Alma Mater Studiorum
Università di Bologna

Department of Computer Science and Engineering

Artificial Intelligence

Master Thesis

in

Natural Language Processing

A Multi-Agent LLM Pipeline for Argument Mining on Legal Judgments

---

Candidate

Safoura Banihashemi

Student ID: 0001109509

Supervisor

Prof. Andrea Galassi

Co-Supervisor

Dr. Giulia Grundler

Prof. Paolo Torroni

Academic year 2025–2026

1st Session

Abstract

Legal Argumentation Mining (LAM) aims to uncover the argumentative structure in legal documents. It identifies which sentences present arguments, their roles, the types of reasoning they use, and how they support or challenge each other. This process naturally breaks down into several related sub-tasks. However, most research focuses on each sub-task in isolation, and it remains unclear how these sub-tasks work together within a complete system.

This thesis presents the design, implementation, and evaluation of a five-agent LLM pipeline for LAM, using the Demosthenes corpus, an annotated dataset of decisions on fiscal State aid by the Court of Justice of the European Union. The pipeline uses a LangGraph state machine to coordinate agents that handle the different sub-tasks: detection of argumentative components, their classification in premises or conclusions, classification of factual and legal premises, identification of argumentation schemes, and directed relation identification.

The main methodological contribution is an evaluation framework to measure the performance of the system under three different settings, to quantify the intrinsic capabilities of individual agents and the impact of error propagation. First, each agent is executed in an isolated setting and evaluated on the golden data. Then, the full pipeline is evaluated both in an upstream-conditioned setting, to evaluate the agents without penalizing them for mistakes made by their predecessors, and in an end-to-end setting to evaluate the performance of the pipeline as a whole, without distinguishing the contribution of single agents.

The experiments compare a commercial model (Gemini) with an open-weight model (Gemma), and show the importance of evaluating modular LLM systems on their overall performance, not just on individual tasks.

Keywords: Legal Argument Mining, Large Language Models, Multi-Agent Systems, Prompt Engineering, Error Propagation, Argumentation Schemes, Court of Justice of the European Union.

Acknowledgements

I would like to express my deep appreciation to my supervisor, Prof. Andrea Galassi, and my co-supervisor, Dr. Giulia Grundler, for their guidance, patience, and valuable feedback throughout this project. Their technical discussions helped me clarify my ideas, guided me in designing and evaluating the experiments, and helped me catch errors I might have missed. Their focus on methodological details improved this thesis and shaped how I now conduct research.

I am also deeply grateful to my family for their constant love, support, and encouragement across the distance, and for reminding me in the most challenging moments that a difficult problem is not the same as an impossible one.

Bologna, July 13, 2026

Safoura Banihashemi

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Legal Argumentation Mining (LAM) is a subfield of Natural Language Processing (NLP) that aims to automatically identify and classify arguments in legal documents [1]. Legal documents are especially challenging because they use dense, specialized vocabulary, follow strict rhetorical rules, and contain complex logical links between arguments [2].

Early work on legal argumentation mining mostly relied on traditional machine learning methods such as Support Vector Machines (SVMs), logistic regression, and manually designed linguistic features [3]. These systems performed moderately well but often struggled to work across different legal contexts because they depended on handcrafted representations. Later research introduced deep learning models, such as recurrent neural networks (RNN) and Transformer-based models, especially those based on BERT. These newer models greatly improved contextual understanding and accuracy [4, 5].

Recently, Large Language Models (LLMs) have performed well on argument-mining tasks. Several studies have turned argumentation sub-tasks into text-generation problems [6]. Despite these advances, a key challenge remains: legal reasoning is structured and interdependent; mistakes made early on can affect later tasks, reducing the reliability of the final argument [7, 6]. In practice, a legal argument graph must meet global structural rules: conclusions must be supported by premises, argumentative links must be logically consistent, and labels across sub-tasks should remain compatible.

Multi-agent systems provide a promising way to address this limitation. Instead of using a single model for every sub-task, these systems break down a complex task into specialized agents that work together and share information through an organized process [8, 9]. This approach makes it easier to understand, control, and diagnose problems in the reasoning pipeline. Importantly, breaking tasks into

sub-tasks also makes error propagation measurable.

This thesis describes how a collaborative multi-agent system was designed, implemented, and evaluated for LAM in CJEU decisions on fiscal State aid [10]. The system uses the LangGraph framework to break the task into specialized agents, each handling a specific sub-task with its own prompting strategy [11]. A final rule-based critic agent checks the argument graph for structural problems such as orphan premises, unsupported conclusions, contradictory links, and low-confidence links, and then reports them as diagnostic annotations.

Each agent is responsible for one of the following sub-tasks: argumentative identification, argument role classification (*premise* / *conclusion*), premise type classification (*factual* / *legal*), argumentation scheme classification (*Rule*, *Prec*, *Aut*, *Class*, *Itpr*, *Princ*), and relation identification between argumentative components (*support* / *attack*) [10]. These sub-tasks are highly interdependent; for example, classifying roles and relations requires accurate detection of argumentative components.

Thesis Structure

The rest of this thesis is organized as follows:

Chapter 2. Gives an overview of the background and related work. It covers classical NLP methods, transformer-based LLMs, argumentation mining, prompting strategies, and multi-agent architectures.

Chapter 3. Explains the materials, methods, and system architecture used to build the pipeline. It covers the Demosthenes corpus, the annotation scheme, the methodology, and how the pipeline components work together in a multi-agent workflow.

Chapter 4. Explains how the system was implemented, focusing on node-level design choices and prompt-engineering strategies.

Chapter 5. Shows the experimental results, compares different prompting configurations, and provides a qualitative analysis of the generated argument graphs.

Chapter 6. Summarizes the main contributions, discusses the limitations, and suggests possible directions for future research.

Research Questions

This thesis is structured around five main research questions.

RQ1 (Decomposition) How much can decomposing the legal argument mining into specialized LLM agents improve performance?

RQ2 (Error propagation) How much downstream performance is affected by errors propagated from earlier stages, and which stage is the most consequential bottleneck?

RQ3 (Isolated agent performance) If a sub-task has low end-to-end performance, is the poor result caused by earlier errors or by the agent's own limitations? In other words, does providing gold upstream input improve performance?

RQ4 (Model capability) How does an open-weight model perform compared to a commercial model across the sub-tasks?

RQ5 (Prompting strategy) How does the pipeline perform on zero-shot prompting in comparison to few-shot ones?

Contributions

This thesis makes two main contributions:

1. **Pipeline architecture and prompt engineering.** This work presents a five-node multi-agent system that covers the entire legal argument mining pipeline.

2. **A diagnostic evaluation methodology.** The pipeline is evaluated in three ways: end-to-end, upstream conditioning, and an isolated setting. As a result, it becomes possible to measure the cost of error propagation directly and to distinguish between errors caused by propagation and those due to the task's inherent difficulty.

Chapter 2

Background and Related Work

2.1 Foundations of Natural Language Processing and Large Language Models

2.1.1 Classical NLP and Statistical Language Models

Classical NLP is based on statistical models implemented using explicit text representations. Before doing any math, unstructured, raw text must be converted into a structured, clean representation that a computer can process.

Step 1: Text Preprocessing

This process includes reducing text variation by lowercasing to ensure that “The” and “the” are treated as the same word and noise removal to strip out irrelevant elements such as HTML tags, punctuation, or special characters, based on the task.

Step 2: Tokenization

This is the process of breaking a text into chunks (tokens). While classical systems used word-level tokenization (splitting by spaces), modern systems use sub-word tokenization (e.g., splitting "unbelievable" into "un-", "believe", and "-able"), called Byte Pair Encoding (BPE), to handle the out-of-vocabulary problem.

Step 3: Normalization

This is the process of converting tokens into a format that a model can process more effectively. Two of the most common normalization techniques in NLP are stemming and lemmatization, which aim to break words down to their root.

- Stemming: It is a rule-based process that cuts prefixes or suffixes of words to reduce them to a stem.

- Lemmatization: It is a process that reduces a word to its lemma, which is its dictionary form.

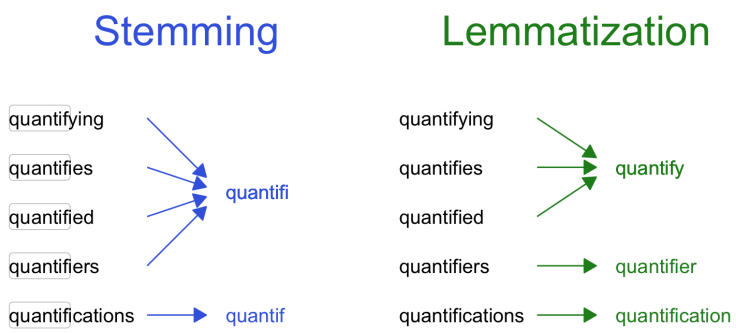

Figure 2.1: stemming and Lemmatization

Step 4: Part-of-Speech (POS) tagging

This is a fundamental step in classical NLP that assigns grammatical categories to tokens, causes improvement in normalization (e.g., POS-aware lemmatization), and creates linguistic features used in statistical feature-based models like Bag-of-Words and TF-IDF classifiers.

Step 5: Feature Extraction

This is the process of converting tokens into numbers that a computer can understand. There are three classical methods for converting tokens into numerical features, including:

1. Bag-of-Words (BoW):

   Creates a vocabulary of all unique words in the dataset. Each document is represented as a vector, where each entity corresponds to a word in the vocabulary and the frequency of that word in the document. This method ignores word order and grammatical structure; therefore, sentences such as "The cat ate the mouse" and "The mouse ate the cat" are treated as identical.

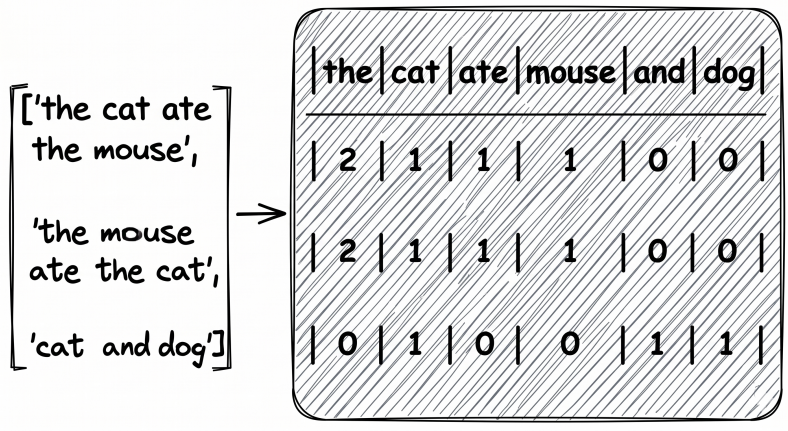

Figure 2.2: Bag-of-Words visualization

2. n-grams:

   Behave exactly like a Bag-of-Words model, but n-grams consider each n consecutive tokens as a unique word to preserve word orders (like "not good" vs "good"). However, as n increases, the number of unique sequences grows exponentially, leading to massive and sparse vectors.

   - $n = 1$ (Unigram): Looks at a single word (Bag-of-Words).
   - $n = 2$ (Bigram): Looks at pairs of consecutive words.
   - $n = 3$ (Trigram): Looks at triplets of consecutive words.

3. Term Frequency-Inverse Document Frequency (TF-IDF):

   TF-IDF is defined to capture lexical meaning in the text by computing an importance weight score to balance the occurrence of a word in a single document compared with the entire collection of documents (the corpus).

   TF-IDF is the product of two separate metrics:

   - Term Frequency (TF)

     Measures the occurrence of a term (t) in a specific document (d) to measure its importance in the document.

     $$\mathrm{TF}(t, d) = \frac{\text{Count of } t \text{ in } d}{\text{Total number of words in } d}$$

   - Inverse Document Frequency (IDF)

Measures the occurrence of a term (t) across the entire corpus (D). If a word appears in every document in the dataset, its informational value drops to zero.

$$\text{IDF}(t, D) = \log\left(\frac{\text{Total number of documents in corpus } |D|}{\text{Number of documents containing term } t}\right)$$

Since the word appearance in 10 documents vs 100 documents, with a corpus of 1 million documents, make a huge ratio difference, use log.

- The Combined Formula

  The product of these two components gives the final score:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

Step 6: Statistical language models

In statistical natural language processing, the earliest idea was to model language through probability by estimating the likelihood of a word appearing in a sequence based on its preceding words. This led to the development of **n-gram** language models, which estimate the probability of a word sequence using the Markov assumption, where each word depends only on the previous $n-1$ words:

$$P(w_t \mid w_{1:t-1}) \approx P(w_t \mid w_{t-n+1:t-1})$$

Example: For a unigram language model:

$$P(\text{I love NLP}) = P(\text{I}) \cdot P(\text{love} \mid \text{I}) \cdot P(\text{NLP} \mid \text{love})$$

However, n-gram language models have their limitations, such as being unable to capture semantic meaning; poor generalization, assigning zero probability for unseen word combinations; and, as the value of n increases, the number of possible word combinations grows exponentially, resulting in high computational cost and sparse representations.

In parallel, another branch of statistical NLP focuses on **text classification** tasks, such as identifying spam emails, sentiment polarity, or document topics, rather than sequential next-word prediction. Traditional statistical classifiers use fixed feature representations taken from text, such as Bag-of-Words (BoW) or TF-IDF. Among these, **Naive Bayes** and **Logistic Regression** act as foundational

baseline approaches, representing generative and discriminative modeling paradigms, respectively.

1. Naive Bayes (Generative Approach)

One of the earliest statistical classifiers in NLP was **Naive Bayes**, which is based on Bayes' theorem. As a generative model, it estimates the joint probability of the text features and the class label. To make computation feasible on high-dimensional text data, the model relies on a strict **conditional independence assumption**—the assumption that the occurrence of every word feature is entirely independent of any other word given the class label.

For a document d represented by a set of features (words) $w_1, w_2, \ldots, w_n$, this independent assumption allows us to express the likelihood $P(d|c)$ as the product of individual probabilities:

$$P(w_1, w_2, \ldots, w_n \mid c) = \prod_{i=1}^{n} P(w_i \mid c)$$

This assumption makes Naive Bayes computationally efficient for high-dimensional text classification tasks. However, this assumption is unrealistic for natural language, since words are highly dependent in reality.

2. Logistic Regression (Discriminative Approach)

As an alternative, **Logistic Regression** represents a discriminative approach that directly models the decision boundary between classes rather than the data-generating distribution. The model assigns a weight (β_i) to every input feature x_i, such as word counts or TF-IDF values. Given a feature vector $x = (x_1, x_2, \ldots, x_n)$, the model computes a linear combination score (z):

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

where β_0 represents the bias term. To map this raw score from a real line ($-\infty$ to $+\infty$) into a valid probability score between 0 and 1, Logistic Regression passes z through the Sigmoid function:

$$P(y = 1 \mid x) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

For multi-class classification tasks, this mapping is generalized using the Softmax function.

Limitations of Traditional Statistical Classifiers

When used with traditional vocabulary-based features, such as BoW or TF-IDF, both Naive Bayes and Logistic Regression have systemic limitations. Because these feature representations ignore sequential structure, the resulting classification systems can not capture word order or contextual nuances. For example, consider the following sentences:

- **Sentence A:** "The movie was not terrible, it was actually good." (Positive)
- **Sentence B:** "The movie was not good, it was actually terrible." (Negative)

When mapped to standard frequency vectors, both sentences contain same features. Consequently, the classifiers treat them identically, completely missing the contextual meaning. Additionally, it can not handle Out-of-Vocabulary (OOV) words. If a document contains a synonym or a new slang word that did not appear during the training phase. This limitation led us to move toward deep learning approaches, which automatically learn meaningful representations from data.

2.1.2 Transition from Statistical NLP to Deep Learning

Despite classical NLP that required domain experts to manually engineer features, deep learning introduced end-to-end representation learning, where the model automatically learns the features directly from the raw text data.

Additionally, deep learning scales well with data and computation. By using self-supervised learning, deep neural networks can train on massive, raw datasets (like the entire internet) to build a foundational understanding of language, which can then be fine-tuned for specific tasks.

Embeddings

One of the breakthroughs in deep learning for natural language processing was introducing distributed word representations, known as word embeddings. Embeddings represent words as dense, numeric vectors in a continuous space that captures both semantic relationships and contextual information. This means that words with similar meanings are positioned close to each other in this multidimensional space.

The distance and direction between these vectors encode the degree of similarity between words. The most famous example of this geometry is the relation between the words "woman", "man", "queen" and "king":

$$\text{Vector("King")} - \text{Vector("Man")} + \text{Vector("Woman")} \approx \text{Vector("Queen")}$$

There are various models for generating word Embeddings. The two most popular models are **Word2Vec** and **GloVe**.

1. Word2Vec

   Word2Vec was developed by Google in 2013. This approach is built based on two distinct architectures: Continuous Bag-of-Words (CBOW) and Skip-gram.

   - Continuous Bag-of-Words (CBOW) Architecture

     The core idea of this architecture is to predict a specific target word based on its surrounding context words. To achieve this, the model maps the high-dimensional one-hot encodings of the context words into their corresponding dense vector representations.

     Imagine a sentence like "The chef cooks delicious food." If the target word is "cooks", the model looks at the surrounding words ("The", "chef", "delicious", "food"). It takes the high-dimensional one-hot encodings of these context words, maps them to their dense vector representations, and averages them together to form a single hidden context layer. The model then uses this averaged context vector to calculate a probability distribution over the vocabulary and predict that "cooks" is the most likely target word.

