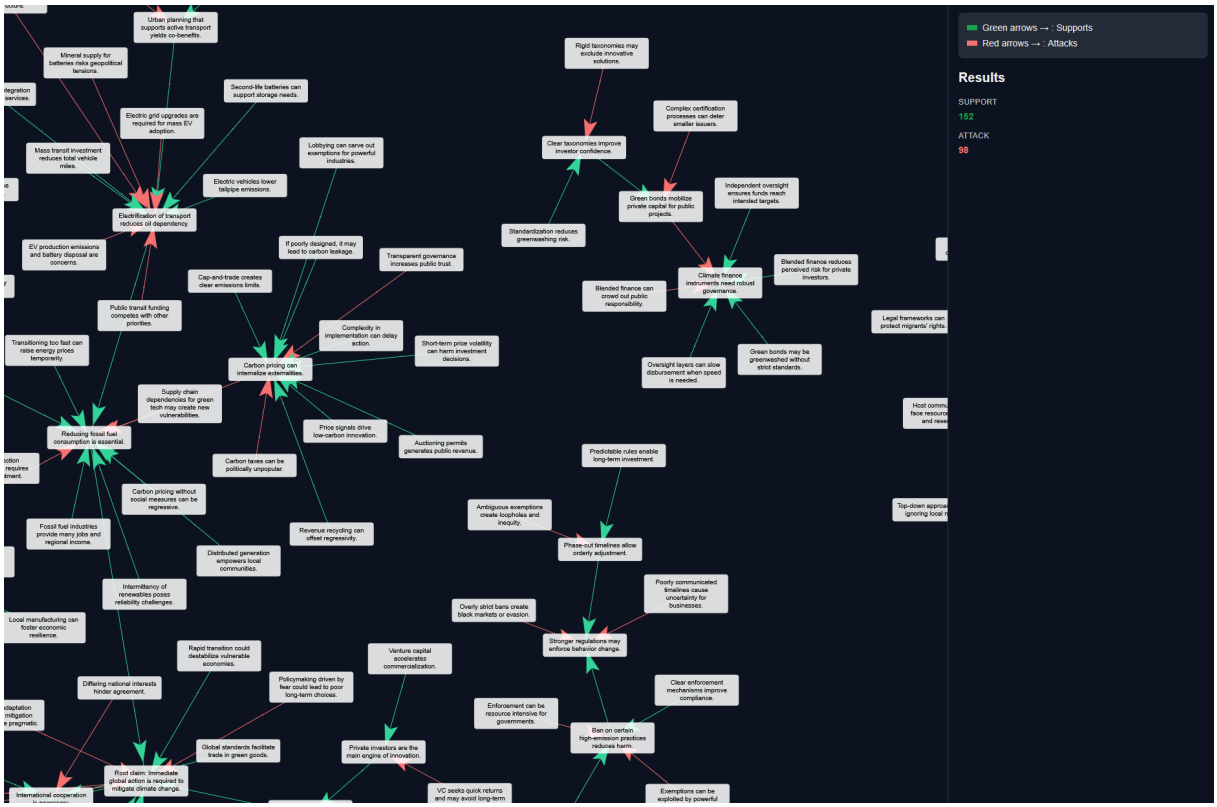
Metric	Mistral-7B (4-bit)	Qwen-3 4B (Thinking)
F1-Score	0.62	**0.68**
Accuracy	0.62	**0.68**
Precision	0.63	**0.68**
Recall	0.61	**0.67**
Inference Time	0.14 s/pair	0.08 s/pair

3.4.4 Discussion

Overall, the experiments confirm three main findings:

1. Increasing dataset size benefits all models, with diminishing returns beyond 25k pairs.
2. Transformer-based encoders, especially RoBERTa, consistently outperform embedding-based baselines.
3. Quantized LLMs offer an efficient zero-shot alternative for quick prototyping or low-resource setups.

In summary, fine-tuning compact transformer encoders remains the most effective strategy for high accuracy, while quantized LLMs provide flexibility for scalable and interpretable inference without additional training.



RBAM visualization

4 Part 3: Weighted h-Categorizer Semantics

4.1 Implementation

This part focuses on the implementation of the weighted h-categorizer semantics within gradual argumentation frameworks. The goal of this work was to approximate and explore the *acceptability degree spaces* resulting from different weight distributions on arguments.

