Team Sophisma at Touché: Transformer-Based Fallacy Detection with Enhanced Argument Representations

Touché at CLEF 2026

İrem Batıgün[1,*], Yiğitalp Öztemir[1]

[1]*İstinye University, İstanbul, Turkey*

Abstract

In this study, Team Sophisma developed a modular and structure-oriented natural language processing system for the Touché CLEF 2026 Fallacy Detection task. Our work focuses specifically on two subtasks: binary fallacy detection, which determines whether a given text contains fallacies, and multi-class fallacy classification, which aims to assign instances containing fallacies to specific fallacy types. Our system utilizes enriched text representations containing claim and supporting argument information. For this purpose, a reusable pipeline has been designed for data preprocessing, label transformation, training/validation separation, sentiment analysis-based additional feature extraction, and the generation of model outputs in a competition format. Our main approach is based on a transformer-based classifier fine-tuned on enriched argument representations. In addition, experimental components evaluating feature-based machine learning approaches for comparative analysis have been developed. This paper reports the system architecture, data preparation process, models used, and submitted outputs.

Keywords

CLEF 2026, Touché, fallacy detection, argument mining, transformer models, RoBERTa, XGBoost, sentiment features

1. Introduction

Fallacy detection can be considered a significant and challenging problem in the field of argument mining. Just because an argument appears persuasive on the surface doesn't mean it's logically valid or reliable. Especially in political discussions, social media content, news commentary, and online debates, fallacious arguments, based on flawed reasoning can be formally very similar to correct and coherent arguments. Therefore, not only the subject matter, emotional tone, or words used in a text should be considered, but also the relationship between the claim and the justification, the assumptions on which the argument is based, and the persuasive strategy. In this respect, fallacy detection is not only a text classification problem but also a natural language processing problem requiring an understanding of argument structure and reasoning [1, 2, 3].

The Touché at CLEF 2026 Fallacy Detection task aims to enable models to distinguish between fallacious and non-fallacious arguments and to perform more detailed fallacy type classification for fallacious examples. Furthermore, semantic similarities can be found between different fallacy types; for example, classes such as authority, population, or tradition may exhibit similar characteristics in terms of whether the argument is based on an external source, social acceptance, or traditional belief. This can lead to the inadequacy of methods relying solely on word-level cues and necessitate more context-sensitive representation methods [1, 2].

This study presents the fallacy detection system developed by Team Sophisma within the scope of Touché at CLEF 2026. Our system is based on an enhanced argument representation approach that combines text, claim, and supporting argument components instead of directly using raw text. This representation aims to enable transformer-based classifiers to better learn not only the superficial

Supervisor: Andres Montoro-Montarroso

CLEF 2026 Working Notes, 21–24 September 2026, Jena, Germany

*Corresponding author.

irembatigun@hotmail.com (: Batıgün)

linguistic features of the argument but also the claim-support relationship. Additionally, a modular and structure-oriented experimental pipeline was developed within the project; data preparation, preprocessing, model training, prediction generation, and evaluation steps were separated. The main contributions of this study are the design of a reusable NLP pipeline for the Touché Fallacy Detection task, the examination of the use of enhanced argument representation, and the evaluation of the performance of transformer-based approaches in fallacy detection and fallacy classification tasks.

1.1. Contributions

- We developed a modular and structure-oriented NLP pipeline that separates the data preparation, preprocessing, model training, prediction generation, and evaluation steps for the Touché 2026 Fallacy Detection task.
- Instead of directly using raw text, we created an argument-structure-sensitive input representation for transformer models by combining the enhanced text field with the claim and support components of the argument.
- We fine-tuned transformer-based sequence classification models for fallacy detection and fallacy type classification tasks and evaluated the system's performance in different tasks.
- We developed supporting analysis modules such as exploratory data analysis, sentiment analysis, feature extraction, and entity masking to better understand the dataset and model behavior.
- By examining the model's behavior on different fallacy types, we discussed error patterns that occur especially in semantically close classes.