   - Skip-gram Architecture

     This architecture is exact opposite of CBOW. It predict the surrounding context words given a single target word.

     Using the same example, if the input target word is "cooks", the Skip-gram architecture uses this single word's vector to independently predict the probability of finding "the", "chef", "delicious", and "food" within its neighborhood.

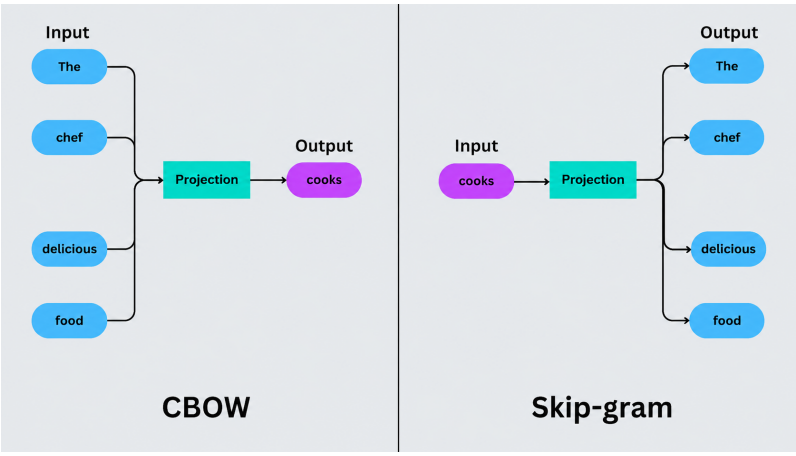

Figure 2.3: Word2Vec Architecture Visualization

2. GloVe

   Global Vectors for Word Representation (GloVe) was developed by Stanford University in 2014. This approach is built based on co-occurrence statistics to create word vectors, meaning analyzing how often words appear together across the entire corpus.

   GloVe builds a massive co-occurrence matrix X from the entire corpus, where every entity X_{ij} represents how many times a word i appears in the context of another word j.

   However, GloVe, instead of looking at raw co-occurrence counts, focuses on the ratios of co-occurrence probabilities. For example, for words "ice" and "steam", the word "solid" has a high co-occurrence with "ice" but low with "steam". The word "gas" has a high co-occurrence with "steam" but low with "ice". A neutral word like water has a high co-occurrence with both. By analyzing these relations, the model learns precise semantic directions.

   GloVe solves this by training word vectors so that the dot product of any two word vectors equals the logarithm of their global co-occurrence probability:

$$\mathbf{w}_i^\top \tilde{\mathbf{w}}_j \approx \log(X_{ij})$$

   To prevent highly frequent words (like "the" and "is") from dominating the space, GloVe introduces a weighting function that reduces the influence of massive counts while still giving meaningful weight to rare word pairs.

The main difference is that Word2Vec learns from local context, updating word vectors by predicting nearby words step by step, while GloVe learns from global information by using global word co-occurrence counts from the whole dataset to directly build the embeddings.

2.1.3 Neural Language Models and Pre-training Paradigms

In modern NLP, embeddings solve the representation problem by mapping tokens into dense numeric vectors; the central challenge in natural language processing shifts from word representation to learning meaningful structure. At this stage, the key problem is no longer how to encode words, but how to enable a model to infer syntactic structure, semantic relationships, and long-range dependencies directly from raw text.

Modern NLP addresses this through a self-supervised learning principle, which means learning language by predicting missing or future tokens. Rather than requiring manually labeled data, models are trained on a massive corpus, using prediction tasks that force them to learn general linguistic patterns before using them for specific downstream tasks. This worked because the model predicts successfully only if the model has learned useful internal representations of grammar, meaning, and discourse structure.

Two main formulations for these modern architectures are the following:

- **Autoregressive (Causal) Language Modeling:** The model predicts each token based on preceding tokens. If a sequence is $\mathbf{x} = (x_1, x_2, \ldots, x_T)$, the model predicts x_t by using the probability chain rule:

$$\mathrm{p}(\mathbf{x}) = \prod_{t=1}^{T} \mathrm{p}(x_t \mid x_1, \ldots, x_{t-1})$$

  It forms the decoder-only models, *e.g.*, GPT [12]. Since the model never accesses future context, it learns representations that are naturally suitable for coherent and fluent text generation. So GPT is literally implementing the following:

$$\mathrm{p}(\text{response} \mid \text{prompt}) = \prod_{t=N+1}^{T} \mathrm{p}(x_t \mid x_1, x_2, \ldots, x_{t-1})$$

- **Masked Language Modeling:** The input of this model is a sequence of tokens with a hidden subset. The model is trained to recover the hidden

subset using both left and right context. This bidirectional conditioning allows the model to build better contextual representations of each token, making it particularly effective for understanding tasks. This objective is commonly used in encoder-only architectures, *e.g.*, BERT.

If a sequence is $\mathbf{x} = (x_1, x_2, \ldots, x_T)$, the model predicts $\mathbf{x}_M$ as follows:

$$\mathrm{p}(\mathbf{x}_M \mid \tilde{\mathbf{x}}) = \prod_{m \in M} \mathrm{p}(x_m \mid \tilde{\mathbf{x}})$$

Where, $\tilde{\mathbf{x}}$ is the corrupted sequence where the chosen tokens have been replaced by a special [MASK] token, and M is the set of indices representing the positions of the masked tokens.

The masked language model looks at the entire corrupted sequence $\tilde{\mathbf{x}}$ all at once (bidirectionally) to reconstruct the missing tokens.

After pre-training, models learn general linguistic patterns and can be adapted to specific downstream tasks through fine-tuning or instruction-based alignment to follow specific prompts for applications such as classification, question answering, and reasoning systems.

2.2 Neural Architectures for Language Models

Section 2.1.3 describes how modern language models learn natural language structure via next-token prediction. This section focuses on the development path leading to large language models, particularly on autoregressive language modeling.

2.2.1 Feedforward Neural Language Models

A feedforward neural network is a multi-layer architecture in which information flows only in one way, from the input layer through one or more hidden layers to the output layer, without recurrent or feedback connections.

In feedforward neural language models, the input sentence is first tokenized, and each token is converted into a continuous vector representation (word embedding). Since these models operate on a fixed context window, a sequence of n tokens is represented as the concatenation of their corresponding embeddings into a single input vector x.

The input vector is then passed through one or more hidden layers, where the model learns to capture non-linear dependencies between input features. Each hidden layer typically performs two main operations:

- **Linear transformation:** the input is projected using a weight matrix W and a bias vector b:

$$z = Wx + b$$

- **Non-linear activation:** to introduce non-linearity and increase the expressiveness of the network, an activation function such as ReLU is applied:

$$a = \text{ReLU}(z) = \max(0, z)$$

Without non-linear activation functions, the combination of multiple layers would still result in a linear transformation, limiting the model's capacity to learn complex patterns.

Finally, the output layer produces a probability distribution over the vocabulary using the softmax function:

$$P(w_t \mid x) = \text{softmax}(hW_o + b_o)$$

Where h is the final hidden representation and W_o, b_o are the output layer parameters. This distribution represents the model's prediction for the next token given the input context.

2.2.2 Recurrent Neural Networks

Recurrent neural networks (RNNs) remove the fixed-window constraint by processing the sequence one token at a time while maintaining a hidden state h_t that summarizes all tokens seen so far:

$$h_t = \sigma(W_{hh}\, h_{t-1} + W_{xh}\, x_t + b_h)\,, \qquad p(x_t \mid x_{<t}) = \text{softmax}(W_{ho}\, h_t + b_o)\,.$$

Since h_t depends on h_{t-1}, information can propagate across the whole sequence, and the same parameters are reused at every time step. This recurrence makes RNNs a natural fit for the autoregressive objective, where h_t can be interpreted as a summary of the conditioning context $x_{<t}$.

In practice, training RNNs over long sequences causes vanishing and exploding gradients, which happen during backpropagation through many time steps. As gradients are repeatedly multiplied across the sequence, they can prevent learning of long-range dependencies. As a result, standard RNNs struggle to capture relationships between distant tokens.

Gated variants, such as the Long Short-Term Memory network (LSTM) and the Gated Recurrent Unit (GRU), address this limitation by introducing gating mechanisms that control the flow of information through the hidden state. These gates regulate what information is retained, updated, or forgotten, allowing gradients to propagate more effectively over long sequences and sequentially improving long-term dependency modeling.

Two limitations persist even with gating. First, computation is inherently *sequential*, meaning the state at a position t can not be computed before the state at $t-1$, so training can not be implemented in parallel across the time dimension, which causes the problem for long documents and large corpora. Second, all information must still pass through a single hidden-state bottleneck, so preserving long-range dependencies remains difficult. The idea of handling dependencies between arbitrarily distant tokens, without forcing all information through one sequential channel, motivates the attention mechanism.

2.2.3 The Attention Mechanism

Attention was first introduced to solve the limitation of recurrent encoder-decoder models for sequence-to-sequence tasks such as machine translation. In these models, the entire source sequence had to be compressed into a single fixed-size vector before decoding. This created an information bottleneck so that many details of the input could be lost.

Attention replaces that single summary with a mechanism that allows the decoder to look at all encoder states directly and, at each step, decides which parts of the input are most relevant and focuses more on them.

The mechanism scores the relevance between the current decoder state and each encoder state. These scores are then normalized into weights α_{ij} that sum to one and form a final context vector as their weighted combination:

$$c_i = \sum_{j=1}^{T} \alpha_{ij}\, h_j, \qquad \alpha_{ij} = \frac{\exp\Big(\text{score}(s_i, h_j)\Big)}{\sum_{k=1}^{T} \exp\Big(\text{score}(s_i, h_k)\Big)},$$

Where s_i is the current decoder state and h_j are the context representations.

This is a form of **soft alignment** that the model learns to focus on, which positions instead of forcing the model to compress everything into a vector.

The attention mechanism also creates a direct connection between any two positions in a sequence, independently of their distance. This removes the long-range limitation of recurrent models and avoids the fixed-size bottleneck of earlier architectures. Another key advantage is that these interactions do not depend on step-by-step recurrence, so they can be computed in parallel.

This raises a key question that changed the direction of the field: if attention alone can model long-range dependencies, is recurrence still necessary?

2.2.4 The Transformer Architecture

The Transformer answered that question by removing recurrence completely and building the model from attention. Its main idea is **self-attention**, where every token in a sequence looks to every other token (including itself) to build a context-aware representation. To do this, each token is mapped into three vectors: a query, a key, and a value. The queries are compared against all keys to produce attention weights (how relevant they are), which are then used to combine the values:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V,$$

The scaling factor $\sqrt{d_k}$ keeps the dot products in a numerically stable range. In addition, several attention operations are run in parallel as **multi-head attention**, allowing different heads to capture different types of relations, such as syntactic agreement or co-reference.

Since self-attention does not process tokens in a fixed order of their inputs, the architecture uses **positional encodings** in the token representations to make sequence order available to the model. Each layer also contains a position-wise feedforward sub-layer for additional capacity, and the network is made deep and stable through two components: **residual connections**, which add each sub-layer's input to its output ($x + F(x)$) so that gradients flow through deep stacks without degrading; and **layer normalization**, which stabilizes the scale of activations.

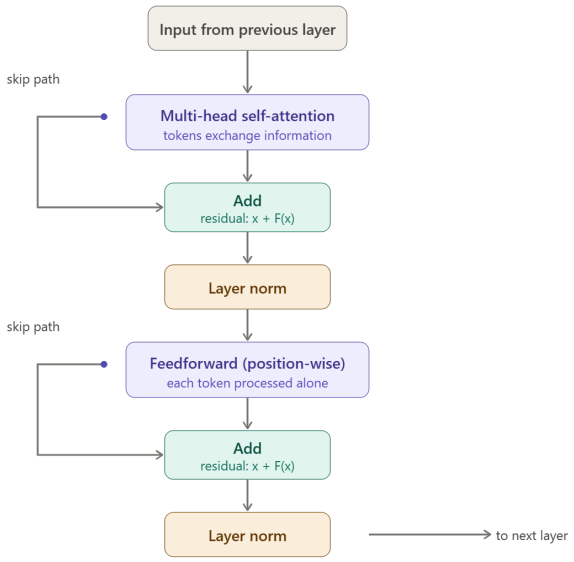

Figure 2.4: Transformer block

The Transformer can also be used in different ways depending on how attention is restricted. If each token is only allowed to attend to previous tokens, the model becomes autoregressive, as in GPT. If tokens can attend in both directions, it becomes a bidirectional encoder model, as in BERT [4]. An encoder-decoder setup combines both ideas.

A key advantage of this design is that all tokens can be processed in parallel, unlike RNNs. In addition, any token can directly interact with any other token in a single step, no matter how far apart they are. This makes training much more efficient and scalable, which is one of the main reasons Transformers work well on very large datasets and form the basis of modern large language models [13].

2.2.5 Scaling Laws and Large Language Models

After the Transformer, large-scale training became possible, and the question of "How does performance improve as models become larger and are trained on more data?" became central. Kaplan et al. showed that the test loss of a Transformer language model decreases as a smooth **power law** in three factors: the number of model parameters N, the size of the training dataset D, and the amount of training compute C, as long as none of the three is the limiting bottleneck [14]. These relationships indicate that performance depends more on overall scale than on architectural details.

Later research by Hoffmann et al. whose compute-optimal analysis (the **Chinchilla** result) showed that for a fixed compute budget, model size and the number

of training tokens should be scaled in roughly equal proportion, on the order of twenty training tokens per parameter, rather than growing the parameter count alone [15]. Empirically, their 70B-parameter Chinchilla model, trained on more data, outperformed the much larger 280B-parameter Gopher under the same compute budget [15]. Scaling laws made model development more predictable: the loss of a larger model can be estimated before committing resources to train it, enabling the development of even larger systems.

Transformer-based models scaled to billions of parameters and trained on web-scale corpora are referred to as **large language models (LLMs)** [12]. They demonstrated abilities that were not present in smaller models, including **in-context learning**, where the model learns from examples provided in the prompt without any parameter update [12]; instruction-following [16]; and multi-step reasoning, such as chain-of-thought prompting [17]. Some researchers describe these as **emergent abilities**, although there is debate about whether they appear suddenly or are influenced by the chosen evaluation criteria[18].

In-context learning and emergent reasoning enable an LLM to be treated not only as a text predictor but also as a general-purpose reasoning component that can follow instructions, decompose problems, and act on intermediate results. This capacity is the foundation for the agentic and collaborative multi-agent systems investigated in this thesis and motivates their application to legal argument mining, as developed in the following sections.

2.3 Argumentation Mining

Argumentation Mining (AM) is a research field in Natural Language Processing (NLP) that focuses on automatically identifying, extracting, and analyzing argumentative structures from natural language text [3].

Argument Mining (AM) aims to understand how arguments are constructed and how conclusions are justified through supporting evidence and reasoning [19]. This process includes identifying which parts of a text are argumentative, what role each argumentative component plays (e.g., premise or claim), and how these components relate to one another (support or attack relation) [3, 20].

The field is based on argumentation theory. One of the important works in this field is Toulmin's work, which described arguments as claims supported by evidence (data) through a warrant [21]. Toulmin claims that human arguing does not match

mathematical reasoning in real life. Therefore, he proposed a practical model of how arguments are structured. For example:

- The Claim (final state of argument, like conclusion):

  "My grandfather should wear a hearing aid."

- The Evidence/Data (The facts and observed data which are the starting points):

  "Over 70% of people over the age of 65 have documented hearing difficulties."

- The Warrant (logical bridge that connects the data to the claim):

  "A hearing aid helps most people hear better."

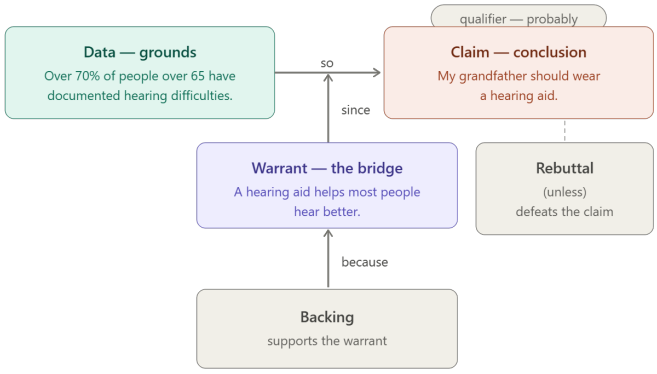

Figure 2.5: Toulmin's model

Another important contribution comes from Walton et al. [22], who introduced **argumentation schemes**, which are reasoning patterns that explain how premises support claims. Walton identified more than 60 of these schemes. These theoretical models provide the foundation for computational argument mining systems.

From a methodological perspective, argument mining is divided into sub-tasks as follows [23, 19]:

1. **argumentative component detection**, which identifies argumentative from non-argumentative text
2. **component classification**, which assigns roles to argumentative units, such as premise or conclusion
3. **structure or relation prediction**, which identifies support and attack relations between components to reconstruct the overall argument structure.

Despite significant advances, argument mining remains challenging due to the ambiguity and implicit nature of arguments. One major difficulty is *segmentation*, as argumentative boundaries do not always align with sentence boundaries; a single sentence may contain both a premise and a claim [7]. A second challenge is *relation prediction*, which requires contextual or domain-specific reasoning. These challenges become more important in specialized domains such as law, where legal reasoning involves implicit assumptions, reliance on precedent, and highly technical language.

