

A Model-Driven Metamodel for the Systematic Design of Configurable NLP Pipelines

Kamal Walji^{1,*}, Allae Erraissi², Abdelali Zakrani¹, and Mouad Banane¹

¹ *Laboratory of Artificial Intelligence, Modeling & Computational Engineering, Hassan II University, ENSAM, Casablanca, Morocco.*

² *Polydisciplinary Faculty of Sidi Bennour, Chouaib Doukkali University, El Jadida 24000, Morocco.*

* *Corresponding author: kamal.walji-etu@etu.univh2c.ma*

Received: 1 July 2026 **Accepted:** 5 September 2026 **Published:** 17 October 2026

DOI: 10.15849/ijasca.172 **License:** Open access under CC BY 4.0. Authors retain copyright.

Publisher: Al-Zaytoonah University of Jordan (ZUJ), Amman, Jordan.

Cite this article: K. Walji, A. Erraissi, A. Zakrani, and M. Banane, "A model-driven metamodel for the systematic design of configurable NLP pipelines," *Int. J. Advance Soft Compu. Appl.*, vol. 18, no. 3, pp. 266–295, 2026, doi: <https://doi.org/10.15849/ijasca.172>.

ABSTRACT

Natural Language Processing systems are increasingly built as configurable pipelines combining heterogeneous components, including data sources, task specifications, linguistic preprocessing, representation methods, learning models, training settings, evaluation protocols, explainability requirements, and deployment constraints. Yet these pipelines are often described either through implementation code or through informal sequences of steps, which makes their design decisions difficult to compare, reproduce, or search over automatically. Existing work at the intersection of NLP and Model-Driven Engineering has mainly used language technologies to generate UML, SysML, or DSL artifacts from textual specifications, while model-driven approaches to machine learning have addressed datasets and ML-enabled systems more broadly. This article proposes a metamodel in which the NLP pipeline itself is the modeling target. The proposed metamodel defines fifteen metaclasses organized through explicit ownership semantics, distinguishing pipeline-level, model-level, shared, and execution-generated components. It also introduces cross-cutting relations that make evaluation results traceable to the configurations that produced them. The metamodel is given a formal definition and thirteen well-formedness constraints expressed in OCL. It is instantiated on two contrasting sentiment analysis pipelines: a classical supervised configuration drawn from an executed experiment and a zero-shot configuration used as a structural test. The instantiation shows that both configurations can be represented within the same structure, that reported measurements can be traced back to their producing configurations, and that plausible malformed configurations can be rejected with an explicit diagnosis. The metamodel is intended as a foundation for NLP pipeline documentation, comparison, reproducibility, and future automatic configuration and multi-objective optimization.

Keywords: Natural Language Processing; Soft Computing; Model-Driven Engineering; NLP Pipelines; Metamodeling; AutoNLP.

1 Introduction

Natural Language Processing has become central to artificial intelligence, supporting text classification, sentiment analysis, information extraction, machine translation, summarization, question answering, dialogue systems, and text generation. The evolution of the traditional approach of machine learning to deep learning and Transformer-based models has enhanced the performance on these tasks; however, it has also introduced complexity into the process of designing the pipeline for such systems. An advanced NLP system can use various sources of information, deal with multiple tasks, use various models in different roles, use various types of representations, and be evaluated through different evaluation protocols.

Descriptions of such systems have not kept pace with this complexity. In most studies, an NLP pipeline is presented either as a sequence of technical steps running from raw text to prediction, or through the code that implements it. Both descriptions are useful, but both are incomplete. Code ties the pipeline to one library, one version, and one implementation context; a sequence of steps hides the design decisions that determine whether two pipelines are comparable. Neither representation makes explicit which model addresses which task, which representation a model consumes, or which configuration produced a reported score. When these decisions remain implicit, pipelines are hard to compare, hard to reproduce, and difficult to search over automatically.