2. Task and Dataset

The Touché 2026 Fallacy Detection task consists of three subtasks: fallacy detection, fallacy classification, and argument scheme classification. Participants are given a short source text and extracted arguments, and the arguments are the main units of analysis [1, 2].

Table 1
Touché 2026 Fallacy Detection subtasks and expected outputs.

Subtask	Input assumption	Prediction target	Labels / output
Subtask 1: Fallacy Detection	Any argument	Whether the argument is fallacious	fallacy / non-fallacy
Subtask 2: Fallacy Classification	Fallacious arguments	Specific fallacy type	authority, black-white, `hasty_generalization`, natural, population, `slippery_slope`, tradition, `worse_problems`
Subtask 3: Argument Scheme Classification	Non-fallacious arguments	Goal-basis scheme label	practical-internal, practical-external, epistemic-internal, epistemic-external

2.1. Dataset Fields Used

The Touché Fallacy Detection dataset contains different text representations and label fields for each instance. The argument fields contain a claim and one or more supports statements for each instance. The training data also includes `fallacy_exists` for fallacy detection, `fallacy_type` for fallacy type classification, and `argument_goal` and `argument_basis` fields under classification for argument scheme information [2, 3]. In our main experimental setup, raw Reddit context fields were not used directly as model input. That is, the `text_raw_title` and `text_raw_parent` fields were not added to the input during preprocessing. Instead, the `text_field` value was set to `text_enhanced` and the

argument_field value to argument_enhanced. Therefore, the model input for each instance was created by combining the enhanced text field with the argument components.

Table 2
Dataset fields and their role in the submitted systems.

Field group	Fields	How Team Sophisma used it
Raw context	text_raw, text_raw_parent, text_raw_title	Not used as direct input in the main runs. Title and parent context were disabled in preprocessing and were mainly kept for inspection and EDA.
Base representation	text_base, argument_base	Used for preliminary inspection and EDA. The main reported preprocessing pipeline did not use the base representation, although the pipeline can be configured to do so.
Enhanced representation	text_enhanced, argument_enhanced	Used as the main input representation. We concatenated enhanced text, claim, and support statements into a single model input sequence.
Labels	fallacy_exists, fallacy_type, classification	fallacy_exists was used for binary fallacy detection. fallacy_type was used for multiclass fallacy classification. Classification was available for scheme classification, but this task was disabled in the main configuration.
Additional metadata	sentiment scores, NER masks, topic features, etc.	Used for analysis and feature-based experiments. Transformer models used only the constructed text input, while the XGBoost pipeline supports sentiment, stylistic, embedding, UMAP, and clustering features.

2.2. Preprocessing

During the preprocessing phase, empty lines were skipped, and each line was processed as a separate JSON object. Training data was loaded from the touchefallacy_2026_train.jsonl file, and test data from the touchefallacy_2026_test_task.jsonl file. Separate train.jsonl, validation.jsonl, test.jsonl, and summary.json files were created for each task.

In our main experimental setup, enhanced fields were used as model input. In the configuration, the text_field value was set to text_enhanced, and the argument_field value was set to argument_enhanced. Reddit title and parent comment fields were not added to the model input. For each instance, the claim and supports fields within text_enhanced and argument_enhanced were converted into a single input text. When there were multiple support statements, they were separated with the [SEP] tag.

Text normalization was kept limited. Some textual fields in the JSONL files were missing or had a null value. After loading the data in Python, these values were represented as None. We converted these missing values to empty strings so that the word "None" would not be inserted into the model input. In addition, we standardized line breaks, removed extra spaces, and eliminated extra leading and trailing spaces. Aggressive cleanup operations such as lowercasing, stopword removal, or stemming were not applied to not to break any important structure of the enchanced texts. If a field like claim or supports was empty, the relevant section was not included in the final input text.