2.3.1 Legal Argument Mining

Legal argument mining (LAM) is a subfield of argument mining (AM) that focuses on extracting argumentative structures from legal texts such as court decisions, statutes, contracts, and legal briefs [1]. Legal texts differ from general domain arguments; they use highly formalized language, domain-specific vocabulary, and reasoning based on legal rules, precedents, and evidential interpretation [5].

In legal reasoning, arguments are constructed from legal norms and factual circumstances, which may include implicit assumptions and references to prior case law. As a result, legal argument mining aims to identify claims and premises but also to capture legally interdependent reasoning structures, such as issue–rule–application–conclusion patterns and precedent-based justification [24]. This makes the task more complex than general argument mining since it requires understanding both linguistic structure and domain-specific legal knowledge.

Existing research in legal NLP has focused on argument mining in judicial decisions and legal case texts, showing that legal arguments can be computationally modeled; however, some challenges remain due to the complexity of legal reasoning and the frequent absence of explicitly stated argumentative relations. Recent neural and transformer-based models have improved performance in identifying argumentative components in legal documents; however, challenges remain in capturing implicit reasoning and maintaining globally coherent argument structures across long legal texts [3, 19].

In the literature, legal argument mining is usually decomposed into three fundamental, standard sub-tasks that form the core pipeline of most architectures:

- **Argument Component Detection:** Detecting argumentative sentences from procedural, administrative, and descriptive passages within a judicial text.
- **Argument Component Classification:** Assigning each argumentative sentence its role. The standard distinction is between *conclusions*, the final

assertions or claims made by the court, and *premises*, the supporting evidence, reasoning, or legal grounds.

- **Relation and Link Prediction:** Determining the structural relationships between the argumentative components, such as premise supports, or attacks on the conclusion, or premise supports or attacks on each other, resulting in a directed argument graph.

These three are the traditional baselines in legal argument mining; however, **the Demosthenes corpus** introduces deeper semantic layers that depend on its underlying annotation frameworks, as discussed in Chapter 3.

2.3.2 Computational Approaches to Argumentation Mining

Over the last two decades, computational research on argument mining has developed significantly, with improvements in natural language processing, including rule-based systems, deep learning models, and, more recently, large language models (LLMs) and agent-based architectures. This progress shows a shift from highly structured, manually designed systems to more data-driven, practical approaches.

Rule-based and Early Linguistic Approaches

Early work in argument mining was mostly influenced by computational linguistics and rule-based natural language processing. Argumentative structures were detected using manually designed linguistic rules, syntactic patterns, and domain-specific heuristics. To detect argumentative relations, systems relied on cue phrases (e.g., "therefore" and “because”), discourse markers, and manual grammar definitions.

Early computational approaches to argument mining modeled argument structures using rule-based grammars and manually designed textual features to identify claims, premises, and relations between argumentative components [1]. Similarly, early research in the legal domain relied on symbolic representations and handcrafted features to extract argumentative components from judicial texts [25].

Although these methods produced interpretable structures, they depended on domain expertise and struggled with linguistic variation, implicit reasoning, and cross-domain generalization.

Classical Machine Learning Approaches

The next step in argument mining was supervised machine learning algorithms. Instead of manually defining rules, these approaches treated argument mining as a classification problem, classifying sentences or discourse segments [3, 19]. Feature engineering remained vital, but features were now learned or statistically computed from data.

Common models were support vector machines (SVMs), conditional random fields (CRFs), and logistic regression classifiers [26, 7]. These systems were used for sub-tasks such as argument detection, classification of premises and claims, and prediction of relations between argumentative units.

One of the most significant limitations of these approaches was their dependency on feature engineering and domain-specific preprocessing. While robustness improved compared to rule-based systems, performance declined on data with different writing styles or argumentative conventions. According to comprehensive surveys, generalization is a significant challenge of this stage [3, 19].

Neural and Deep Learning Approaches

The development of deep learning has significantly advanced argument mining research. Neural architectures learned distributed text representations without requiring manual feature engineering. Early neural models used convolutional neural networks (CNNs) and recurrent neural networks (RNNs), especially long short-term memory (LSTM) networks, to capture sequential and contextual information in arguments [6, 27].

However, transformer-based architectures became dominant with the introduction of models such as BERT. Argument mining systems improve their performance by using pre-trained language models. Fine-tuning pre-trained transformers became the standard approach for tasks such as argumentative classification and relation extraction [4, 6].

Neural models are effective in structured argument analysis, especially in domains such as persuasive essays where argumentative components and relations are well-defined [7]. Transformer-based models have also been successfully used in legal argument mining tasks, where contextual understanding is essential due to implicit reasoning and domain-specific language [5, 2].

However, some challenges remain. Deep learning models still struggle to capture

global argumentative coherence and reasoning across multi-step structures, especially when arguments rely on external knowledge [28]. This motivates research into hybrid reasoning approaches and more structured architectures [28, 27].

Large Language Models and In-Context Learning

Large language models (LLMs) have introduced a new paradigm in argument mining. Instead of training a classifier for a specific task, an LLM can be prompted to perform argument mining by providing the task definition and, if needed, a few labeled examples within its context window. This approach is called in-context learning [29].

There are two main reasons why this approach works well in the legal field. The first is **the length of the document**. Legal judgments are long, and their argumentative chain spans many paragraphs. Encoder models like BERT can only handle a few hundred tokens at a time, so they have to break legal texts into smaller pieces, which means they lose the context of the argument. In contrast, LLMs can process much larger sections at once, making it possible to consider an entire line of reasoning as a whole. The second reason is **data scarcity**. Labeling legal arguments requires expert knowledge; it is a slow and costly process. As a result, available datasets are few, usually limited to a single area of law, and do not provide enough examples to effectively fine-tune an encoder model [10, 30]. LLMs address this problem because they can be used with little or no specialized training.

LLMs have been used throughout the argument mining pipeline for general applications, such as detecting argumentative components, classifying support and attack relations between argument pairs, and reconstructing argument graphs using multi-stage prompting [31]. In legal contexts, LLMs are applied to detect argumentative sentences in the ECHR corpus [32, 30], are used to generate case-based arguments and counterarguments based on specific argumentation schemes [33], and are prompted to identify premises and conclusions based on the structure of legal syllogisms, showing that organizing reasoning according to legal principles can improve the accuracy and reliability of legal judgment predictions [34]. The most relevant work for this thesis examines LLMs with few-shot prompting on three classification tasks in the Demosthenes corpus, which is also used in this study [35].

The evidence on whether LLMs outperform supervised models is different. Evaluation of GPT models for argumentative classification on the ECHR corpus shows results below a legal-BERT baseline [36], and similar findings have also been reported on Demosthenes Alfieri et al. [35]. Recent surveys in the field agree that LLMs

offer strong generalization and remove the fine-tuning bottleneck, but complex legal reasoning remains a challenge, and domain-specific encoders remain competitive with sufficient supervision [29]. In practice, this means there is a trade-off rather than a full replacement. LLMs are useful when data is limited, but this comes at a cost to task accuracy.

They are also highly sensitive to prompt design, so their performance depends on how the prompt is worded and which demonstrations are chosen [35]. In addition, these models are used for tasks that produce graphs; they can not offer structural reliability [29].

Reasoning Strategies for a Single Agent

The following techniques help a single model think more effectively.

1. **Reasoning and acting (ReAct).** This approach combines reasoning (or chain-of-thought) with actions. It allows a model to use external tools or take intermediate steps during inference, then keep reasoning based on what it observes [37] .

2. **Self-reflection (Reflexion).** This approach proposes a form of verbal reinforcement learning in which an agent reviews its own natural-language output, stores the feedback in episodic memory, and uses it to guide future attempts. The agent improves over time without changing its parameters. In this approach, a single agent revises itself through multiple iterations, rather than relying on a group of agents [38] .

3. **Self-consistency.** This approach samples several reasoning paths and selects the most frequent final solution via majority voting [39].

Multi-Agent Systems: Decomposition and Interaction

A **multi-agent system** (MAS) is a setup in which several LLMs, each with a distinct role, prompt, and often a unique view of the shared state, cooperate to solve a task together [9]. The main questions of MAS are not about how the model reasons, but about how the problem is divided and how the parts communicate.

1. **Decomposition into specialized sub-tasks.** The main reason to use MAS is to break down a complex problem into smaller sub-tasks. Each sub-task is assigned to an agent with a specific prompt [9]. This approach has three main benefits: First, each agent can focus on a single decision point. Second,

intermediate results can be checked; if something goes wrong, it is easier to find where the problem occurred. Third, agents can be improved or replaced independently, which is useful for tasks that are already organized hierarchically, such as legal reasoning.

2. **Interaction paradigms.** MAS mainly differ in how agents interact with each other. In a **pipeline** (or cascade), agents are arranged sequentially, with each using the output of the previous agent, and coordinate through the shared state. In **debate** architectures, several agents propose their own answers and then critique each other's responses over several rounds, reaching agreement through discussion rather than by ordering. In **hierarchical** (orchestrator–worker) designs, a supervisor agent decomposes the task into sub-tasks, assigns them to worker agents, and then combines their results. In **blackboard** architectures, agents use shared memory to read and write information, coordinating by updating the shared state rather than addressing each other directly [9].

Multi-Agent Systems in the Legal Domain

Multi-agent designs have recently been used in legal NLP, reflecting how legal work is divided in practice, as in Sadowski et al. [40], which decomposes legal analysis into verifiable intermediate steps, improving the transparency and consistency of results. Another work, LegalGraphRAG, assigns specific roles to a researcher, an auditor, and an adjudicator to support legal reasoning based on retrieved evidence [41]. Additionally, L-MARS orchestrates agents for decomposition, retrieval, and verification to improve the reliability of legal question answering [42]. Systems such as LawLuo and ChatLaw show that multi-agent collaboration can simulate legal consultation workflows and reduce hallucinations [20, 43]. And finally, AgentsCourt models adjudication as an interaction between agents playing different roles [44].

2.4 Prompting Strategies for Complex NLP Tasks

Since our pipeline is based on prompting rather than fine-tuning, its accuracy depends on how each sub-task is presented to the model. This section reviews the prompting techniques used by individual nodes, as well as inference-time controls such as sampling temperature and majority voting.

2.4.1 Chain-of-Thought Prompting

Wei et al. [17] demonstrated that prompting LLMs with intermediate reasoning steps significantly improves performance on complex tasks. Chain of Thought (CoT) is a prompt engineering technique that encourages LLMs to generate step-by-step reasoning before providing the final answer. Shifting the objective of the model from predicting the final answer to predicting the reasoning path toward the answer.

2.4.2 Self-Consistency and Majority Voting

Self-Consistency, proposed by Wang et al. [39], is a prompting technique that involves generating multiple reasoning paths and selecting the most frequent answer. One decision-making method is **Majority Voting**. After generating a pool of diverse outputs, the algorithm selects the final output based on the one that appears most frequently among all generated paths.

2.4.3 Few-Shot Prompting and Domain Matching

Brown et al. [12] showed that LLMs can learn to perform a task from a small number of demonstrations presented directly in the prompt, known as *in-context learning*, without any parameter updates. However, the selection of demonstrations is not neutral. Min et al. [45] found that examples from the same domain as the target input perform better than generic examples. Since CJEU State-aid judgments use specialized terminology, the demonstrations in this work are derived from the same legal material. This choice helps align the model's in-context expectations with the reasoning patterns and language found in the judgments.

2.4.4 Decomposed Prompting

Khot et al. [46] showed that decomposing a complex task into simpler sub-tasks with a specialized prompt can improve performance. Instead of asking an LLM to solve a massive, complex problem all at once, which often leads to reasoning errors or hallucinations, decomposed prompting introduces a decomposer (often a specific LLM prompt or agent) that breaks the main problem into sequential or parallel sub-problems. These sub-problems are then solved independently, sometimes using specialized prompts, different models, or external tools, before their results are integrated to form the final output.

Chapter 3

Materials and Methods

3.1 Dataset

3.1.1 The Demosthenes Corpus

This thesis evaluates a collaborative multi-agent system for legal argument mining using the **Demosthenes** dataset [10]. The dataset contains manually annotated legal documents with detailed argumentative structures, making it a suitable choice to evaluate whether multiple agents can collaboratively identify and classify legal arguments. The system is designed to model how specialized agents collaborate in legal reasoning to reconstruct argumentative structures.

The Demosthenes dataset contains decisions from the Court of Justice of the European Union (CJEU) about fiscal state aid law. It offers hierarchical annotations of argumentative components, premise types, legal reasoning schemes, and their relationships, making it a strong benchmark for evaluating complex legal argumentation tasks [10, 47].

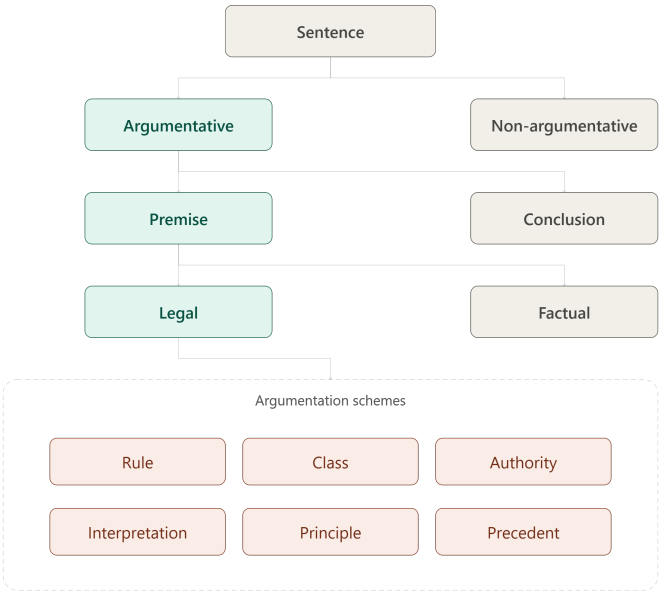

Figure 3.1: Hierarchical Annotation of Dataset

3.1.2 Document Structure

Each CJEU decision is organized into distinct sections. This annotation looks specifically at the **Findings of the Courts** sections, which detail all argumentative steps leading to the final ruling [10]. The following sections are not included in the annotation:

- Preamble: Party information, composition of the Court, and references to the appealed judgment.
- Case background: Facts and procedural history before the General Court.
- Judgment under appeal: Assessment of the General Court at first instance.
- Arguments of the parties: Submissions of the appellant, respondent, and interveners.
- Costs and Ruling: Cost attribution and final orders, which merely repeat the conclusion of each argument chain.

Annotation Schema

The dataset includes hierarchical annotations as follows:

Level 1 — Argumentative elements. In the Findings of the Court section, each argumentative sentence is labeled as a premise (`<prem>`) if it represents evidence, legal rules, or interpretations that support a ruling. It is labeled as a conclusion (`<conc>`) if it represents the court's final decision on a plea or the whole case.

Level 2 — Premise type. Each premise is classified as follows:

- `Type="F"` (Factual): Denotes factual situations, events, or procedural aspects of the case.
- `Type="L"` (Legal): Indicates legal content, including rules, precedents, and interpretations of applicable laws and principles.
- `Type="L|F"` (Both): Integrates legal and factual aspects within a single sentence.

Level 3 — Argumentation scheme. Each legal premise receives one or more argumentation scheme labels, as defined in Table 3.1, when different types of reasoning are present.

Table 3.1: Argumentation scheme labels and EU law signals

Label	Walton Scheme	EU Law Signal
Prec	Arg. from Precedent	References a prior ECJ case by name or citation number
Rule	Arg. from Rules	Cites a treaty article, directive, or Steel Aid Code provision
Itpr	Arg. from Interpretation	Interprets scope or meaning of a legal text or concept
Aut	Arg. from Authority	References an Advocate General opinion or expert body
Class	Arg. from Classification	Classifies a situation under a legal category
Princ	Arg. from Principle	Invokes legal certainty, proportionality, or legitimate expectation

The original study indicates that the `Itpr` and `Class` scheme labels may be noisy due to their low inter-annotator agreement ($\kappa = 0.46$ and $\kappa = 0.00$, respectively). In contrast, the `Prec`, `Rule`, and `Aut` schemes are more reliable, as shown in Table 3.2. These addressed in the original study by introducing "$\text{Avg}_{\text{reliable}}$" metric and evaluation scheme classification, both with and without noisy classes [10].

Table 3.2: Demosthenes argumentation scheme labels

Scheme	Tag value	Count	IAA (κ)
Precedent	`S="Prec"`	503	0.88
Rule	`S="Rule"`	322	0.93
Interpretation	`S="Itpr"`	296	0.46
Classification	`S="Class"`	56	0.00
Authority	`S="Aut"`	53	0.79
Principle	`S="Princ"`	15	0.43

Inter-annotator agreement (Cohen's κ) from Grundler et al. [10].

Level 4 — Link. A link (p, c) is annotated when a sentence p gives a direct, single-step reason for or against sentence c. Indirect or transitive relations are not valid links. Link relations are classified as Support (`<sup>`) if p supports c, and Attack (`<att>`) if p directly contradicts or refutes c. These relationships can be between premises or between a premise and a conclusion.