Model-Driven Engineering offers established instruments for this kind of problem. A metamodel defines the concepts, relationships, attributes, and constraints governing a family of models, thereby providing a description that is independent of any particular implementation [10]–[13]. A pipeline implemented with scikit-learn and another implemented with Hugging Face may differ technically while sharing a conceptual structure, and it is precisely this structure that a metamodel can capture.

Most work connecting NLP with modeling has followed a route different from the one adopted in this article. In many studies, NLP techniques, deep learning models, or, more recently, large language models are used to transform natural-language descriptions into structured artifacts such as UML diagrams, SysML models, DSL specifications, or executable models [3]–[8]. These studies are useful because they show how language technology can support model construction. However, the object being modeled is usually a software or systems-engineering artifact. The NLP pipeline itself, including its data, preprocessing, representations, models, training choices, and evaluation settings, is rarely treated as a modeling object in its own right.

A related body of work applies model-driven engineering to machine learning systems, including datasets, ML components, and ML-enabled architectures [29]–[31]. This line of research is closer to the present proposal because it treats machine learning artifacts as objects that can be modeled. Nevertheless, it remains broader than the problem addressed here. NLP pipelines raise specific modeling issues, including linguistic levels, processing granularity, language-dependent preprocessing, representation choices ranging from TF–IDF vectors to instruction prompts, and evaluation protocols that differ between classification, extraction, and generation tasks.

Fig. 1 situates the present contribution at the intersection of Natural Language Processing, Model-Driven Engineering, and pipeline configuration.

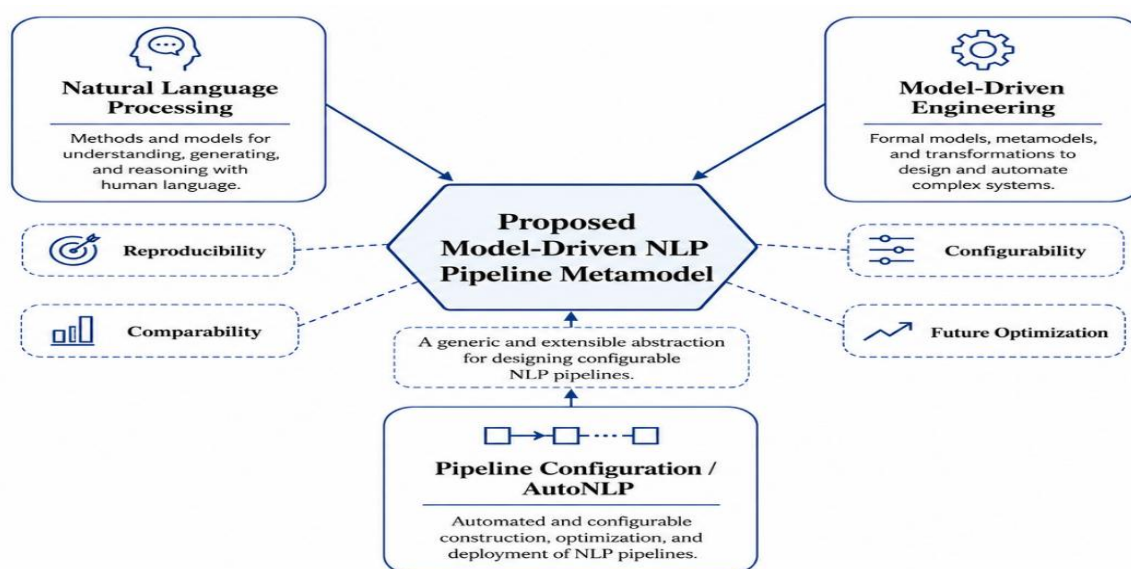


Fig. 1. Positioning of the proposed work.