Labels were mapped on a task basis. For fallacy detection, the fallacy_exists field was used; a value of 1 was converted to fallacy, and a value of 0 to non-fallacy. For fallacy classification, only examples containing fallacy were used, and the fallacy_type field was taken as the target label. Eight classes were used in this task: authority, black-white, hasty_generalization, natural, population, slippery_slope, tradition, and worse_problems. The code also normalizes the blackwhite label to the black-white format.

As a result of preprocessing, 750 training, 188 validation, and 233 test examples were created for the fallacy detection task; and 372 training, 93 validation, and 233 test examples were created for the fallacy classification task. The maximum sequence length was applied in the transformer tokenizer phase, not the preprocessing phase. In the RoBERTa BF16 model training, `max_length=512` was used, and long texts were truncated during tokenization.

3. System Overview

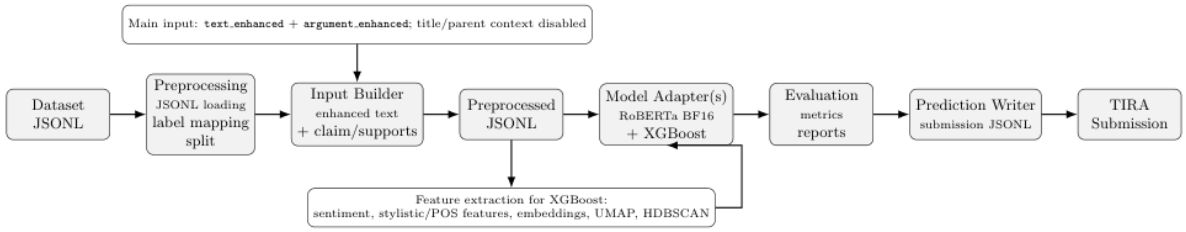

Figure 1: Overview of Team Sophisma's Touché 2026 pipeline from JSONL data loading to preprocessing, model training/evaluation, and run-file generation for TIRA.

Table 3
System pipeline overview.

Stage	Purpose	Implementation Details
Data loading	Read Touché JSONL files	Loaded train/test JSONL records and mapped fields into an internal sample structure.
Preprocessing	Create task-specific splits	Generated separate train, validation, and test JSONL files using a stratified 80/20 split with seed 42.
Input construction	Build model-ready text	Used `text_enhanced` plus claim/supports from `argument_enhanced`; supports were separated with `[SEP]`.
Metadata and features	Support analysis and baseline features	Used sentiment, stylistic/POS, embeddings, UMAP, and HDBSCAN features for XGBoost and analysis; not for RoBERTa input.
Modeling	Train classifiers	Submitted system used RoBERTa BF16; feature-based XGBoost was implemented as a baseline.
Evaluation	Validate performance	Reported accuracy, macro-F1, per-class precision/recall/F1, classification reports, and confusion matrices.
Prediction writing	Create run files	Exported TIRA-compatible JSONL predictions with task, id, label, tag, and `system_description` fields.

4. Methods

Two models were developed in this study. Our main system is built on a Hugging Face-based transformer sequence classification model, and the RoBERTa model for the submitted run was fine-tuned with BF16 mixed-precision training. In addition, a feature-based XGBoost classifier was also supported within the pipeline for comparison purposes. Both approaches used data pods generated from the same preprocessing outputs.

4.1. RoBERTa BF16 Transformer Model

A transformer based sequence classification model was used for the main submitted run. On the code side, the model is loaded via a Hugging Face `AutoModelForSequenceClassification` adapter. And, the same structure is designed to work with different transformer checkpoints; however, the system we submitted is RoBERTa-based, and BF16 was used during training. The text field created during the preprocessing phase was used as the model input.

Labels were converted to numerical values from the training data using LabelEncoder. In the fallacy detection task, the model distinguished between two classes: fallacy and non-fallacy. In the fallacy classification task, only examples containing fallacies were used, and the model was trained to predict one of eight fallacy types. The maximum sequence length for transformer training was set to 512. Long inputs were truncated during the tokenizer phase, and examples within the batch were dynamically padded using `DataCollatorWithPadding`. Training utilized a learning rate of 2e-5, batch size of 8, epoch count of 10, weight decay of 0.01, warmup ratio of 0.1, and a cosine learning-rate scheduler. Training and evaluation were performed at the end of each epoch, and accuracy and macro-F1 scores were calculated on the validation set. The random seed value was set to 42.