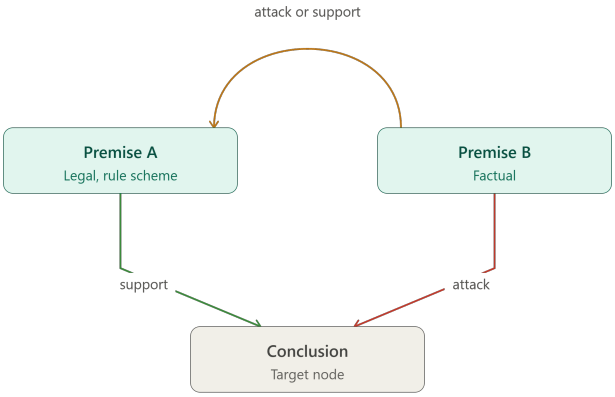

Figure 3.2: Links between Argumentative components

3.1.3 Dataset Usage in the Proposed System

Each legal document is divided into sections (argumentative chains) based on the **"Findings of the Court"** referred to in Table 3.3 and provided to the agents as raw legal text to prevent information loss from long documents. This approach helps agents focus on smaller legal contexts and reduces the risk of missing information during reasoning.

Because the dataset includes annotation labels, a preprocessing step is performed

before inference. Original argumentative annotations are removed to prevent information leakage and ensure agents make independent predictions. As a result, agents receive only the legal text, without gold labels.

During execution, agents collaborate to analyze each argumentative chains. They identify argumentative sentences, determine whether each is a premise or conclusion, classify premises as factual or legal, assign legal argumentation schemes to legal premises, identify argumentative links between related sentences, and save all results in JSON format.

During evaluation, the gold annotations are extracted from the original dataset and stored in a structured JSON format, as shown in Table 3.4. The system outputs are then compared with these gold files to measure performance.

Table 3.3: The ten CJEU State-aid judgments in the evaluation set. A = appeal decision, R = rejection decision.

Document	**Argumentative chains**
A2017_Ellinikos Chrysos AE Metalleion kai Viomichanias Chrysou v European Commission	3
A2018_Commission v Spain	4
A2018_Scuola Elementare Maria Montessori Srl v European Commission	5
R2002_Associação dos Refinadores de Açúcar Portugueses	7
R2004_Daewoo Electronics Manufacturing España SA and Territorio Histórico de Álava	3
R2004_Ramondín SA and Ramondín Cápsulas SA (C-186_02 P) and Territorio Histórico de Álava	2
R2010_AceaElectrabel Produzione SpA v European Commission	6
R2011_European Commission v Kronoply GmbH & Co	2
R2013_Telefónica SA v European Commission	3
R2016_Orange v European Commission	6

3.1.4 Corpus Statistics

Table 3.4 presents the statistical details regarding the structure of the documents analyzed in this thesis [10]:

Table 3.4: Dataset composition

Element	**Count**
Documents (CJEU decisions)	10
Total sentences	1,040
Argumentative sentence	612
Premises (`<prem>`)	570
Conclusions (`<conc>`)	42
Factual premises (`F`)	385
Legal premises (`L`)	213
Scheme: Precedent (`Prec`)	113
Scheme: Rule (`Rule`)	87
Scheme: Interpretation (`Itpr`)	63
Scheme: Classification (`Class`)	13
Scheme: Authority (`Aut`)	16
Scheme: Principle (`Princ`)	3
Support link (`sup`)	559
Attack link (`att`)	21

Several structural properties of the corpus are important for pipeline design decisions:

- Class imbalance between premises and conclusions: The ratio of premises to conclusions is approximately 14 : 1. Conclusions are a small part of argumentative sentences.
- More factual premises than legal premises: The ratio of factual to legal premises is approximately 1.8 : 1.
- Scheme distribution is heavily skewed: `Prec` (113) and `Rule` (87) together account for over 67% of all scheme annotations, while `Princ` (3) is underrepresented. This distribution shows the class imbalance challenge described in the original paper.

3.1.5 Inter-Annotator Agreement and Label Reliability

The original paper reports inter-annotator agreement based on 14 documents annotated by two legal domain experts and addresses conflicts with a third expert [10]. The agreement scores are:

- **Argumentative element detection** (prem/conc vs. non-argumentative): Cohen's $\kappa = 0.95$ — almost perfect.
- **Argumentative classification** (prem vs. conc, given both annotators agreed the sentence is argumentative): $\kappa = 0.86$ — strong.
- **Type classification** (F vs. L): $\kappa = 0.87$ (F) and $\kappa = 0.82$ (L) — strong for both.
- **Scheme classification**: varies widely by class; $\kappa = 0.93$ (Rule) and $\kappa = 0.88$ (Prec) are reliable; $\kappa = 0.46$ (Itpr) and $\kappa = 0.00$ (Class) are not.

These agreement scores set a human performance ceiling for each sub-task. Results within 5 to 10 points of these ceilings are considered competitive. However, results that get close to the ceiling for the noisy Itpr and Class schemes should be viewed with caution.

3.1.6 Limitations

Although Demosthenes is suitable for evaluating legal argument mining systems, the dataset only focuses on fiscal state aid cases from the CJEU, limiting domain diversity. As a result, the proposed system is evaluated within a small legal domain, and its performance in other areas of law (e.g., criminal or contract law) can not be directly inferred from the current experiments.

3.2 LLM and Embedding Setup

3.2.1 Orchestration Framework

The multi-agent pipeline is implemented in Python with **LangGraph** (version 1.2.5), a graph-based framework for designing stateful multi-agent LLM applications [11]. LangGraph handles agent interaction through a typed shared state, conditional routing between nodes, and explicitly defined dependencies in the execution graph. This structure ensures that each agent receives only the information it needs from earlier stages.

The pipeline consists of five specialized LLM agents plus a rule-based critic node as follows:

- **Node 1 — Argumentative Sentence Detection** (Section 4.4.1): Segments the raw text into argumentative propositions and labels each as argumentative or non-argumentative.
- **Node 2 — Role Classification** (Section 4.4.2): Assigns each argumentative component the role of *premise* or *conclusion*.
- **Node 3 — Premise Type Classification** (Section 4.4.3): A multi-label task assigning each premise the type *factual* (F), *legal* (L), or both (FL).
- **Node 4 — Argumentation Scheme Classification** (Section 4.4.4): A multi-label task assigning each legal premise one or more of six schemes (*Rule*, *Prec*, *Itpr*, *Class*, *Aut*, *Princ*).
- **Node 5 — Link Detection** (Section 4.4.5): A two-pass agent predicting directed *support* and *attack* relations, first from premises to each conclusion and then among premises.
- **Node 6 — Structural Critic** (Section 4.4.6): This rule-based checker identifies issues in the argument graph. It does not use any LLM calls and, in its final setup, works only as a diagnostic tool.

Each agent is an independent node within a directed graph. This modular design makes the system flexible, allowing nodes to be inspected, adjusted, or replaced without affecting the rest of the pipeline. The design also provides process-level fault isolation. For example, if one node produces a malformed output or an API fails, the pipeline does not crash. However, classification errors can still pass through the shared state to later stages. This issue is measured separately in Chapter 5 by comparing cascaded and isolated evaluations.

At each step, the pipeline produces intermediate outputs as structured JSON objects with a fixed schema. This approach makes the reasoning process clear, easy to review, and comparable to gold-standard annotations. It also supports the node-level evaluations in Section 4.7, where each node can be tested in isolation on ground-truth inputs and produce outputs in the same format as the full pipeline. This allows for reliable comparisons across different experiments.

3.2.2 Language Model Configuration

All agents in the pipeline use a large language model (LLM) accessed through the OpenRouter API with an OpenAI-compatible interface. The current setup uses a large model **Gemini Pro 3.1** (*"google/gemini-pro-latest"*) and an open-weight model **Gemma 4-26B** (*"google/gemma-4-26b-a4b-it"*) as the main reasoning engine.

The system uses the same temperature settings ($Temperature = 0.0$), based on a preliminary test on document 1 (*A2008_Commission of the European Communities v Salzgitter AG*). Initially, implementing 3-run majority voting at temperature 0.3 for the more challenging classifications (premise type (Node 3) and scheme classification (Node 4)). However, at temperature 0.3, running Document 1 twice yielded inconsistent results: one run failed to assign both L and F labels to certain sentences, while the other assigned both labels correctly. Similar instability occurred with scheme classification. At temperature 0, a single deterministic pass consistently identified FL correctly.

FL and the minority schemes lie on the model's decision boundary. At temperatures above zero, these challenging cases become nearly random results across runs. Majority voting resolves disagreements by selecting the most common label, which often eliminates the rare correct label rather than preserving it. As a result, voting reduced performance instead of supporting it.

The architecture is designed to be *model-agnostic*. Since all communication is performed through a single API interface, the underlying model can be replaced (e.g., GPT, Claude, or other OpenRouter-supported models) without changing the pipeline logic.

3.3 Evaluation Methodology

3.3.1 Schema Alignment Between Demosthenes and Our Pipeline

Table 3.5 shows how the Demosthenes annotation schema matches the output schema of each pipeline node. However, since the IDs of the pipeline's predicted sentences do not match the gold IDs, the system uses **Intersection over Union (IoU)** to compute similarity between predicted and gold text. If the score is ≥ 0.6, they are considered to be the same semantic component, and the predicted unit is assigned to the ground-truth ID.

$$\text{Overlap Score} = \frac{|\text{Tokens}_{\text{pred}} \cap \text{Tokens}_{\text{gold}}|}{|\text{Tokens}_{\text{pred}} \cup \text{Tokens}_{\text{gold}}|}$$

Table 3.5: Mapping between the Demosthenes annotation schema and our pipeline node outputs.

Node	Task	Demosthenes label	Our label
1	ArgDetect	`<prem>` / `<conc>` / unmarked	`is_argumentative` = True/False
2	Prem/Conc	`<prem>` vs `<conc>`	`role = "prem" / "conc"`
3	F/L/FL	`T="F"` / `T="L"` / `T="L|F"`	`premise_type` = `F`/ `L`/ `FL`
4	Scheme	`S="Rule" |"Prec"|...` (multi-label)	`premise_scheme` = List[*scheme*]
5	LinkDetect	*SUP="ID", ATT="ID"*	`has_link`, `link_type`
6	Critic	*n/a*	`CriticNote` list

3.3.2 Evaluation Protocol

The proposed pipeline is evaluated using the annotated dataset described in Section 3.1. Each argumentative chain is processed through the multi-agent pipeline in cascaded and isolated ways, and the predicted outputs are compared with manually annotated ground-truth labels. The system is evaluated in zero-shot and few-shot prompting; no training phase is applied.

To examine error propagation, the system is evaluated in three scenarios:

1. Pipeline scenario 1: Error propagation occurs when upstream errors affect downstream tasks. For example:

   - **A missed legal premise (FN):** If the pipeline misses a legal premise, it can not assign a role, type, or scheme. As a result, a single upstream error can cause four downstream errors, showing error propagation.
   - **A hallucination (FP):** If the pipeline predicts an argumentative premise, but there is no matching gold segment. This false detection counts as FP for the argument and role.

2. Pipeline scenario 2: Without error propagation, each task is evaluated only on sentences correctly predicted by the upstream component.

- **Agent 2 (role: prem/conc):** Consider only sentences that are TP. A TP is a sentence that is argumentative in both the gold data and the pipeline prediction of Agent 1.
- **Agent 3 (type: F / L):** Consider only sentences that are premises in both the gold data and the predictions of Agent 2.
- **Agent 4 (scheme):** Consider only sentences that are legal in both the gold data and the predictions of Agent 3.
- **Agent 5 (link: sup/att):** Consider only edges where both endpoints are TP nodes.

The performance difference between scenarios 1 and 2 reflects the cost of error propagation.

3. Isolated: each node is evaluated on its own. The gold data are used as input, while the output is stored in the same format as in the full pipeline.

3.3.3 Performance Metrics

Performance is evaluated using the F_1 score with four different settings: (1) micro-F_1 is calculated per document, (2) macro-F_1 is calculated per document, (3) micro-F_1 is calculated over the pooled set of all documents, and (4) macro-F_1 is calculated over the pooled set of all documents.

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Where,

$$Precision = \frac{\text{TP}}{\text{TP} + \text{FP}}, \qquad Recall = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.1}$$

where,

- True Positive (TP): a gold segment aligned to a predicted sentence with "$is\_argumentative = True$".
- False Positive (FP): a predicted sentence with "$is\_argumentative = True$" is not aligned to any gold segment.
- False Negative (FN): a gold segment that is either (i) aligned to a predicted sentence with "$is\_argumentative = False$", or (ii) not aligned to any predicted sentence at all.

- True Negative (TN): a predicted sentence with "$is\_argumentative = False$" that is not aligned to any gold segment.

3.3.4 Textual Alignment: Mapping Predictions to Gold IDs

During segmentation, the model often changes punctuation, removes annotation artifacts, or slightly rephrases the text. This can cause predicted units to not be matched to gold units by exact string comparison. To address this, each candidate pair is first normalized by converting to lowercase, removing accents and punctuation, and collapsing whitespace. Then, two complementary similarity measures are used to score them. The first, token set comparison by **Intersection over Union (IoU)**, is the Jaccard index of the two token sets:

$$\text{IoU} = \frac{|\text{Tokens}_{\text{pred}} \cap \text{Tokens}_{\text{gold}}|}{|\text{Tokens}_{\text{pred}} \cup \text{Tokens}_{\text{gold}}|}.$$

Second, a character-level comparison by **sequence-similarity ratio**, $\text{seq}(\cdot, \cdot) \in [0, 1]$. It is based on the longest matching contiguous subsequences between the two normalized strings and assigns higher scores to nearly identical wording, even if a few tokens differ. The final alignment score is the higher value of the two measures:

$$\text{Score}(\text{pred}, \text{gold}) = \max\Big(\text{IoU},\ \text{seq}(\text{pred}, \text{gold})\Big).$$

In each section (argumentative chain), predicted sentences are matched to gold sentences one at a time, starting with the highest-scoring pair. If a pair has a Score ≥ 0.6, it is considered the same semantic unit, and the predicted sentence is assigned the matching gold ID (for example, `A1`, `A3bis`, or `B7`). Predicted sentences that do not meet the threshold remain unaligned and can not be a correct link.

3.3.5 Directed Edge-Set Evaluation

After predictions and gold data use the same ID space, each relation is shown as a directed edge $A \to B$. Here, A is the sentence that supports or attacks, and B is the target sentence. Both gold annotations and model predictions follow this format and are stored as sets of `(doc, source_id, target_id)` tuples. These are grouped by document to prevent ID overlap between documents. The direction of each edge matters, so $A \to B$ and $B \to A$ are treated as different. No graph-level reasoning, such as cycle detection or transitive closure, is used. The support and attack relations are checked as two separate sets of edges.

Edge-level confusion matrix. When comparing the predicted edge set to the gold edge set, the following results are given:

- **True Positive (TP):** an edge $A \to B$ present in *both* the predicted and the gold sets.
- **False Positive (FP):** a predicted edge $A \to B$ that is *not* in the gold set. This includes hallucinated links, links to the wrong unit, reversed-direction links, and *transitivity shortcuts*, such as predicting $A \to C$ when the gold data only includes the intermediate steps $A \to B \to C$.
- **False Negative (FN):** a gold edge $A \to B$ that is *not* in the predicted set, such as a direct relation the model missed or predicted in the reverse direction.

Link detection is scored with TP, FP, and FN. The number of TN sentences does not count because the number of possible non-edges grows dramatically with the number of units and can not be enumerated.

Chapter 4

Implementation

The system is a multi-agent LLM pipeline for Legal Argument Mining (LAM) using a directed graph structure to enable information sharing between specialized agents. The task is decomposed into five sequential sub-tasks, based on the Demosthenes corpus [47]. Each task is assigned to a specialized agent (node), and the structure is verified by a rule-based critic agent, as shown in Figure 4.1. This thesis involved ensuring that a sequence of LLM calls per document section could run reliably, deterministically, and reproducibly via hosted model APIs. It also involved building an evaluation setup that allows the same predictions to be evaluated in different scenarios, as shown in Chapter 5.

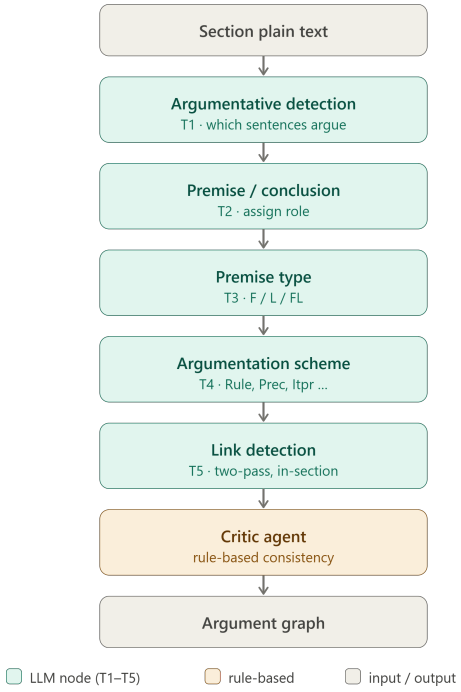

Figure 4.1: Legal Argument Mining Pipeline Nodes

4.1 System Overview and Development Environment

The system was implemented in **Python** using **Google Colab notebooks**. All intermediate results, such as pipeline outputs, per-document metrics, and unified prediction tables, were saved to cloud storage. This allowed long-running experiments to be interrupted and restarted later. The final implementation uses three notebooks:

- `Thesis_pipeline.ipynb`: the main few-shot pipeline, which includes agent definitions, LangGraph orchestration, the cascaded corpus driver, and the isolated-node runners.
- `zero_shot_pipeline.ipynb`: a version of the same pipeline where all examples are removed from the prompts (Section 4.6).
- `evaluation_scenarios.ipynb`: the scenario-based $F1$ scorer that uses the unified prediction tables to produce the results reported in Chapter 5.