This article proposes a metamodel in which the NLP pipeline itself is the modeling target. The structure rests on an explicit ownership semantics: most components are owned by the pipeline through composition, data sources are shared resources attached by aggregation because a corpus may outlive any pipeline using it, training configurations belong to individual models because training parameters are model-specific, and evaluation results belong to the evaluation protocols that generated them. Cross-cutting relations complete the structure by recording which model addresses which task, which representation a model consumes, and which model, task, and data source an evaluation assesses. In this way, a reported score is not treated as an isolated number. It remains connected to the task, data source, representation, model, preprocessing choice, and evaluation protocol from which it was obtained. This

link is important for reproducibility, because reproducing a result requires knowing not only the value that was reported, but also the configuration that produced it.

The purpose of the article is therefore deliberately limited. It does not introduce another NLP algorithm, nor does it present a new benchmark. It defines a conceptual layer above implementation code, where NLP pipelines can be described, checked, compared, and later used as a basis for automatic configuration. The contributions of this article are as follows:

1. A metamodel representing NLP pipelines as configurable model-driven artifacts, organized around fifteen metaclasses covering data sources, tasks, linguistic levels, preprocessing, representation, feature engineering, models, training, optimization, evaluation, explainability, deployment, results, and individual metric values.
2. An explicit ownership semantics distinguishing pipeline-level, model-level, shared, and execution-generated components, which clarifies the status of elements whose placement is otherwise left to interpretation.
3. Cross-cutting relations supporting traceability from reported measurements back to the model, task, data source, representation, preprocessing strategy, and evaluation protocol that produced them.
4. Variability mechanisms, including flexible multiplicities, controlled vocabularies, and meaningful absence, allowing classical, neural, Transformer-based, zero-shot, hybrid, and retrieval-augmented pipelines to be described within one structure without structural exceptions.
5. A formal definition of a pipeline configuration together with thirteen well-formedness constraints expressed in OCL, making it possible to check whether a description is structurally complete, referentially consistent, and coherent with its declared paradigm.
6. A worked instantiation on two contrasting sentiment analysis pipelines, one grounded in an executed experiment and one used as a structural test, together with verification against the constraints and four malformed configurations.

The remainder of the article is organized as follows. Section 2 reviews related work on NLP-based model generation, LLM-assisted model-driven engineering, meta-level approaches in NLP, and standards and model-driven approaches for machine learning workflows. Section 3 presents the methodology, covering the foundations of Model-Driven Engineering, the design science orientation of the work, and the requirements the metamodel must satisfy. Section 4 presents the proposed metamodel, its metaclasses, ownership structure, cross-cutting relations, and variability mechanisms. Section 5 gives the formal definition and the well-formedness constraints. Section 6 instantiates the metamodel on two sentiment analysis pipelines and verifies them against the constraints. Section 7 discusses the contribution of the proposed metamodel, its relevance to soft computing and optimization, and its limitations. Section 8 concludes the article and outlines future work.

2 Related Work

2.1 NLP-Based Generation of UML, SysML, and DSL Models

The intersection between Natural Language Processing and Model-Driven Engineering has attracted increasing attention in recent years, particularly for the automatic generation of modeling artifacts from textual specifications. A significant part of this research focuses on extracting structured models such as UML, SysML, or DSL-based artifacts from natural language requirements, technical documents, manuals, or domain descriptions. These works are relevant to the present study because they demonstrate the growing role of NLP in supporting modeling activities. However, they mainly use NLP as a means to generate models, whereas the objective of the present work is different: to propose a metamodel for representing NLP pipelines themselves.

Babaalla et al. [1] provide a comparative study and critical analysis of approaches for extracting UML class diagrams from natural language specifications using NLP, computational linguistics, machine

learning, and deep learning techniques. Their study underlines why extracting UML class diagrams from textual requirements remains difficult. Requirements written in natural language are rarely complete, and their meaning often depends on context. This makes it hard for automatic or semi-automatic tools to identify classes, attributes, and relationships with sufficient reliability. For the present article, the interest of this work lies mainly in what it reveals about text-to-UML generation. Its objective is not to describe the NLP workflow itself, nor to define the components of an NLP pipeline as a reusable modeling structure.