Table 4
Transformer model training configuration.

Parameter	Value
Model checkpoint	roberta-base fine-tuned as a Hugging Face sequence classification model
Input field	Constructed text field from `text_enhanced` + `argument_enhanced.claim` + `argument_enhanced.supports`. Title and parent context were not included.
Maximum sequence length	512 tokens
Batch size	8 per device for training and evaluation
Epochs	10
Learning rate	2e-5
Precision	BF16 mixed precision (`bf16=True`)
Class imbalance handling	No explicit class weighting or resampling was applied in the transformer run. The train/validation split was stratified by label.
Validation strategy	Stratified hold-out validation split with `validation_size=0.2` and random seed 42

4.2. Feature-Based XGBoost Baseline

A feature-based XGBoost model was also implemented within the pipeline. The purpose of the XGBoost model is to compare the transformer based approach with a more classical and interpretable machine learning method. This model was trained using numerical feature vectors extracted for each example, rather than directly using transformer tokenization. This is done through each instance was represented with a fixed-length numerical feature vector extracted from the `text_enhanced` field. The feature vector included basic text statistics, part-of-speech ratios, sentiment scores, sentence embedding features, UMAP features, and HDBSCAN-based cluster features. Basic statistics consisted of character count, word count, average word length, sentence count, question mark count, and exclamation mark count. POS ratios, including noun, verb, adjective, adverb, pronoun, proper noun, and punctuation ratios, were calculated with spaCy. Sentiment was represented using `sentiment_positive` and `sentiment_negative` scores.

For sentiment features, 384 dimensional embeddings were extracted using the `sentence-transformers/all-MiniLM-L6-v2` model, these embeddings were reduced to 2 and 5-dimensional representations using UMAP, and cluster/outlier features were generated using HDBSCAN.

The XGBoost classifier was trained with 150 estimators, a maximum depth of 6, a learning rate of 0.05, and 42 random seeds. For binary classification, the binary logistic objective was used, and for multi class classification, the multi softprob objective was used. Model validation was evaluated on the set using accuracy and macro F1 scores.

4.3. Topic and metadata features

The presented RoBERTa BF16 model is solely text-enhanced, generated from claims and supports. Sentiment scores, NER masks, UMAP coordinates, or clustering information were not added to the RoBERTa input. In contrast, our XGBoost base model utilized custom digital features, including `sensitive_positive` and `sensitive_negative`, stylistic features, POS ratio features, sentence-transformer embedded vectors, UMAP projections, and HDBSCAN cluster/outlier features. Sentiment scores were derived from the input text itself, not from class labels, to prevent label leakage. The EDA and entity masking modules were primarily used for dataset analysis and interpretation.

4.4. Training, Evaluation, and Prediction Generation

The original labeled training file was split into a training set and a validation set. We used 80/20 split with random seed 42, so that the label distribution was preserved in both splits. Model training was run only on `train.jsonl`, while `validation.jsonl` was used for model selection and error analysis. Therefore, the accuracy, precision, recall, and macro-F1 scores reported in Section 7 are validation scores evaluated on `validation.jsonl`. Since the test data did not contain labels, the test set was used only in the prediction generation phase.

Model predictions were saved in JSONL format. Each prediction includes the task, id, label, tag, and `system_description` fields. The tag field is tagged as "enhanced" to indicate that enhanced preprocessing was used. These output files were generated according to the TIRA submission format [4].

5. Experimental Setup

Table 5
Experimental environment and reproducibility details.