Two main groups of components support the implementation. Orchestration is managed by **LangGraph**, which designs the pipeline as a typed state graph. The nodes are specialized agents, while the edges define the control flow, such as early termination for unrecoverable errors. Model access is provided through the **OpenRouter** gateway using the OpenAI-compatible chat-completions interface. This configuration allowed switching between the evaluated models, **Gemini** and **Gemma**, by changing only the model identifier without modifying prompts or orchestration code.

4.2 Data Preparation

As explained in Section 3.1, all experiments use the Demosthenes corpus, which contains decisions from the Court of Justice of the European Union (CJEU) for fiscal state aid law. In this thesis, a set of ten judgments was selected from both appeal (A-prefixed) and rejection (R-prefixed) decisions from 2002 to 2018 (see Table 3.3) [47]. Each judgment was prepared in two parallel representations:

Clean input files. Each document has one JSON file with the plain text of each section in reading order (`sections` $\rightarrow$ `section_`k $\rightarrow$ `plain_text`), with all annotations removed. These files are the only input the pipeline uses.

Structured gold files. Each document also has one JSON file with gold annotations for every argumentative segment. This includes its ID, text, role (`prem`/`conc`), premise type (`F`, `L`, or the multi-value `F|L` mapped to `FL`), legal argumentation schemes, and its support (`sup`) and attack (`att`) relations.

The gold data uses two representational conventions that require handling in the code. First, premise types are stored as a pipe-separated field. A mapping function reduces $\{\texttt{F}\} \to \texttt{F}$, $\{\texttt{L}\} \to \texttt{L}$, and mixed set to `FL` as shown in Table 3.4. Second, and more importantly, the gold link annotation. Each sentence lists the IDs of the sentences that support or attack it, while the pipeline predicts directed edges from source to target. As a result, the final parser converts the gold annotation into a set of explicit directed edges $\{(p, e) \ : \ p \in \sup(e)\}$ for each section. This ensures that both the gold and predicted links use the same directed format before comparison, as explained in Section 3.3.5.

All processing is done at the section level. As explained in Section 3.1.3, giving the model a whole judgment at once goes beyond practical token limits and makes it more likely to lose important information in the middle. Sections are the natural argumentative units in a CJEU decision, such as a plea, a part of a plea, or an admissibility discussion. Gold links in Demosthenes also do not cross section boundaries. For this reason, each part of the pipeline works on one section at a time and starts with a fresh state for each.

4.3 Agent Orchestration Architecture

The pipeline state is a `TypedDict AgentState` object shared by all nodes. It has the attributes shown in Table 4.1.

Table 4.1: Attributes of the LangGraph State definition (`AgentState`).

Attribute	Type	Description
`document_id`	str	Unique document identifier
`raw_text`	str	Input judicial text
`sentences`	List[dict]	Sentence records accumulated across Nodes 1–4 (detection, role, type, scheme)
`links`	List[dict]	ArgumentLink records written by Node 5
`critic_notes`	List[dict]	Active Critic Note records written by Node 6 that the graph is trying to resolve
`error`	Optional[str]	Error message; triggers routing to `END` when set
`retry_count`	int	The refinement loop acts as a circuit breaker; it stops the process when a set limit is reached, helping prevent endless refinement loops.
`audit_log`	List[dict]	Unresolved notes that hit the retry limit are recorded here instead of being dropped silently

The graph is structured as a linear pipeline with six nodes, as shown in Figure 4.2. Each step between nodes uses a conditional edge controlled by a `should_continue` check. If a node logs an error in the shared state, the pipeline responds differently.

The two structural nodes, argumentative detection and premise/conclusion classification, use a **fail-fast** strategy. If the model output can not be parsed, the node logs an error in the state, and the router sends execution to the `END`. Because later stages rely on these identified sentences and their argumentative roles, the pipeline can not continue without them.

In contrast, the three downstream nodes, such as premise type and legal scheme classification and link detection, use a **fail-soft** strategy. If a response is malformed, the error is caught and logged as a warning, and the affected argumentative unit does not get that specific annotation. In this way, a failure only affects one argumentative unit instead of stopping the whole document's processing, so incomplete annotations can be reviewed and fixed later. Empty responses and API errors are retried within a fixed budget, with exponentially increasing wait times before an error is finally raised. However, if a response is malformed but non-empty, it is not retried: the retry logic checks only for emptiness and transport errors, so an unparseable response passes through and is handled by the node's fail-soft catch, which logs a warning and leaves the affected unit unannotated.

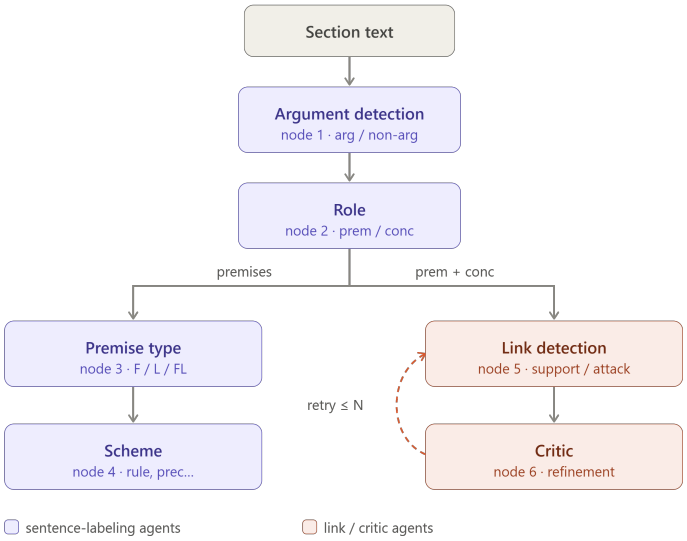

Figure 4.2: Multi-Agent Architecture based on its input dependency

4.4 Node Implementation Details

Each agent generates a specific system prompt, makes one or more LLM calls as shown in Table 4.2, parses the JSON response, and writes its predictions to the shared state. All calls in the final configuration use a temperature of 0 (see Section 3.2.2 for details). The prompts are based on class definitions and developed through an iterative prompt-engineering process in this thesis. The following paragraphs summarize the final version of each prompt.

Table 4.2: LLM call complexity per pipeline node (per section).

Node	Task	LLM Calls
1	Argumentative detection	1
2	Prem/Conc classification	1
3	Type classification (F/L/FL)	1
4	Scheme classification	1
5a	Link detection: prem→conc (one call per conclusion)	C
5b	Link detection: prem→prem (single batched call)	1
6	Critic agent	0 (rule-based)
Total		$C + 5$

For Node 5, the total number of API calls is given by $C + 1[|P| \geq 2]$, where C is the number of conclusions and $|P|$ is the number of premises. If a section contains

no premises, the `"detect_links"` function terminates immediately, resulting in zero calls. If it contains exactly one premise, the second pass contributes zero calls instead of one. Consequently, in the standard scenario where a section contains at least two premises, the expression simplifies directly to $C + 1$.

4.4.1 Node 1: Argumentative Sentence Detection

The first agent takes the raw section text and, in a single pass, both segments it and labels each resulting unit with a binary `is_argumentative` flag. Segmentation is performed at the level of **argumentative propositions** instead of grammatical sentences, since one sentence can often have more than one argumentative role. The prompt is guided by three main rules as shown in Appendix Prompt A.1:

- **Proposition-level splitting.** A single sentence can have two argumentative parts with different roles, usually a premise followed by a conclusion, so it should be split in these cases. However, it should never be split into two parts with the same role. This rule was added after early tests showed that splitting only by sentences did not work well, as seen in the *A2008_Commission of the European Communities v Salzgitter AG* document and the following example:

  > *"In light of the response to the first limb of the first plea, it must be held that the second limb of that plea is unfounded [...], and that the second limb must therefore be rejected."*

  A segmenter that splits by periods would keep this as one unit, but it actually contains two roles: the finding *"the second limb [...] is unfounded"* (a premise) and the verdict *"the second limb must therefore be rejected"* (a conclusion). Splitting at the proposition level separates these into a `PREM` and a `CONC`. To avoid the opposite problem—splitting a single long premise into two premise fragments—a rule forbids splitting parts with the same role.

- **Scope.** Only the Court's own reasoning counts as argumentative. Sentences that just report what the parties or the Commission said are considered context and should be left out. This follows the corpus's annotation policy and was the most effective way to reduce false positives (FP).

- **Component definitions.** Premises are evidence, rules, or interpretations used to justify a ruling, while conclusions are the claims that follow. Legal interpretations used to justify a rejection are considered premises, not conclusions. These components are defined for better splitting.

4.4.2 Node 2: Premise/Conclusion Classification

Role classification in this thesis uses the DISPOSE/DESCRIBE test. The prompt is guided as follows, as shown in Appendix Prompt A.2:

- A sentence counts as a conclusion only if it disposes of an argumentative unit by formally closing a plea, limb, or ground of appeal with a verdict, such as *"must be rejected," "must be set aside," or "cannot succeed."*.
- A sentence that describes a legal position, fact, or consequence is considered a premise, even if it sounds conclusive.
- The prompt tells the model not to rely on words like *"consequently," "therefore," or "it follows,"* since these appear in both conclusions and premises. An annotated example with three sentences shows that a sentence starting with *"consequently"* can still be a premise if it states a consequence regarding a party's legal position rather than a verdict. This test is designed to address the main mistake in naive prompting, in which every sentence started with those specific words is marked as a conclusion.

The last rule focuses on a common mistake in naive prompting, where any sentence starting with a certain marker is labeled as a conclusion. The prompt includes a three-sentence example to show this problem clearly.

[1] "The aid falls under Article 4(c) CS." → `prem`
[2] "Consequently, Germany cannot rely on an authorisation under Article 67 CS." → `prem`
[3] "Therefore, the plea must be rejected." → `conc`

Although both [2] and [3] begin with deductive markers, [2] outlines a consequence for a party's legal position and should be classified as a premise. In contrast, only [3] resolves the plea with a verdict. It is the logical action of the sentence, not just the cue word. Previous versions of the prompt relied primarily on cue words and often produced too many conclusions. The Dispose/Describe test was added to fix this issue.

4.4.3 Node 3: Premise Type Classification (F/L/FL)

Premises are labeled as factual (F), legal (L), or both (FL) using a single main test, as shown in Appendix Prompt A.3: *"Is the sentence asserting something specific to this case, or stating a general legal norm that holds beyond it?"*

- If a sentence names a specific party and uses phrases such as *"the aid"*, *"the decision"*, or *"this plea"* signals factual (F).
- If a sentence could appear in a legal textbook without a case context, it is legal (L).
- There are two important exceptions that make this task more challenging, and the prompt explains them using a minimal pair based on the same article:

  1. **Exception 1:** Mentioning a legal article does not make a sentence legal if the article only points to a case-specific finding.

     "The aid fell within the scope of Article 4(c) CS." $\rightarrow$ **F** (the article only locates the finding)

  2. **Exception 2:** Naming an institution does not make a sentence factual if it states a general interpretive rule.

     "Article 4(c) CS prohibits aid in any form whatsoever." $\rightarrow$ **L** (a general norm)

  Both examples mention "Article 4(c) CS" but the function of the reference is different in each case.

- The mixed class (FL) is rare and used only for sentences in which both a general rule and its application to a specific fact are essential. For example, in the below sample where a general prohibition and its specific application are both essential:

  "Given that Article 1 prohibited all steel aid, the Commission could not withdraw the 1971 Decision."

4.4.4 Node 4: Argumentation Scheme Classification

Scheme classification is used only for legal and mixed premises (L/FL) following the corpus's annotation policy [10]. There are six multi-label schemes: *Rule*, *Prec*, *Itpr*, *Aut*, *Class*, and *Princ*, as shown in Table 3.1. The prompt uses an ordered decision rule to address the main confusion that occurred during development, especially the distinction between *Rule* and *Itpr*, as shown in Appendix Prompt A.4:

1. If a sentence names a specific article, treaty provision, or code, including sentences that derive consequences from a named provision, it is always labeled as *Rule*.

2. If it cites a specific past case by name or number, *Prec* is also added (so a sentence can be both *Rule* and *Prec*).

3. An interpretation without a named legal source is labeled as *Itpr*.

4. *Aut*, *Class*, and *Princ* are checked separately, based on their typical signals: phrases like *"as the commission submits,"* classification under a legal category, and mention of a named general principle, respectively.

The most difficult distinction to make was between *Rule* and *Itpr*. The key test is whether a specific legal source is named:

> *"It follows from Article 305(1) EC that the EC and ECSC Treaties are independent."* → **Rule** (derives from a named article)
> *"The ECSC Treaty does not distinguish between new and existing aid."* → **Itpr** (interpretation with no named source)

This criterion was developed through several revisions. At first, *Itpr* was defined as an interpretation that "goes beyond the literal text" of a provision, but this overlapped with *Rule* whenever an article was mentioned. The final version defines *Itpr* as an interpretation without a named legal source and sets a clear rule: if any specific article is named, it counts as *Rule*, not *Itpr*. This made the distinction clearer.

4.4.5 Node 5: Two-Pass Link Detection

Link detection is the most challenging part of the pipeline. The design choice is based on the structure of the argument graphs in the corpus. There are three main features: (1) The number of premises is more than conclusions; (2) the graphs are very sparse, with only 2% to 4% of all candidate pairs actually linked; and (3) the links between premise and conclusion are directional, meaning premises always support or attack conclusions. These three factors guide the design described below. It works in two passes for each section, as shown in Appendix Prompt A.5.

Pass 1 (anchor-based premise→conclusion). Each conclusion acts as an anchor. The agent gets one conclusion along with all the premises in the section and must judge each premise as a possible direct supporter or attacker. For each premise, it gives a verdict, a short chain-of-thought reasoning, a type (support/attack), and a confidence level. Focusing on one conclusion at a time keeps the process efficient and allows the model to reason about a single target.

Pass 2 (premise→premise chains). All ordered pairs of premises in the section are checked together for serial-support chains and collective support.

Both passes follow three rules as follows:

- **Directness.** A link must be made in a single reasoning step. Transitive links are not allowed. For example, if $1 \to 2$ and $2 \to 3$, the edge $1 \to 3$ can not be added unless premise 1 directly supports premise 3.
- **Collective support**, where several premises together support a target, still counts as one step.
- **Direction and default-to-false.** Edges always run from premise to target, never the other way around. The prompt makes it clear that most premise-conclusion pairs are not linked, helping to prevent the model from forming too many links.
- **Confidence.** Links are rated by the type of textual evidence. Explicit signal words indicate high confidence, implicit but clear logic indicates medium, and just topical similarity indicates low. In the premise-to-premise pass, low-confidence positives are removed during parsing to avoid adding speculative links, but premise-to-conclusion links are considered regardless of their confidence.

In the few-shot configuration, each pass includes a selected block of examples from *A2008_Commission of the European Communities v Salzgitter AG* document, which is not included in the evaluation set, as shown in Few-Shot Prompt A.6. The examples show not only positive support and attack cases but also mention the two main failure types seen in early tests: the transitivity trap, where a premise links to a distant conclusion across missing steps, and independent rules, where two legal norms on the same topic have no deductive link.

The directness rule addresses a failure mode from early tests called the transitivity trap. In this situation, the model connects a premise to a distant conclusion and skips intermediate links. The prompt shows this as a negative example from *A2008_Commission of the European Communities v Salzgitter AG* document:

> Premise: *"The Commission had knowledge of Salzgitter's aid since the end of 1988 through annual reports submitted to it."*
> Conclusion: *"The cross-appeal must therefore be dismissed in its entirety."*
> Expected: `has_link = false`.

In the few-shot configuration, the link detection is implemented with a block of examples from *A2008_Commission of the European Communities v Salzgitter AG* document as shown in Appendix Prompt A.6.

4.4.6 Node 6: Critic Agent

The final node is a rule-based structural checker. It verifies graph-level correctness properties as follows:

- **Orphan premises**: A premise that is the source of no active link.
- **Unsupported conclusions**: A conclusion that is not supported or attacked by any premise.
- **Contradictory links**: The same ordered pair predicted as both a support and an attack edge. This is the only check that was raised as an `error` rather than a `warning`.
- **Low-confidence links**: Links are flagged for review.

Its output is diagnostic at the graph level. The critic's notes are added to the output JSON for later analysis.

Note. The prompt texts were developed with the assistance of generative AI for two purposes: to clarify legal concepts in the Demosthenes corpus and to improve the grammar, clarity, and formatting of the prompt text. The task decomposition, annotation rules, prompt structure, and all methodological decisions are the author's own. The complete prompts are reported in Appendix A.

4.5 Robustness Engineering

Running a five-agent cascade through the OpenRouter API across 10 multi-section documents, which involves several hundred LLM calls per configuration, reveals failure patterns that do not appear in single-prompt experiments. 4 mechanisms were developed in response.

Retries with exponential sleep intervals. The shared `call_llm` function for the Gemma wrapper retries each call up to four times, using exponentially increasing wait times after each attempt. If a hosted endpoint returns an *empty* completion, such as when rate limiting or truncation occurs with only a `finish_reason`, the

wrapper treats this as a failure and retries the call instead of passing it along. This approach addresses a silent failure found during development, where rate-limited calls returned `None` content appeared to be empty predictions and could not be distinguished from real model abstention in the metrics.