The iterative computation of the weighted h-categorizer semantics follows the recursive definition of HC_k:

$$HC_{k+1}(a) = \frac{w(a)}{1 + \sum_{b \in Att(a)} HC_k(b)}$$

A convergence threshold ϵ is used to terminate the iteration once stability is achieved for all arguments. This process operates on a weighted argumentation graph defined as:

$$A = \{a_1, a_2, a_3, \dots\}, \quad R = \{a_i \to a_j\}, \quad w = [w_1, w_2, w_3, \dots]$$

Each argument is assigned an acceptability degree between 0 and 1. Formally, the corresponding mapping function is defined as:

$$F = (A, R, w) \Rightarrow \Sigma HC(F) : A \to [0, 1]$$

Function explanation. The following functions structure the implementation of the gradual semantics:

- **Method** `build_att(A, R)` Constructs the attack dictionary for a given argumentation framework. Each argument is associated with the list of arguments that attack it.

Input: A: list of arguments. R: set of attack relations represented as pairs $(attacker, target)$.

Process: Iterates through each attack relation and appends the attacker to the list of attackers of the target argument.

Output: Returns a dictionary $\{a : [attackers\_of\_a]\}$. $\Rightarrow$ This preprocessing step simplifies later computations by allowing constant-time access to each argument's attackers.

- **Method** `h_categorizer(A, R, w, max_iter, epsi=1e-4)` Implements the core iterative computation of the weighted h-categorizer semantics. This method computes the acceptability degree of each argument according to the recursive definition above.

  Input: A: list of arguments. R: attack relations. w: dictionary of initial weights, one per argument. max_iter: maximum number of iterations. ϵ: convergence threshold.

  Process: Builds the attacker list using `build_att()`. Initializes all acceptability degrees as $HC_0(a) = w(a)$. Iteratively updates each argument's value using the recursive formula. Stops when the maximum change between two iterations is below ϵ.

  Output: Returns a dictionary $\{a : HC(a)\}$ mapping each argument to its converged acceptability value. $\Rightarrow$ This function operationalizes the gradual semantics by iteratively balancing argument strengths until stability is reached.

- **Method** `dict_to_vector(A, d)` Converts a dictionary of argument values into a vector representation following a fixed order on A.

  Input: A: ordered list of arguments. d: dictionary $\{argument : value\}$.

  Output: Returns a NumPy vector $[d[a_1], d[a_2], \dots]$ in the order of A. $\Rightarrow$ This ensures consistency between the argument order and the corresponding coordinates used in geometric visualizations.

- **Method** `sample_and_compute_X(A, R, epsilon=1e-4, max_iter=1000, n_samples=10000, seed=42)` Generates a large set of random weighted frameworks and computes their corresponding acceptability vectors under the h-categorizer semantics.

  Input: A: list of arguments. R: attack relations. ϵ, max_iter: parameters controlling convergence. $n\_samples$: number of random samples to generate. $seed$: random generator seed for reproducibility.

  Process: Generates $n\_samples$ random weight vectors $w_i \in [0,1]^{|A|}$. For each vector, computes its acceptability degrees using `h_categorizer()`. Converts results into a NumPy matrix X where each row represents an acceptability vector.

  Output: Returns matrix $X \in [0,1]^{n\_samples \times |A|}$. $\Rightarrow$ The resulting matrix X represents the point cloud of all sampled acceptability outcomes, later used to construct the convex hull visualized in the Streamlit application.

4.2 Visualization

To study the influence of weights, we generated 100,000 random weight vectors:

$$W_i \in [0,1]^A$$

for a fixed argumentation framework (A, R). Instead of passing a matrix of weights, we fixed an arbitrary order on A so that each weight assignment w could be represented as a vector in $[0,1]^A$. For instance, if $w(a) = 0.2$ and $w(b) = 0.5$, then w is represented as $[0.2, 0.5]$.

For each sampled vector W_i, the h-categorizer semantics produces a corresponding acceptability vector:

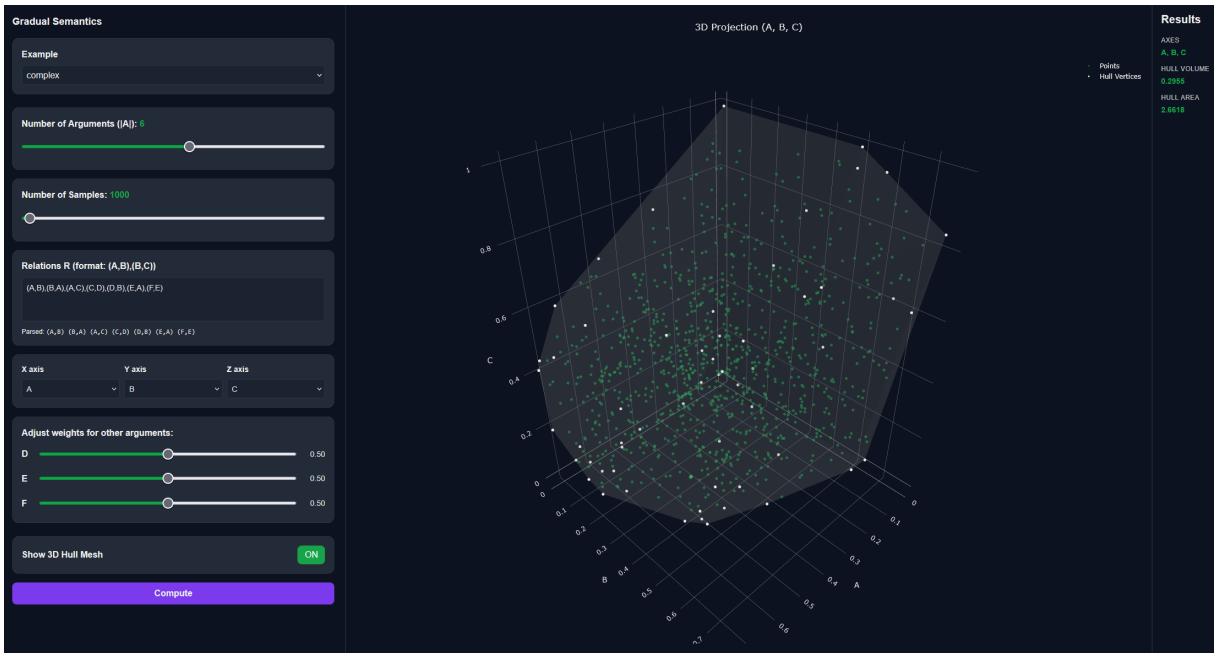
$$[x_1, x_2, x_3, \dots] \in [0,1]^{|A|}$$

where each x_j corresponds to the acceptability level of the j-th argument in the fixed ordering of A. The overall pipeline is therefore:

$$\text{Ordered weight vectors } (w_i) \longrightarrow \text{Acceptability points } (x_i) \longrightarrow \text{Convex Hull of } \{x_i\}$$

The resulting cloud of points $\{x_i\}$ was used to construct a convex hull, which serves as a geometric approximation of the acceptability degree space. This convex hull effectively captures the set of possible configurations of argument strength under the h-categorizer semantics.

Concretely, we implemented a program that takes as input an argumentation framework (A, R), samples weight vectors $w \in [0, 1]^A$ according to the fixed ordering on A, computes the corresponding acceptability vectors x, and constructs the convex hull of the resulting point cloud for visualization and further analysis.



Weighted h-Categorizer semantics visualization

5 Discussion and Conclusion

5.1 Discussion and Analysis

The project successfully integrated three complementary components (ABA reasoning, argument mining, and gradual semantics) into a unified argumentative reasoning platform. Each part contributed to bridging a different level of abstraction, from symbolic logic to data-driven learning and continuous evaluation.

From a technical standpoint, the modular design of the `argument-backend` allowed for clean separation between reasoning modules (ABA, ABA+, semantics) and machine learning pipelines (RBAM). The `argument-frontend` offered an intuitive visualization layer that made it possible to explore attacks, supports, and gradual strengths interactively.

A few points of reflection emerged during experimentation:

- **Scalability of ABA:** The framework generation process can become computationally heavy when the number of rules or assumptions grows, since the argument and attack spaces expand combinatorially. Optimizations such as memoization or lazy evaluation could reduce this complexity.

- **Expressivity vs. Performance:** Adding preference reasoning in ABA+ significantly improves realism but introduces complexity in attack computation. Maintaining a balance between theoretical accuracy and runtime efficiency is key.
- **Model generalization in RBAM:** The RoBERTa model achieved the best balance between accuracy and training cost. However, its interpretability remains limited compared to symbolic reasoning. Exploring hybrid neuro-symbolic methods could help unify both paradigms.
- **Gradual semantics behavior:** The weighted h-categorizer exhibited smooth convergence and intuitive sensitivity to attacker weights. Nonetheless, its convergence speed depends on both graph density and initial weight distribution.

5.2 Conclusion and Perspectives

Overall, the DDC project demonstrates the feasibility and usefulness of connecting symbolic argumentation frameworks with data-driven relation extraction and gradual semantics visualization.

- The **ABA generator** provided a strong formal backbone for structured reasoning.
- The **relation-based mining module** enabled automated extraction of argumentative relations from textual debates.
- The **gradual semantics visualizer** gave quantitative insight into argument strength under varying weights.

Future work could include:

- Integrating real-time reasoning over text streams using a hybrid symbolic–neural pipeline;
- Expanding the set of semantics (e.g., *categorizer*, *DF-QuAD*, or *probabilistic variants*);
- Deploying a fully interactive online dashboard combining textual analysis, graph visualization, and numerical evaluation.

In conclusion, this project successfully bridges the gap between **formal argumentation theory** and **practical computational reasoning**, demonstrating how multi-layered approaches can provide deeper insight into structured debate analysis.