Item	Value
Programming language	Python 3.12
Main libraries	PyTorch, transformers, accelerate, scikit-learn, xgboost, pandas, numpy, spaCy, sentence-transformers, UMAP, matplotlib, seaborn, PyYAML, tqdm
Hardware	4050 Laptop 6 GB GDDR6 Vram / i7-13620H 2.40Ghz / 16gb DDR4 ram
Random seed	42
Repository	https://github.com/Team-Sophisma/touche-fallacy-pipeline
Evaluation metrics	precision, recall, F1-score, macro-F1, confusion matrix

Table 6
Submitted Touché runs.

Run ID / filename	Subtask	Tag	Model	Input fields
RoBERTa-BF16	`fallacy_detection`	Enhanced	RoBERTa BF16	Enhanced
RoBERTa-BF16-Class	`fallacy_classification`	enhanced	RoBERTa BF16	Enhanced

6. Submitted Runs

7. Results

Our transformer-based architecture model (RoBERTa) as well as the baseline feature-based model (XGBoost) were tested for their performance in Fallacy Detection and Fallacy Classification tasks. We consider the Macro-F1 score as our primary evaluation metric, which is also the official metric of the shared task.

On the Fallacy Detection (a binary classification task), our fine-tuned RoBERTa model achieved very good results with an Accuracy (which is mathematically equivalent to Micro-F1) of 0.872 and Macro F1 of 0.872. However, the XGBoost model, which was based on numerical feature vector composed of manually crafted features such as text statistics, POS ratios, sentiment scores, MiniLM sentence embeddings, UMAP projections, and HDBSCAN features, performed much worse with an Accuracy of 0.654 and Macro F1 of 0.654.

With regard to the Fallacy Classification (classification into multiple classes, involving 8 different types of fallacies) task, there was an even bigger disparity in the performances of both approaches. RoBERTa obtained almost perfect scores, attaining an Accuracy of 0.978 and a Macro F1-Score of 0.980. On the other hand, the XGBoost algorithm performed rather poorly in this specific task, having an Accuracy of 0.502 and a Macro F1-Score of 0.494.

This remarkable disparity clearly illustrates the significance of deep contextual embeddings.

7.1. Validation Results

Table 7
Validation performance of compared systems.

Model / run	Subtask	Precision	Recall	Accuracy	Macro-F1	Notes
RoBERTa BF16	Fallacy Detection	0.878	0.873	0.872	0.872	Transformer fine-tuning with enhanced representations.
XGBoost	Fallacy Detection	0.655	0.654	0.654	0.654	Baseline using extracted features.
RoBERTa BF16	Fallacy Classification	0.982	0.981	0.978	0.980	Highly distinct contextual separation achieved.
XGBoost	Fallacy Classification	0.502	0.507	0.502	0.494	Baseline struggled with multi-class granularity.

7.2. Official Results

Table 8
Official Touché Results

Run	Subtask	Official precision	Official recall	Official F1	Rank
RoBERTa-BF16	Task 1 — enhanced	0.914	0.914	0.914	6
RoBERTa-BF16-Class	Task 2 — enhanced	0.974	0.972	0.972	3

7.3. Confusion Matrix and Class-level Analysis

Fallacy Detection (Binary Classification): An examination of the confusion matrix associated with the fallacy detection binary subtask is revealing of an intriguing pattern with respect to the two class labels fallacy and non-fallacy. Though the recall of the fallacy category is high at 0.935, many of the items in the non-fallacy category have been mistakenly classified into the fallacy category as false positives. It is evident, therefore, that the dataset purposely includes emotionally and/or logically persuasive claims