Defensive JSON parsing. Model responses are parsed by finding the outermost brace-delimited section before using `json.loads`. This approach tolerates preambles and Markdown fences that models may emit, even when instructed otherwise.

Section-level fresh state and checkpointing. The cascaded driver creates new agents and, therefore, a clean LangGraph state for each section. Creates a new `LegalArgumentAgent` for each section, releases it (`del` + `gc.collect()`) before proceeding, and saves the document's output JSON after each section as a checkpoint. If a run crashes or is rate-limited, the completed sections are preserved, allowing processing to resume at the document level using the driver's `start` index. This avoids re-running the entire corpus.

Determinism: from self-consistency voting to a single temperature-0 pass. Nodes 3 and 4 originally used self-consistency majority voting, sampling three runs at temperature 0.3 and selecting the most frequent label. Preliminary experiments performed at temperature 0.3 show inconsistency in predicting labels for the most challenging cases, including the `FL` class and minority schemes. Because voting did not stabilize the targeted cases and tripled the number of API calls, the final configuration runs each node once at a temperature of 0.

4.6 Zero-Shot prompting

To measure the effectiveness of few-shot prompting, a second notebook was created that copies the pipeline exactly. It uses the same architecture and output format as with $temperature = 0$, but removes all in-context examples from each prompt and keeps the definitions unchanged. The zero-shot version was run with the open-weight Gemma model, described in Chapter 5.

4.7 Isolated-Node setup

Cascaded scores have two types of errors: a node's own limitations and any mistakes it inherits from earlier steps. To separate them, the isolation performance of each node

is executed. Each node from Nodes 2 to Node 5 is run in isolation on gold upstream input. Node 2 gets the gold argumentative segments; Node 3, the gold premises; Node 4, the gold legal and mixed premises; and Node 5, the gold argumentative segments with their gold role. The setup uses the same node functions as the pipeline, and their outputs match the pipeline's JSON schema.

Chapter 5

Evaluation

This chapter presents the evaluation of the multi-agent legal argumentation mining pipeline. The evaluation focuses on analyzing various aspects, including error propagation through the cascade pipeline, comparisons between the commercial and open-weight models, the impact of few-shot examples, and individual agent behavior.

5.1 Experimental Setup

5.1.1 Configurations

Three configurations were evaluated, each using the same architecture and deterministic temperature 0 as described in Chapter 4:

1. **Gemini FS** — the few-shot pipeline run with the Gemini model, cascaded and isolated;
2. **Gemma FS** — the few-shot pipeline run with the open-weight Gemma model, cascaded and isolated;
3. **Gemma ZS** — the zero-shot pipeline (Section 4.6) run with the open-weight Gemma model, cascaded and isolated.

5.2 Evaluation Protocol

5.2.1 Scenarios

Each configuration is evaluated in three scenarios:

Scenario 1 — end-to-end pipeline (S1). This is the real end-to-end setup, where earlier mistakes affect later tasks, meaning errors propagate across the pipeline.

Scenario 2 — upstream conditioning (S2). This uses the same cascaded pipeline, but each task is scored only where the previous step was correct. For example, roles are scored only on components that were correctly detected (true positives), and types are scored on true positive components that are also correctly classified as premises. The difference between S2 and S1 shows the propagation cost for each task.

Isolated (Iso). In this scenario, each node receives gold as input. Iso shows the best possible performance for each node with the current prompts. The difference between Iso and S2 shows how much the cascade affects when a node's input is correct.

5.2.2 Aggregation Modes and Metrics

Each scenario is reported using two aggregation methods. In the **doc-by-doc** approach, each metric was calculated for each document, and the results were then averaged. This method gives equal weight to each judgment and shows how scores differ across documents. In the **pooled** approach, all instances are combined, and a single F1 score is computed, giving equal weight to each instance. Both methods are reported because they often differ for rare classes. Pooled scores are mostly influenced by the few documents where a rare class appears, while doc-by-doc scores reflect averages over smaller groups within each document.

Macro-F1 takes the average of the F1 scores for every class that appears in either the gold standard or the model's predictions. If a class is predicted but not found in the gold standard, it gets a score of 0, so classes with only false positives are penalized. Classes that appear in neither the gold standard nor the predictions are excluded. Micro-F1 works differently: it adds up the true positives, false positives, and false negatives from all classes before calculating a single F1 score. This approach gives equal weight to each instance, so common classes have a bigger influence. The gap between Macro-F1 and Micro-F1 highlights how much a task's performance depends on its rare classes.

Link F1 is a directed edge-set F1 score calculated using $(document, source, target)$ triples. A predicted edge counts as a true positive only if all three fields match a gold edge exactly. The score is calculated separately for support and attack relations, then combined into a macro variant (the average of the two) and a micro variant (using their pooled counts).

If a class or relation does not have any gold or predicted positives in the scored

Table 5.1: Macro-F1 (pool) Results for argumentative detection, role, premise type, and scheme classification.

	Arg	Role			Type			Scheme						
Model / Setting	**Arg**	**Prem**	**Conc**	**M-F$_1$**	**F**	**L**	**M-F$_1$**	**Aut**	**Class**	**Itpr**	**Prec**	**Princ**	**Rule**	**M-F$_1$**
Isolated setting														
Gemma ZS	.84	.95	.59	.77	.95	.86	.90	.71	.38	.59	.93	.36	.90	**.65**
Gemma FS	.74	.96	.68	.82	.94	.85	.90	.60	.07	.55	.92	.31	.92	.56
Gemini FS	**.89**	.97	.72	**.84**	.96	.92	**.94**	.57	.00	.50	.91	.27	.97	.54
Upstream conditioning														
Gemma ZS	-	.79	.52	.65	.74	.77	.76	.50	.50	.40	.62	.67	.65	.56
Gemma FS	-	.84	.65	.75	.82	.85	.84	.67	.11	.47	.79	.00	.83	.48
Gemini FS	-	.97	.71	**.84**	.96	.92	**.94**	.53	.00	.52	.91	.31	.97	.54
End-to-End Pipeline														
Gemma ZS	-	.67	.37	.52	.40	.62	.51	.20	.40	.25	.43	.40	.39	.34
Gemma FS	-	.61	.46	.54	.41	.63	.52	.33	.09	.32	.50	.00	.47	.29
Gemini FS	-	.89	.50	.69	.81	.88	.84	.35	.00	.46	.86	.31	.82	.47

Table 5.2: Macro-F1 (doc-by-doc) Results for argumentative detection, role, premise type, and scheme classification.

	Arg	Role			Type			Scheme						
Model / Setting	**Arg**	**Prem**	**Conc**	**M-F$_1$**	**F**	**L**	**M-F$_1$**	**Aut**	**Class**	**Itpr**	**Prec**	**Princ**	**Rule**	**M-F$_1$**
Isolated setting														
Gemma ZS	.81	.94	.61	.78	.95	.78	.86	.46	.15	.67	.95	.11	.77	.62
Gemma FS	.74	.97	.71	.84	.94	.77	.86	.53	.03	.57	.95	.11	.79	.60
Gemini FS	**.87**	.97	.77	**.87**	.96	.89	**.93**	.43	.00	.57	.95	.10	.95	**.62**
Upstream conditioning														
Gemma ZS	-	.78	.54	.66	.77	.71	.74	.33	.25	.39	.51	.50	.45	.39
Gemma FS	-	.82	.67	.74	.76	.70	.73	.40	.08	.41	.68	.00	.83	.47
Gemini FS	-	.96	.73	.85	.96	.89	.92	.32	.00	.55	.95	.10	.95	.57
End-to-End Pipeline														
Gemma ZS	-	.65	.41	.53	.42	.55	.49	.10	.17	.26	.33	.33	.28	.23
Gemma FS	-	.59	.47	.53	.38	.51	.45	.17	.04	.21	.35	.00	.38	.21
Gemini FS	-	.86	.51	.69	.80	.80	.80	.20	.00	.44	.88	.10	.66	.44

set, its F1 score is considered (NaN) and is not included in the averages.

5.3 Results

Table 5.1, Table 5.2, Table 5.3, Table 5.4, and Table 5.5 report the results for component detection, role classification, premise type classification, scheme classification, and link identification. Each of these results is analyzed below.

5.4 Analysis

5.4.1 Task-by-Task Overview

The multi-agent pipeline performs well in the early stages of LAM, but scheme classification and relational identification remain challenging. With the best setup

Table 5.3: Micro-F1 (pool) Results for argumentative detection, role, premise type, and scheme classification.

	Arg	Role			Type			Scheme						
Model / Setting	**Arg**	**Prem**	**Conc**	**μ-F$_1$**	**F**	**L**	**μ-F$_1$**	**Aut**	**Class**	**Itpr**	**Prec**	**Princ**	**Rule**	**μ-F$_1$**
Isolated setting														
Gemma ZS	.84	.95	.59	.90	.95	.86	.92	.71	.38	.59	.93	.36	.90	**.79**
Gemma FS	.74	.96	.68	.94	.94	.85	.91	.60	.07	.55	.92	.31	.92	.75
Gemini FS	**.89**	.97	.72	**.95**	.96	.92	**.95**	.57	.00	.50	.91	.27	.97	.76
Upstream conditioning														
Gemma ZS	-	.79	.52	.75	.74	.77	.76	.50	.50	.40	.62	.67	.65	.57
Gemma FS	-	.84	.65	.82	.82	.85	.84	.67	.11	.47	.79	.00	.83	.66
Gemini FS	-	.97	.71	.94	.96	.92	.94	.53	.00	.52	.91	.31	.97	.78
End-to-End Pipeline														
Gemma ZS	-	.67	.37	.62	.40	.62	.49	.20	.40	.25	.43	.40	.39	.37
Gemma FS	-	.61	.46	.59	.41	.63	.50	.33	.09	.32	.50	.00	.47	.41
Gemini FS	-	.89	.50	.84	.81	.88	.84	.35	.00	.46	.86	.31	.82	.69

Table 5.4: Micro-F1 (doc-by-doc) Results for argumentative detection, role, premise type, and scheme classification.

	Arg	Role			Type			Scheme						
Model / Setting	**Arg**	**Prem**	**Conc**	**μ-F$_1$**	**F**	**L**	**μ-F$_1$**	**Aut**	**Class**	**Itpr**	**Prec**	**Princ**	**Rule**	**μ-F$_1$**
Isolated setting														
Gemma ZS	.81	.94	.61	.90	.95	.78	.90	.46	.15	.67	.95	.11	.77	**.81**
Gemma FS	.74	.97	.71	.94	.94	.77	.90	.53	.03	.57	.95	.11	.79	.78
Gemini FS	**.87**	.97	.77	**.95**	.96	.89	**.95**	.43	.00	.57	.95	.10	.95	.79
Upstream conditioning														
Gemma ZS	-	.78	.54	.74	.77	.71	.77	.33	.25	.39	.51	.50	.45	.48
Gemma FS	-	.82	.67	.80	.76	.70	.74	.40	.08	.41	.68	.00	.83	.53
Gemini FS	-	.96	.73	.94	.96	.89	.94	.32	.00	.55	.95	.10	.95	.79
End-to-End Pipeline														
Gemma ZS	-	.65	.41	.61	.42	.55	.49	.10	.17	.26	.33	.33	.28	.30
Gemma FS	-	.59	.47	.57	.38	.51	.43	.17	.04	.21	.35	.00	.38	.28
Gemini FS	-	.86	.51	.82	.80	.80	.81	.20	.00	.44	.88	.10	.66	.65

(Gemini FS), the system detects argumentative components with a pooled F1 score of **0.89**, providing downstream agents with a solid starting point.

In the end-to-end pipeline (S1), role classification achieves **0.84** micro-F1 and 0.69 macro-F1, while premise type classification achieves **0.84** for both. This shows that, once argumentative components are detected, the model classifies their properties well.

Performance drops on the more complex reasoning tasks. For argumentation scheme classification, the model reaches 0.69 micro-F1 but only **0.47** macro-F1. It therefore performs well on common schemes, such as Rule and Prec, but struggles with rare ones; removing the two hardest classes (Itpr, Class) raises the scheme micro-F1 to **0.80**. This indicates that most errors come from a few ambiguous or rare schemes.

Link identification is the weakest part of the pipeline. Support relations achieve

Table 5.5: Link detection results across all evaluation settings: Macro-F1 (pool), Macro-F1 (doc-by-doc), Micro-F1 (pool), and Micro-F1 (doc-by-doc). Sup and ATT report per-class F1 for the *support* and *attack* relations.

	Macro-F_1 (pool)			Macro-F_1 (doc-by-doc)			Micro-F_1 (pool)			Micro-F_1 (doc-by-doc)		
Model / Setting	**Sup**	**ATT**	**M-F_1**	**Sup**	**ATT**	**M-F_1**	**Sup**	**ATT**	**μ-F_1**	**Sup**	**ATT**	**μ-F_1**
Isolated setting												
Gemma ZS	.30	.00	.15	.30	.00	.20	.30	.00	.28	.30	.00	.29
Gemma FS	.23	.00	.11	.22	.00	.15	.23	.00	.22	.22	.00	.21
Gemini FS	.48	.00	.24	.47	.00	.29	.48	.00	.46	.47	.00	.46
Upstream conditioning												
Gemma ZS	.25	.00	.12	.26	.00	.23	.25	.00	.25	.26	.00	.26
Gemma FS	.26	.00	.13	.28	.00	.24	.26	.00	.25	.28	.00	.28
Gemini FS	.48	.10	**.29**	.47	.07	**.38**	.48	.10	**.47**	.47	.07	**.46**
End-to-End Pipeline												
Gemma ZS	.20	.00	.10	.21	.00	.14	.20	.00	.20	.21	.00	.21
Gemma FS	.20	.00	.10	.20	.00	.13	.20	.00	.19	.20	.00	.19
Gemini FS	.43	.06	.24	.40	.05	.27	.43	.06	.41	.40	.05	.39

an F1 score of **0.43**, while attack relations are rarely detected (**0.06**). Overall, link micro-F1 is 0.41, and macro-F1 is 0.24. This shows that identifying argumentative relations is much harder than classifying individual components.

5.4.2 The Cost of Error Propagation

The comparison between the end-to-end setting (S1) and the conditioned cascade (S2) shows the effect of error propagation. Since S2 receives the correct upstream predictions, the performance gap shows the amount of information lost due to earlier mistakes in the cascade.

Providing correct upstream input for Gemini FS improves every downstream task. Role classification increases from 0.69 to **0.85** macro-F1, premise type classification from 0.80 to **0.92**, scheme classification from 0.44 to **0.57**, and link detection from 0.39 to **0.46** micro-F1 (doc-by-doc evaluation). Overall, around 12-16 F1 points are lost due to upstream errors in the classification tasks, while link identification loses around 7 F1 points.

The impact is larger for the open-weight Gemma model. Role classification improves from 0.53 to **0.74** macro-F1, while premise type classification increases from 0.45 to **0.73**. These larger gaps indicate that Gemma produces more missed and hallucinated components during the earlier stages.

To determine whether the cascade itself introduces additional limitations, the conditioned cascade (S2) can be compared with the isolated evaluation (Iso), where each node receives gold input directly. For Gemini, the two settings are nearly the same for role classification (0.847 vs. 0.874), premise type classification (0.924

vs. 0.928), and link detection (0.459 vs. 0.458), with a slightly larger difference for scheme classification (0.566 vs. 0.623). These results show that, once the correct components are available, the downstream agents perform almost as well inside the pipeline as they do in isolation. In contrast, Gemma shows larger gaps between S2 and Iso, especially for premise types (0.728 vs. 0.857), suggesting that even when components are detected correctly, downstream agents receive less reliable context.

In summary, the results suggest that the proposed task decomposition works well. Most of the performance loss in later stages comes from mistakes made earlier, not from the downstream agents themselves.

5.4.3 Link Identification Is an Intrinsic Weakness

The interesting finding from the isolated evaluation is about link detection. If low link scores were mainly due to propagation, the isolated node, which receives gold components with gold roles, should score much higher than the cascade node. However, the results show a different scenario. For Gemini FS, the isolated Support-F1 is 0.47, vs. 0.47 in the conditioned cascade and 0.40 end-to-end (doc-by-doc). For Gemma FS, the isolated performance (0.22) is even slightly below the conditioned cascade (0.28). This shows that link identification is inherently weak. Attack edges, with only 21 gold instances, are almost never found (best F1 = 0.10).

5.4.4 Model Comparison: Gemini vs. Gemma

The commercial Gemini model usually performs better than the open-weight Gemma model, although the performance gap depends on the task complexity. For local classification tasks, the difference is small. In the isolated setting, role classification reaches **0.87** macro-F1 for Gemini compared with **0.84** for Gemma, indicating that both models can classify argumentative roles accurately with correct input.

The difference is larger for premise type classification (0.93 vs. 0.86), grows even more for scheme classification, and is largest for link identification. In this last task, Gemini achieves more than twice the isolated link micro-F1 score of Gemma (0.46 vs. 0.21). While Gemma can achieve performance close to Gemini on classification tasks, it is weaker at detecting relations between components, resulting in much lower overall performance, especially when reconstructing the argumentative graph.

These findings suggest that a hybrid architecture could effectively combine the strengths of both models: an open-weight model for relatively simple classification tasks and a stronger model for more complex tasks such as relation prediction.