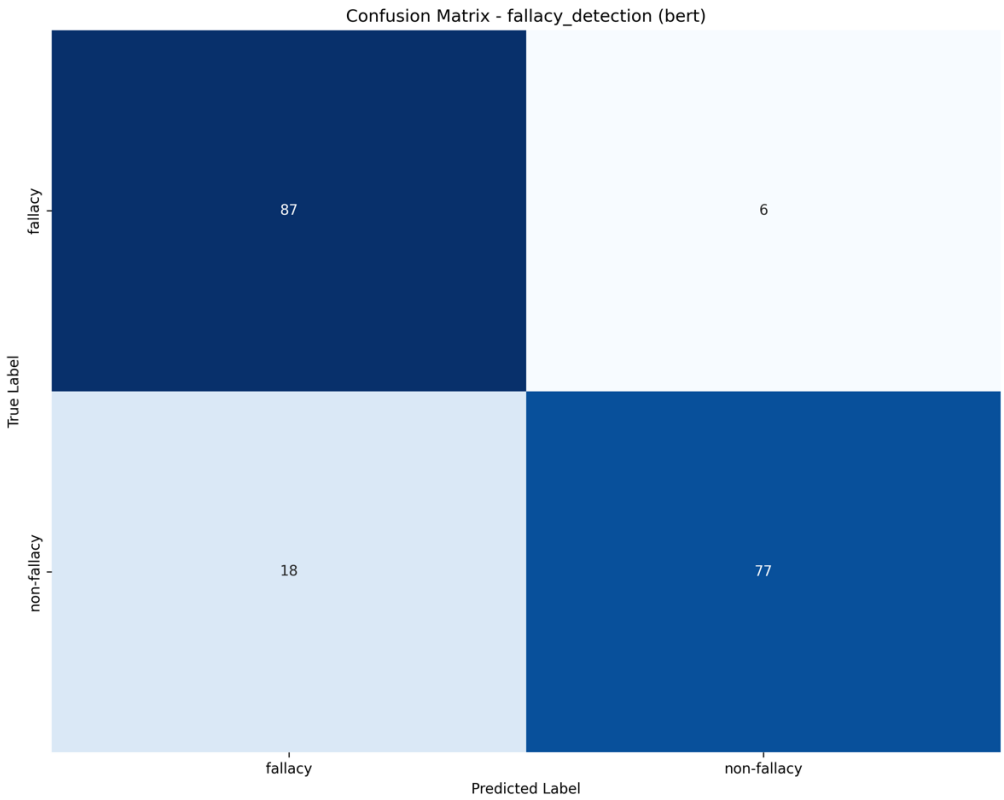

Figure 2: Confusion Matrix - `fallacy_detection` (bert).

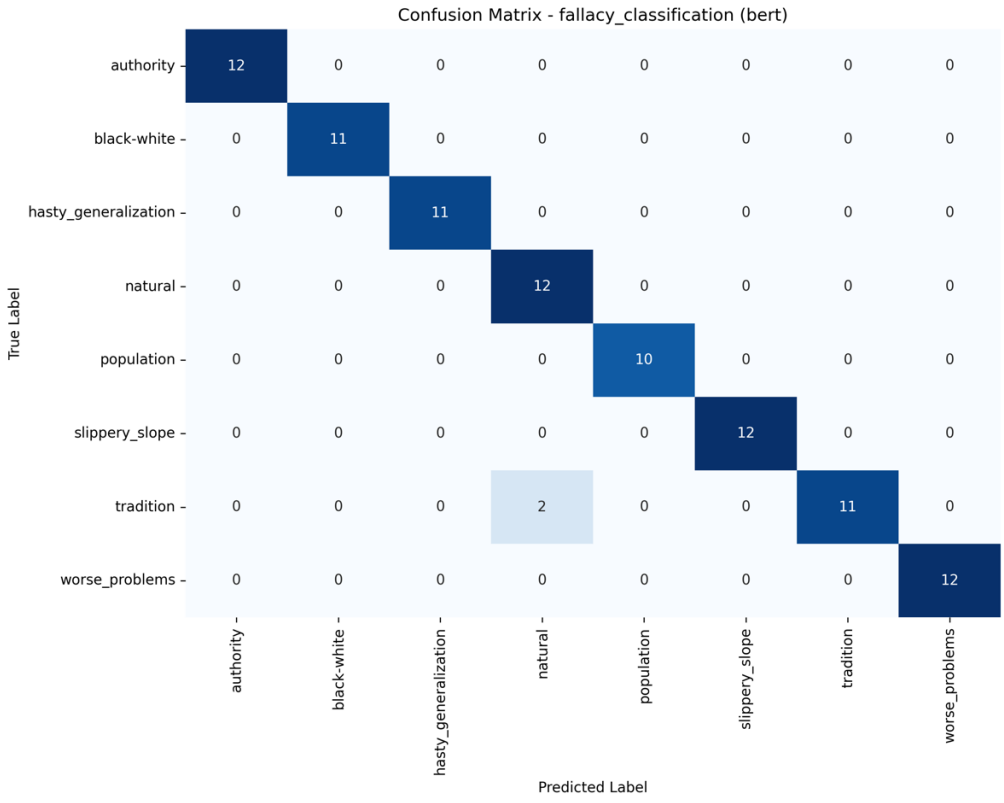

Figure 3: Confusion Matrix - `fallacy_classification` (bert).

that mimic logical fallacies in nature. The RoBERTa model has occasionally mistook a strong argument for a logical fallacy due to its contextually intelligent classification capabilities.

Classification of Fallacy Types (Multi-class Classification): It can be seen from the confusion matrix for the multi-class classification task that the fine-tuned RoBERTa model is a highly discriminative tool. Most of the classes, including 'authority', 'hasty_generalization', and 'worse_problems', have practically no instances where confusion arises, resulting in a virtually perfect diagonal on the matrix. It means that for the statements identified as being fallacies, their type is defined by very distinctive linguistic features.

However, another confusion arises regarding the classification of the appeal to nature and the appeal to tradition fallacies. It appears that the model sometimes mislabels the argumentation that is based on an appeal to tradition as belonging to the appeal to nature fallacy. There are reasonable grounds for this semantic confusion because, similarly to the previous fallacy, the appeal to tradition fallacy is based on the idea that what was established throughout time or what is inherent is automatically right and better.

8. Error Analysis

Table 9
Error Analysis

Observed error pattern	Example / class pair	Likely reason	Possible fix
Confusion between similar fallacy types	natural vs. tradition	Precision for natural dropped to 0.85, while recall for tradition was 0.84. Both appeal to long-standing, historical, or "inherent" concepts, causing semantic overlap.	Add more targeted examples distinguishing between "appeal to nature" and "appeal to tradition".
False positives in fallacy detection	non-fallacy resembling fallacy	The model has high recall (0.93) but lower precision (0.83) for the fallacy class. The dataset intentionally contains valid arguments resembling fallacies.	Train with contrastive examples or add a rationale-aware classifier to distinguish nuanced logical structures.
Weak performance on structurally complex classes	black-white & population (XGBoost)	The baseline XGBoost F1 dropped to 0.24 for black-white and 0.26 for population. These classes require deep logical context rather than static keywords.	Transition fully to contextualized embeddings (transformers).

9. Discussion

The experimental results confirm our hypothesis that transformer-based models are very effective at detecting and classifying logical fallacies when given enhanced argument representations.

During development and training phase we encountered numerical instability (NaN loss values) when training with standard mixed-precision (FP16). To mitigate these hardware level arithmetic overflows,

we moved to Bfloat16 (BF16) precision without compromising the efficiency of training.

The exceptional performance of the RoBERTa model on the multi-class classification task (98.0% Macro-F1) is particularly striking. This means that when a statement is labeled as fallacious, the particular sub-type of fallacy has very distinguishable vocabulary, phrasing or structural markers that the attention mechanisms of the transformer can readily map. Binary detection was harder (87.2% Macro-F1) as distinguishing between a poorly worded valid argument and a real logical fallacy requires a deeper level of reasoning that is still a challenge for current language models.

This clear difference with respect to the XGBoost baseline demonstrates the limits of traditional non-contextual natural language processing techniques. Logical fallacies are structural and contextual in nature, and cannot reliably be detected by counting word frequencies or measuring sentence-level sentiment.