5.4.5 Few-Shot vs. Zero-Shot Prompting

Comparing Gemma in the few-shot (FS) and zero-shot (ZS) settings shows the effect of demonstrations in the pipeline. The results reveal three different patterns:

First, few-shot prompting does not improve argumentative component detection. Zero-shot performs slightly better (0.81 vs. 0.74 detection F1, doc-by-doc). The prompt already provides clear decision rules, so adding demonstrations only lengthens the prompt without giving the model any new information. This small difference in detection leads to the next pattern: the few-shot detector forwards fewer, noisier components to later nodes.

Second, the impact of few-shot examples on downstream tasks depends on the evaluation setting. In the end-to-end pipeline (S1, doc-by-doc), few-shot prompting does not help: Gemma FS performs about the same as, or slightly worse than, Gemma ZS on role- and premise-type classification and on scheme macro-F1 (role micro-F1 0.57 vs. 0.61, premise-type micro-F1 0.43 vs. 0.49, scheme macro-F1 0.21 vs. 0.23). This is because the few-shot detector passes fewer and noisier components to the later nodes. The value of examples appears only once the upstream input is corrected. In the conditioned cascade (S2, doc-by-doc), few-shot prompting clearly improves the harder classification tasks: role macro-F1 increases from 0.66 to 0.75 and scheme macro-F1 from 0.39 to 0.47, while premise-type classification stays about the same (0.74 to 0.73). Demonstrations help, but only when the node gets clean input. When error propagates, this benefit disappears.

Finally, the link detector is the most sensitive to different settings. When using clean gold input, zero-shot performs better than few-shot (Support-F1: 0.30 vs. 0.22), suggesting that the link demonstrations, which are used to illustrate transitivity and independence traps, make the smaller model overly cautious when the input is already clean. Few-shot only does better than zero-shot in the conditioned cascade (0.28 vs. 0.26 in S2). When there is end-to-end error propagation, the two methods are almost the same, with zero-shot slightly ahead (0.20 vs. 0.21). This means the demonstrations do not make the model more robust to upstream noise. Their small benefit appears only when the upstream input is correct.

Chapter 6

Conclusion and Future Work

6.1 Summary of the Thesis

This thesis evaluates the performance of a multi-agent system for legal argument mining using Gemini and Gemma LLMs. The task is divided into five specialized LLM agents: argumentative detection, role classification, premise type classification, legal scheme classification, and two-pass directed link identification, plus a rule-based critic node. The LangGraph framework manages these agents, which are evaluated on the Demosthenes corpus of CJEU state-aid judgments.

The pipeline is implemented for both cascaded and isolated agent predictions. Performance is measured across three scenarios: end-to-end with error propagation, upstream conditioning without error propagation, and isolated performance, at two aggregation levels, per-document and pooled.

6.2 Main Findings

The main findings of this evaluation can be summarized in five key point statements.

1. **Decomposition:** The result shows that decomposing legal argument mining into specialized sub-tasks is an effective strategy, as agents achieve promising performance when provided with correct upstream inputs, particularly in argumentative detection, role classification, and premise type classification. However, the end-to-end pipeline is still limited by error propagation, underscoring the importance of accurate upstream predictions and robust intermediate outputs. It also performs weakly on the hardest sub-task, as link identification remains weak even in an isolation setting. The other benefit of this approach is in diagnosis: evaluating each agent across different settings reveals where the

pipeline fails and separates upstream errors from those inherent to the task's difficulty.

2. **Error propagation:** The difference between the upstream conditioned setting, where each agent is scored only if its predecessors were correct, and the end-to-end setting, where errors can carry forward, shows the impact of error propagation on downstream tasks. Both the commercial model (Gemini) and the open-weight model (Gemma) are affected by error propagation. However, Gemma shows greater vulnerability. Its weaker detection leads to more missed and hallucinated components, and each one causes errors in every downstream task. This shows that detection quality at the start of the cascade is the most important factor for overall performance.

3. **Isolated agent performance:** When each agent receives gold upstream input in isolation, the agents achieve promising results for argumentative detection and role and premise type classification and good results for scheme classification, while the agent struggles with rare ones. The link identification agent is different; its performance does not improve. This suggests that the challenge of relation identification comes from its intrinsic difficulty, not from errors in earlier stages. These isolated results highlight two types of limitations: for the first four agents, the main end-to-end problem is upstream errors, while for the link agent, the problem persists, indicating that relation identification is the main challenge for the system.

4. **Model capability:** Relation identification is where the two models show the biggest difference. On isolated link identification, where each receives gold input so the comparison reflects reasoning ability rather than input quality, Gemini reaches more than twice the micro-F1 of Gemma. The gap on the first four agents is considerably smaller. This pattern suggests that the classification sub-tasks transfer well to the open-weight model, but relation identification still requires the stronger model, motivating a hybrid design that reserves the commercial model for link detection.

5. **Prompt strategy:** Comparing the two Gemma configurations, the effect of few-shot examples differs by agent and by whether error propagation is present. Argumentative detection is slightly better under zero-shot prompting, since the detection rules are already explicit and extra demonstrations do not help. Once the upstream input is clean (the conditioned setting), few-shot prompting improves role and scheme classification, while premise type

classification remains unchanged. In the true end-to-end pipeline, however, this benefit disappears: the few-shot detector passes fewer and noisier components downstream, so few-shot performs at or below zero-shot overall. The difference in relation identification is small across all settings.

6.3 Future Work

The pipeline proposed in this thesis is implemented in a feed-forward manner. Each agent uses the output of the preceding ones, so any early error will propagate to later stages and impact their performance. In the next step, can explore advanced multi-agent mechanisms to provide feedback or corrections on earlier stages before the argument graph is complete.

Multi-agent debate. Instead of relying on just one agent's verdict, a debate setup can be used. For example, agents might decide whether a sentence is a premise or a conclusion. They then critique each other's reasoning over one or more rounds before deciding the final label. This approach is more useful for tasks where a single agent is least reliable, such as scheme classification or relation detection, since using it for each task would be too expensive.

Iterative self-reflection. Another useful approach is to let agents reflect on their own outputs. After producing an initial argument graph, either the agent or a dedicated reviewer would review the result in natural language, identify possible mistakes, and rerun the affected predictions. This process continues until the output becomes stable. This method works well for correcting upstream classification errors early, before they affect later steps.

Closing the loop with the structural critic. It already finds structural problems in the final graph, such as orphan premises, unsupported conclusions, and contradictory links, but currently it only reports them. In the future, this process could be improved by sending each flagged defect back as a targeted prompt to revise only the affected parts, rather than repeating the entire process, such as the edges or labels involved in restructuring the graph, so that reflection helps stabilize the graph.

Bibliography

[1] Raquel Mochales Palau and Marie-Francine Moens. "Argumentation mining: the detection, classification and structure of arguments in text". In: *The 12th International Conference on Artificial Intelligence and Law, Proceedings of the Conference, June 8-12, 2009, Barcelona, Spain*. ACM, 2009, pp. 98–107. DOI: 10.1145/1568234.1568246. URL: https://doi.org/10.1145/1568234.1568246.

[2] Ilias Chalkidis et al. "LexGLUE: A Benchmark Dataset for Legal Language Understanding in English". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, 2022, pp. 4310–4330. DOI: 10.18653/v1/2022.acl-long.297. URL: https://aclanthology.org/2022.acl-long.297/.

[3] Marco Lippi and Paolo Torroni. "Argumentation Mining: State of the Art and Emerging Trends". In: *ACM Trans. Internet Techn.* 16.2 (2016), 10:1–10:25. DOI: 10.1145/2850417. URL: https://doi.org/10.1145/2850417.

[4] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423/.

[5] Ilias Chalkidis et al. "LEGAL-BERT: The Muppets Straight out of Law School". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, 2020, pp. 2898–2904. DOI: 10.18653/v1/2020.findings-emnlp.261. URL: https://aclanthology.org/2020.findings-emnlp.261/.

[6] Hao Li et al. "Large Language Models in Argument Mining: A Survey". 2025. arXiv: 2506.16383. URL: https://arxiv.org/abs/2506.16383.

[7] Christian Stab and Iryna Gurevych. “Parsing Argumentation Structures in Persuasive Essays”. In: *Comput. Linguistics* 43.3 (2017), pp. 619–659. DOI: 10.1162/COLI_A_00295. URL: https://doi.org/10.1162/COLI%5C_a%5C_00295.

[8] Huihao Jing et al. “MASLegalBench: Benchmarking Multi-Agent Systems in Deductive Legal Reasoning”. 2025. arXiv: 2509.24922. URL: https://arxiv.org/abs/2509.24922.

[9] Taicheng Guo et al. “Large Language Model Based Multi-agents: A Survey of Progress and Challenges”. In: *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*. ijcai.org, 2024, pp. 8048–8057. URL: https://www.ijcai.org/proceedings/2024/890.

[10] Giulia Grundler et al. “Detecting Arguments in CJEU Decisions on Fiscal State Aid”. In: *Proceedings of the 9th Workshop on Argument Mining*. Online and in Gyeongju, Republic of Korea: International Conference on Computational Linguistics, 2022, pp. 143–157. URL: https://aclanthology.org/2022.argmining-1.14.

[11] LangChain AI. *LangGraph: Build Resilient Language Agents as Graphs*. https://github.com/langchain-ai/langgraph. 2024.

[12] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle et al. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[13] Ashish Vaswani et al. “Attention Is All You Need”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 30. 2017. URL: https://arxiv.org/abs/1706.03762.

[14] Jared Kaplan et al. “Scaling Laws for Neural Language Models”. 2020. arXiv: 2001.08361. URL: https://arxiv.org/abs/2001.08361.

[15] Jordan Hoffmann et al. “Training Compute-Optimal Large Language Models”. In: *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. 2022. URL: https://proceedings.neurips.cc/paper_files/paper/2022/hash/c1e2faff6f588870935f114ebe04a3e5-Abstract-Conference.html.

[16] Long Ouyang et al. “Training language models to follow instructions with human feedback”. In: *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. Ed. by Sanmi Koyejo et al. 2022. URL: http://papers.nips.cc/paper%5C_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html.

[17] Jason Wei et al. “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”. In: *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. Ed. by Sanmi Koyejo et al. 2022. URL: http://papers.nips.cc/paper%5C_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

[18] Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. “Are Emergent Abilities of Large Language Models a Mirage?” In: *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. Ed. by Alice Oh et al. 2023. URL: http://papers.nips.cc/paper%5C_files/paper/2023/hash/adc98a266f45005c403b8311ca7e8bd7-Abstract-Conference.html.

[19] John Lawrence and Chris Reed. “Argument Mining: A Survey”. In: *Comput. Linguistics* 45.4 (2019), pp. 765–818. DOI: 10.1162/COLI_A_00364. URL: https://doi.org/10.1162/coli%5C_a%5C_00364.

[20] Jingyun Sun et al. “LawLuo: A Multi-Agent Collaborative Framework for Multi-Round Chinese Legal Consultation”. 2024. arXiv: 2407.16252. URL: https://arxiv.org/abs/2407.16252.

[21] Stephen E. Toulmin. *The Uses of Argument*. Cambridge University Press, 1958.

[22] Douglas Walton, Chris Reed, and Fabrizio Macagno. *Argumentation Schemes*. Cambridge University Press, 2008.

[23] Elena Cabrio and Serena Villata. “Five Years of Argument Mining: A Data-driven Analysis”. In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*. 2018, pp. 5427–5433. URL: https://doi.org/10.24963/ijcai.2018/766.

[24] Kevin D. Ashley. *Artificial Intelligence and Legal Analytics: New Tools for Law Practice in the Digital Age*. Cambridge University Press, 2017.

[25] Raquel Mochales and Marie-Francine Moens. “Argumentation Mining”. In: *Artificial Intelligence and Law* 19 (Mar. 2011), pp. 1–22. DOI: 10.1007/s10506-010-9104-x.

[26] Christian Stab and Iryna Gurevych. “Identifying Argumentative Discourse Structures in Persuasive Essays”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. ACL, 2014, pp. 46–56. DOI: 10.3115/V1/D14-1006. URL: https://doi.org/10.3115/v1/d14-1006.

[27] Andrea Galassi et al. “Neural-Symbolic Argumentation Mining: An Argument in Favor of Deep Learning and Reasoning”. In: *Frontiers Big Data* 2 (2019), p. 52. DOI: 10.3389/FDATA.2019.00052. URL: https://doi.org/10.3389/fdata.2019.00052.

[28] Boyang Liu et al. “Global information-aware argument mining based on a top-down multi-turn QA model”. In: (2023). URL: https://www.sciencedirect.com/science/article/pii/S0306457323001826.

[29] Ruta Liepina et al. “Legal Argument Mining: Recent Trends and Open Challenges”. In: *Proceedings of the First Argument Mining and Empirical Legal Research Workshop co-located with the 20th International Conference on Artificial Intelligence and Law (ICAIL 2025), Chicago, United States, June 20, 2025*. Ed. by Tomás Koref, Lena Held, and Ivan Habernal. Vol. 4089. CEUR Workshop Proceedings. CEUR-WS.org, 2025, pp. 1–14. URL: https://ceur-ws.org/Vol-4089/paper1.pdf.

[30] Prakash Poudyal et al. “ECHR: Legal Corpus for Argument Mining”. In: *Proceedings of the 7th Workshop on Argument Mining*. Ed. by Elena Cabrio and Serena Villata. Online: Association for Computational Linguistics, Dec. 2020, pp. 67–75. URL: https://aclanthology.org/2020.argmining-1.8/.

[31] Deniz Gorur, Antonio Rago, and Francesca Toni. “Can Large Language Models perform Relation-based Argument Mining?” In: *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*. Ed. by Owen Rambow et al. Association for Computational Linguistics, 2025, pp. 8518–8534. URL: https://aclanthology.org/2025.coling-main.569/.

[32] Christina Berghegger et al. “Discovering the Potential of LLMs in Annotating Legal Texts for Argument Mining”. In: *Proceedings of the Second International Workshop on Argumentation and Applications (Arg&App 2025), Melbourne, Australia, November 11, 2025*. Ed. by Oana Cocarascu et al. 2025, pp. 33–44. URL: https://hal.science/hal-05339535/document%5C#page=35.

[33] Morgan A. Gray, Li Zhang, and Kevin D. Ashley. “Generating Case-Based Legal Arguments with LLMs”. In: *Proceedings of the 2025 Symposium on Computer Science and Law, CSLAW 2025, Munich, Germany, March 25-27, 2025*. ACM, 2025, pp. 160–168. DOI: 10.1145/3709025.3712216. URL: https://doi.org/10.1145/3709025.3712216.

[34] Cong Jiang and Xiaolei Yang. “Legal Syllogism Prompting: Teaching Large Language Models for Legal Judgment Prediction”. In: *Proceedings of the Nineteenth International Conference on Artificial Intelligence and Law, ICAIL 2023, Braga, Portugal, June 19-23, 2023*. Ed. by Matthias Grabmair, Francisco Andrade, and Paulo Novais. ACM, 2023, pp. 417–421. DOI: 10.1145/3594536.3595170. URL: https://doi.org/10.1145/3594536.3595170.

[35] Francesco Alfieri et al. “Dynamic Demonstrations Selection for Few-Shot Legal Argument Mining”. In: *Proceedings of the First Argument Mining and Empirical Legal Research Workshop co-located with the 20th International Conference on Artificial Intelligence and Law (ICAIL 2025), Chicago, United States, June 20, 2025*. Ed. by Tomás Koref, Lena Held, and Ivan Habernal. Vol. 4089. CEUR Workshop Proceedings. CEUR-WS.org, 2025, pp. 15–29. URL: https://ceur-ws.org/Vol-4089/paper2.pdf.

[36] Abdullah Al Zubaer, Michael Granitzer, and Jelena Mitrovic. “Performance analysis of large language models in the domain of legal argument mining”. In: *Frontiers Artif. Intell.* 6 (2023). DOI: 10.3389/FRAI.2023.1278796. URL: https://doi.org/10.3389/frai.2023.1278796.

[37] Shunyu Yao et al. “ReAct: Synergizing Reasoning and Acting in Language Models”. In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=WE_vluYUL-X.

[38] Noah Shinn et al. “Reflexion: language agents with verbal reinforcement learning”. In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: https://openreview.net/forum?id=vAElhFcKW6.

[39] Xuezhi Wang et al. “Self-Consistency Improves Chain of Thought Reasoning in Language Models”. In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=1PL1NIMMrw.

[40] Albert Sadowski and Jaroslaw A. Chudziak. "On Verifiable Legal Reasoning: A Multi-Agent Framework with Formalized Knowledge Representations". In: *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*. CIKM '25. ACM, Nov. 2025, pp. 2535–2545. DOI: 10.1145/3746252.3761057. URL: http://dx.doi.org/10.1145/3746252.3761057.

[41] Zerui Chen et al. "LegalGraphRAG: Multi-Agent Graph Retrieval-Augmented Generation for Reliable Legal Reasoning". 2026. arXiv: 2605.28120. URL: https://arxiv.org/abs/2605.28120.

[42] Ziqi Wang and Boqin Yuan. "L-MARS: Legal Multi-Agent Workflow with Orchestrated Reasoning and Agentic Search". 2025. arXiv: 2509.00761. URL: https://arxiv.org/abs/2509.00761.

[43] Jiaxi Cui et al. "Chatlaw: A Multi-Agent Collaborative Legal Assistant with Knowledge Graph Enhanced Mixture-of-Experts Large Language Model". 2024. arXiv: 2306.16092. URL: https://arxiv.org/abs/2306.16092.