10. Conclusion

In this paper, Team Sophisma presented a modular pipeline for the CLEF 2026 Touché Fallacy Detection task. We leveraged a fine-tuned RoBERTa model evaluating on enriched textual representations to identify and classify fallacies. Our system achieved highly competitive results, with an 87.2% Macro F1-score in binary fallacy detection and a 98.0% Macro F1-score in multi-class fallacy classification.

Our findings demonstrate that transformer-based architectures significantly outperform traditional feature-based machine learning approaches like XGBoost in understanding the nuances of argumentative flaws.

Declaration on Generative AI

During the preparation of this work, the authors used generative AI tools to take help with the writing process, including grammar correction, structural editing, and the organization of report sections. Generative AI was also used to improve the clarity of explanations. The experimental results, validation metrics, confusion matrices, and submitted prediction files were produced by the authors' own implemented pipeline. After using generative AI tools, the authors carefully reviewed, verified, and edited the content.

References

[1] J. Kiesel, M. Feger, T. Hagen, S. Heineking, M. Heinrich, M. Fröbe, K. Boland, C. Mathews, W. Pertsch, J. Romberg, I. Zelch, S. Dietze, M. Hagen, M. Potthast, B. Stein, Overview of touché 2026 — argumentation systems, in: M. Hagen, M. Potthast, B. Stein, P. Schaer, E. Zangerle, S. MacAvaney, J. M. Struß, E. S. Salido, A. Barrón-Cedeño, A. G. S. Herrera (Eds.), Experimental IR Meets Multilinguality, Multimodality, and Interaction. 17th International Conference of the CLEF Association (CLEF 2026), Lecture Notes in Computer Science, Springer, Berlin Heidelberg New York, 2026.

[2] J. Kiesel, M. Feger, T. Hagen, S. Heineking, M. Heinrich, M. Fröbe, K. Boland, C. Mathews, W. Pertsch, J. Romberg, I. Zelch, S. Dietze, M. Hagen, M. Potthast, B. Stein, Overview of touché 2026 — argumentation systems — extended version, in: E. S. Salido, A. Barrón-Cedeño, A. G. S. Herrera, S. MacAvaney, J. M. Struß (Eds.), CLEF 2026 Working Notes, CEUR Workshop Proceedings, CEUR-WS.org, 2026.

[3] S. Y. Sahai, O. Balalau, R. Horincar, Breaking down the invisible wall of informal fallacies in online discussions, in: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, 2021, pp. 644–657. doi:10.18653/v1/2021.acl-long.53.

[4] M. Fröbe, M. Wiegmann, N. Kolyada, B. Grahm, T. Elstner, F. Loebe, M. Hagen, B. Stein, M. Potthast, Continuous integration for reproducible shared tasks with tira.io, in: J. Kamps, L. Goeuriot, F. Crestani, M. Maistro, H. Joho, B. Davis, C. Gurrin, U. Kruschwitz, A. Caputo (Eds.), Advances

in Information Retrieval. 45th European Conference on IR Research (ECIR 2023), Lecture Notes in Computer Science, Springer, Berlin Heidelberg New York, 2023, pp. 236–241. doi:10.1007/978-3-031-28241-6_20.

[5] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, Roberta: A robustly optimized bert pretraining approach, arXiv preprint arXiv:1907.11692 (2019).

[6] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of NAACL-HLT 2019, 2019.

[7] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.

[8] F. Macagno, Argumentation profiles and the manipulation of common ground: The arguments of populist leaders on twitter, Journal of Pragmatics 191 (2022) 67–82.

[9] S. Lundberg, Shap: Shapley additive explanations documentation, 2018. Accessed: 27 May 2026.

[10] G. Lim, J. Kim, S. T. Perrault, Iffy-or-not: Critically evaluating potential misinformation with fallacy detection and socratic questioning using llms, ACM Transactions on Computer-Human Interaction 33 (2025) 8:1–8:40. doi:10.1145/3771935.

[11] J. Lawrence, C. Reed, Argument mining: A survey, Computational Linguistics 45 (2019) 765–818. doi:10.1162/coli_a_00364.