[44] Zhitao He et al. "AgentsCourt: Building Judicial Decision-Making Agents with Court Debate Simulation and Legal Knowledge Augmentation". In: *Findings of the Association for Computational Linguistics: EMNLP 2024*. Ed. by Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 9399–9416. DOI: 10.18653/v1/2024.findings-emnlp.549. URL: https://aclanthology.org/2024.findings-emnlp.549/.

[45] Sewon Min et al. "Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?" In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 11048–11064. DOI: 10.18653/v1/2022.emnlp-main.759. URL: https://aclanthology.org/2022.emnlp-main.759/.

[46] Tushar Khot et al. "Decomposed Prompting: A Modular Approach for Solving Complex Tasks". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=_nGgzQjzaRy.

[47] adele-project. *Demosthenes: Argument Mining Corpus for CJEU Decisions on Fiscal State Aid*. https://github.com/adele-project/demosthenes. 2022.

Appendix A

Key Prompt Templates

A.1 Node 1: Argumentative/Non-Argumentative Classification Prompt

```
1
2  You are an expert in legal argumentation mining. Your task
       is to extract argumentative components from a legal
     judgment.
3
4  CRITICAL RULES:
5  1. PROPOSITION-LEVEL SPLITTING: A single grammatical
     sentence may contain multiple argumentative components (
     e.g., a premise followed by a conclusion). Split these
     into separate entries.
6     - Example: "Because the law was followed [PREM], the
         appeal is rejected [CONC]."
7
8  Point: ONLY split a sentence if it contains two components
       with DIFFERENT roles (one PREM and one CONC). Never
     split a sentence into two components of the same role.
9
10    Example of a FALSE split to avoid:
11      Fragment A: "It follows from this that Regulation X
          and Directive Y operate in separate legal spheres"
12      Fragment B: "and that, consequently, measures adopted
           under Regulation X cannot override obligations
          arising exclusively within the scope of Directive Y
          "
13      -> These must be ONE premise.
```

```
14
15 2. SCOPE: Identify ONLY the Court's reasoning and findings
      .
16    - EXCLUDE: Sentences that merely report the submissions
         or contentions of the parties (Salzgitter, the
        Commission, etc.). These are context, not
        argumentative components.
17
18 3. COMPONENT DEFINITIONS:
19    - PREMISE (prem): Evidence, legal rules, or
        interpretations used to justify a ruling.
20      * Note: Legal interpretations used to justify a
          rejection are PREMISES, not final conclusions.
21    - CONCLUSION (conc): The final claim, ruling, or
        decision that follows logically from the premises.
22
23 Respond ONLY with valid JSON:
24 {
25   "sentences": [
26     {"id": 1, "text": "...", "is_argumentative": true},
27     {"id": 2, "text": "...", "is_argumentative": false},
28     ...
29   ]
30 }
```

Listing A.1: System prompt for Argumentative/Non-Argumentative classification agent.

A.2 Node 2: Premise/Conclusion Classification Prompt

```
1
2 You are a legal argumentation expert working with EU Court
     of Justice judgments.
3 For each sentence, determine if it is a:
4 - "prem" (premise): a supporting statement, evidence, fact
    , legal rule, or interpretation used to build an
    argument.
```

```
- "conc" (conclusion): a sentence that DISPOSES OF an argumentative unit -- it formally closes a plea, limb, or ground of appeal with a verdict.

CORE RULE:
Ask: "Does this sentence DISPOSE OF an argumentative unit, or does it DESCRIBE a legal position, fact, or consequence?"

  DISPOSE -> conc   ("must be rejected", "must be dismissed", "must be set aside", "cannot succeed", "committed an error of law in holding...")
  DESCRIBE -> prem  (states a rule, derives a consequence, establishes a fact, states what a party or institution can or cannot do)

Do NOT rely on signal words alone: "consequently", "therefore", "it follows", "must" can appear in both prem and conc.

Example:
[1] The aid falls under Article 4(c) CS.
[2] Consequently, Germany cannot rely on an authorisation under Article 67 CS.
[3] Therefore, the plea must be rejected.

Labels:
[1] -> prem
[2] -> prem
[3] -> conc

Reasoning:
[1] DESCRIBE -- states a legal finding about the aid -> prem.
[2] DESCRIBE -- "consequently" signals a deduction, but the sentence states a consequence about a party's legal position, not a verdict that closes a plea unit -> prem.
[3] DISPOSE -- formally closes the plea with a verdict ("must be rejected") -> conc.
```

```
KEY: "consequently" in [2] and "therefore" in [3] are both signal words, but [2] describes and [3] disposes. The logical act matters, not the signal word.

Respond ONLY with valid JSON:
{
  "classifications": [
    {"id": 1, "role": "prem"},
    {"id": 2, "role": "conc"},
    ...
  ]
}
```

Listing A.2: System prompt for Premise/Conclusion classification agent.

A.3 Node 3: F/L/FL Binary Classification Prompt

```

Legal argumentation expert for EU Court of Justice judgments.
Classify each premise as F, L, or FL.

CORE TEST: "Is this sentence asserting something specific to THIS case, or stating a general legal norm true beyond this case?"

F -- Case-specific finding (even if a legal article is mentioned):
  • Reports what the Court found/determined in THIS case
  • States what a party can/cannot do IN THIS SITUATION
  • KEY SIGNAL: names a specific party, "the aid", "the decision", "this plea"
  ✓ "The aid fell within the scope of Article 4(c) CS." -> F (article just locates the finding)
  ✓ "Germany cannot rely on an authorisation under Article 67 CS." -> F
  ✓ "It must be held that the second limb is unfounded." -> F

```

```
L -- General legal norm (true in any case):
  • States what a rule/article/treaty says or means IN
     GENERAL
  • KEY SIGNAL: could appear in a legal textbook with no
     case context
  ✓ "Article 4(c) CS prohibits aid in any form whatsoever
     ." -> L
  ✓ "The ECSC Treaty does not distinguish between new and
     existing aid." -> L
  EXCEPTION: naming an institution ("Commission", "Court
     of First Instance")
  does NOT make it F if the sentence asserts a general
     interpretive principle.
  ✓ "Commission practice cannot affect treaty
     interpretation." -> L (general principle)

FL -- RARE. Legal norm APPLIED to a specific fact; both
   parts load-bearing:
  ✓ "Given that Article 1 prohibited all steel aid, the
     Commission could not
     withdraw the 1971 Decision." -> FL
  X "The aid fell within Article 4(c) CS." -> F not FL (
     article just referenced)

DECISION:
1. Names specific party/fact + no general norm stated -> F
2. General norm, true beyond this case -> L
3. Both (1) and (2) load-bearing in same sentence -> FL

Respond ONLY with valid JSON:
{"classifications": [{"id": 1, "premise_type": "F", "
   reason": "..."}]}
```

Listing A.3: System prompt for one voting run of the F/L/FL classification agent.

A.4 Node 4: Scheme Classification Prompt

```

```

```
Legal argumentation expert for EU Court of Justice judgments.

SCHEME DEFINITIONS:

Rule -- the sentence cites a SPECIFIC article, treaty provision, or code BY NUMBER OR NAME and states what it says, requires, or implies.
  This includes sentences that derive consequences FROM a named article.
  ✓ "Article 4(c) CS prohibits aid in any form whatsoever." -> Rule
  ✓ "Article 305(1) EC states that the EC Treaty shall not affect the ECSC Treaty." -> Rule
  ✓ "It follows from this [citing Article 305(1)] that the EC Treaty and the ECSC Treaty are independent treaties." -> Rule  <- DERIVES FROM named article = Rule
  ✓ "Article 6 of the Third Steel Aid Code provided an obligation to notify." -> Rule
  KEY: if a specific article/code/provision is named anywhere in the sentence -> Rule.

Prec (Precedent) -- the sentence cites a specific past court decision BY CASE NAME or NUMBER.
  ✓ "Joined Cases 188/80 to 190/80 France v Commission [1982] ECR 2545" -> Prec
  ✓ "Case 328/85 Deutsche Babcock [1987] ECR 5119" -> Prec
  ✓ "Falck and Acciaierie di Bolzano v Commission, paragraph 117" -> Prec
  KEY: must name a specific case. "The Court has held..." alone -> NOT Prec.
  NOTE: a sentence can be BOTH Prec and Rule if it cites both a case and an article.

Itpr (Interpretation) -- interprets the meaning or scope of a legal norm WITHOUT citing a specific
  article by number or name. The interpretation stands on its own without a
  named legal source.
```

```
  ✓ "The ECSC Treaty does not distinguish between new aid and existing aid." -> Itpr (no specific article cited -- general interpretation of treaty structure)
  X "It follows from Article 305(1) EC that..." -> NOT Itpr -> Rule (Article 305(1) is cited by name -> this is Rule)
  HARD RULE: if any specific article/provision is named -> Rule, NOT Itpr.
  Itpr = interpretation with NO named legal source.

Aut (Authority) -- cites an expert, Commission communication, or authoritative body for its VIEW.
  ✓ "As the Commission submits, Article 6 provided..." -> Aut
  KEY: "as [authority] submits/states/argues" signals Aut.

Class (Classification) -- classifies a situation under a legal category.
  ✓ "The aid falls within the category of existing aid." -> Class

Princ (Principle) -- invokes a general legal principle by name.
  ✓ "the fundamental requirement of legal certainty" -> Princ
  ✓ "the principle of proportionality" -> Princ

DECISION RULE -- apply in order:
  1. Does the sentence cite a specific article/provision by name? -> Rule (always)
  2. Does the sentence cite a specific case by name? -> add Prec
  3. Does the sentence interpret WITHOUT any named article? -> Itpr
  4. Check for Aut, Class, Princ independently

Respond ONLY with valid JSON:
{"classifications": [{"id": 1, "schemes": ["Rule"], "reason": "..."}]}
```

Listing A.4: System prompt for Scheme classification agent.

A.5 Node 5: Link Detection Prompt

```

You are an expert in legal argumentation mining for EU
   Court of Justice judgments.

{FEW_SHOT_PREM_CONC}

TASK: For a given CONCLUSION, evaluate each PREMISE as a
   potential direct support or attack.

RULES:
1. DIRECTNESS: One logical step only. Reject if
   intermediates are required.
2. DIRECTION: Always PREMISE -> CONCLUSION. Never reverse.
3. DEFAULT TO FALSE: Most premises do not directly link to
    a given conclusion.
4. CONFIDENCE -- based on explicit textual evidence:
   "high": Link signalled EXPLICITLY ("it follows", "
      consequently", "therefore", "however", "must be
      pointed out").
   "medium": Link is IMPLICIT but logically clear -- no
      signal word, but content makes dependency clear.
   "low": SPECULATIVE -- topic similarity only. Prefer
      has_link=false.

Respond ONLY with valid JSON:
{{
  "conclusion_id": <int>,
  "premise_evaluations": [
    {{
      "premise_id": <int>,
      "reasoning": "What does premise establish? What does
          conclusion claim? Direct?",
      "has_link": true/false,
      "link_type": "Support" | "Attack" | null,
```

```
      "confidence": "high" | "medium" | "low"
    }}
  ]
}}
```

Listing A.5: System prompt for link detection (Premise - Conclusion).

```

You are an expert in legal argumentation mining for EU
   Court of Justice judgments.

{FEW_SHOT_PREM_PREM}

TASK: Evaluate whether PREMISE A directly or indirectly
   supports or attacks PREMISE B within the same
   argumentative section.

RULES:
1. DIRECT LINK: B explicitly deduces from or challenges A
   in one step.
   Support signals: "it follows from this", "consequently
      ", "therefore".
   Attack signals:  "however", "must be pointed out", "
      misreading", "contrary to".
   -> has_link=true, link_type="Support" or "Attack",
      confidence=high

2. COLLECTIVE LINK: A is one of multiple premises that
   together support or attack B. B may not deduce from A
   alone, but A contributes to B's reasoning.
   Support signals: "in the light of the foregoing", "
      taken together", "consequently" after several
      premises established.
   Attack signals:  "however", "contrary to the foregoing
      ", "nevertheless","that reasoning is flawed" after
      prior premises established.
   -> has_link=true, link_type="Support" or "Attack",
      confidence=medium

3. INDEPENDENT: related by topic but no logical dependency
   .
```

```
   -> has_link=false

4. DEFAULT TO FALSE for genuinely independent premises.

IMPORTANT:
A link must be DIRECT -- it must hold in a single step, without passing
through an intermediate premise.
- Do NOT infer transitive links: if [1] -> [2] and [2] -> [3], do NOT add [1] -> [3], UNLESS [1] independently supports/attacks [3] on its own (i.e. not only by way of [2]).
- A premise that is one of several DIRECT supporters of B (Rule 2, collective) is still a one-step link and IS allowed. "Direct" means one step, not "sole".

Respond ONLY with valid JSON:
{{
  "prem_prem_evaluations": [
    {{
      "source_id": <int>,
      "target_id": <int>,
      "reasoning": "...",
      "has_link": true/false,
      "link_type": "Support" | "Attack" | null,
      "confidence": "high" | "medium" | "low"
    }}
  ]
}}
```

Listing A.6: System prompt for link detection (Premise - Premise).

A.6 Few-shot Example for Link Detection Prompt

```

--- FEW-SHOT EXAMPLES ---

Example 1 -- SUPPORT:
```

```
Premise: [P] "Article 4(c) CS explicitly prohibits State aid in any form whatsoever, without distinction between individual and general aid."
Conclusion: [C] "Article 4(c) CS applies even under a general aid scheme not specific to the steel sector."
Reasoning: One direct step -- premise establishes scope, conclusion extends it.
Result: {"has_link": true, "link_type": "Support", "confidence": "high"}

Example 2 -- ATTACK:
Premise: [P] "The EC Treaty and the ECSC Treaty are independent instruments; EC secondary legislation cannot produce effects in ECSC scope."
Conclusion: [C] "The 1971 Decision under EC Treaty implicitly authorised ECSC aid."
Reasoning: Independence directly negates cross-authorisation. One step.
Result: {"has_link": true, "link_type": "Attack", "confidence": "high"}

Example 3 -- FALSE LINK (transitivity trap):
Premise: [P] "The Commission had knowledge of Salzgitter's aid since 1988."
Conclusion: [C] "The cross-appeal must therefore be dismissed in its entirety."
Reasoning: Three missing intermediates required: knowledge -> legal certainty -> recovery decision legality -> dismissal. Multi-hop jump, not direct.
Result: {"has_link": false, "confidence": "high"}
--- END EXAMPLES ---
```

Listing A.7: Few-shot prompt for link detection (Premise - Conclusion).

```

--- FEW-SHOT EXAMPLES ---

Example 1 -- DIRECT LINK (Support):
Premise A: [P3] "Article 305(1) EC states that EC Treaty provisions shall not affect ECSC Treaty provisions."
Premise B: [P4] "It follows from this that the EC Treaty
```

```
   and the ECSC Treaty are independent treaties."
Reasoning: "It follows from this" -- explicit one-step
   deduction. Support.
Result: {"has_link": true, "link_type": "Support", "
   confidence": "high"}

Example 2 -- COLLECTIVE SUPPORT:
Premise A: [A1] "Article 4(c) CS prohibits State aid
   without distinguishing between individual aid and aid
   under a general scheme."
Premise B: [A9] "In the light of the foregoing, it must
   consequently be held that Article 4(c) CS applies to
   State aid under a general scheme."
Reasoning: A9 synthesizes multiple prior premises
   including A1. A1 contributes.
Result: {"has_link": true, "link_type": "Support", "
   confidence": "medium"}

Example 3 -- ATTACK (premise challenges another premise):
Premise A: [D1] "It is not sufficient that the initial
   promotion scheme was financed by a compulsory levy to
   enable it to be regarded as financed through State
   resources."
Premise B: [D2] "However, it must be pointed out that that
    argument is based on a misreading of that judgment."
Reasoning: D2 directly challenges the legal reading
   asserted in D1. Attack.
Result: {"has_link": true, "link_type": "Attack", "
   confidence": "high"}

Example 4 -- FALSE LINK (independent rules):
Premise A: [P8] "The ECSC Treaty does not distinguish
   between new and existing aid."
Premise B: [P8bis] "Article 4(c) CS prohibits aid in any
   form whatsoever."
Reasoning: Both are independent legal rules. Neither
   deduces from the other.
Result: {"has_link": false, "confidence": "high"}

Example 5 -- INTERMEDIATE SUPPORT:
```

```
Premise A: [A3] "Articles 4 CS and 67 CS cover two distinct areas: the first prohibits certain Member State actions under Community jurisdiction; the second prevents competition distortions from residual Member State powers."
Premise B: [A4] The Court deduces from this that Article 67 CS covers only general economic/social measures and measures applied to industries other than coal and steel."
Premise C: [A10] Consequently, Article 4(c) CS applies to aid paid to coal/steel firms even under a scheme that is not specific to that sector."
Reasoning: A3 Support A4 Support A10 Support. Note that the link connection MUST be one step therefore because of A4 is intermediate support, A3 has no link with A10.
Result [A3] Support [A4]: {"has_link": true, "link_type": "Support", "confidence": "high"}
Result [A4] Support [A10]: {"has_link": true, "link_type": "Support", "confidence": "high"}
Result [A3] Support [A10]: {"has_link": false, "confidence": "high"}
--- END EXAMPLES ---
```

Listing A.8: Few-shot prompt for link detection (Premise - Premise).