Zhong et al. [2] extend this line of research to systems engineering by proposing an approach for the auto-matic generation of SysML diagrams from unstructured natural language text. Their method extracts entities and relationships from textual resources such as specifications, manuals, technical reports, and maintenance reports, and converts them into SysML structure and requirement diagrams. The proposed approach is described as flexible and open-domain, and its applicability is demonstrated through six case studies. This contribution is particularly relevant because it shows how NLP techniques can support model-based systems engineering. Nevertheless, the modeling target remains the SysML diagram itself, while the internal structure of the NLP work-flow used to perform such generation is not formalized as a reusable metamodel.

Recent work has also explored the transformation from text to computational models via SysML. Hendricks and Cicirello [4] propose a pipeline that exploits NLP strategies, LLMs, knowledge graph generation, SysML diagram generation, code generation, and computational model generation to derive dynamical system models from unstructured natural language text. Their approach is notable because it uses SysML diagrams as an intermediate representation between text and executable computational models. It also emphasizes the need for controllable intermediate artifacts rather than relying only on zero-shot LLM generation. However, this work is domain-oriented toward dynamical systems and engineering modeling. It does not aim to define a generic metamodel for describing NLP pipelines across different tasks, representations, models, evaluation protocols, and deployment contexts.

Similarly, Baber et al. [8] evaluate open-source LLMs for generating DSL-conformant models from natural language descriptions. Their work focuses on grammar-based model generation and examines syntactic validity, semantic completeness, and inter-model reference consistency within MDE workflows. This direction is important because it confirms that natural language can be used as an entry point for generating models that conform to explicit metamodels or DSL grammars. However, the objective remains the generation of DSL models, not the specification of a conceptual metamodel for NLP pipeline design.

Taken together, these studies show that NLP, deep learning, and LLMs are increasingly used to generate structured modeling artifacts. However, they share a common limitation with respect to the objective of the present work: they treat NLP as an enabling technology for model generation, rather than treating the NLP pipeline itself as the object of metamodeling.

2.2 LLM-Assisted Model-Driven Engineering

Large Language Models have recently introduced new opportunities for Model-Driven Engineering, especially for tasks involving natural language specifications, structured artifact generation, and model synthesis. Unlike earlier rule-based approaches, LLM-based methods can handle more flexible linguistic inputs and can produce structured outputs such as PlantUML, DSL specifications, or engineering artifacts. However, the reliability of such outputs remains a major concern, especially when generated artifacts must satisfy schemas, constraints, semantic consistency, or traceability requirements.

ATLAS, proposed by Ma et al. [5], represents a recent example of LLM-assisted MDE. It introduces a layered constraint-guided framework for generating structured engineering artifacts under explicit schemas, domain rules, and audit requirements. The framework integrates metamodel assets, compiles heterogeneous requirements into an Integrated Constraint Model, and combines constrained generation with validation-backed repair. This work is highly relevant from a methodological perspective because it shows the importance of metamodel integration, validation mechanisms, and evidence-oriented

generation in reliable AI-assisted engineering work-flows. However, ATLAS is primarily designed for structured artifact generation in schema-rich engineering environments such as AUTOSAR. It does not address the modeling of NLP pipelines as configurable systems composed of data, tasks, preprocessing, representation, models, training, evaluation, explainability, deployment, and results.

Giannouris and Ananiadou [6] propose NOMAD, a modular multi-agent LLM framework for generating UML class diagrams from natural language requirements. NOMAD decomposes the task into specialized subtasks such as entity extraction, relationship classification, and diagram synthesis. This decomposition improves interpretability and enables targeted verification. The authors also introduce a taxonomy of errors in LLM-generated UML diagrams, covering structural, relationship, and semantic/logical deviations. This contribution is important because it shows that reliable language-to-model workflows require decomposition, verification, and systematic error analysis. Nevertheless, NOMAD remains focused on UML generation from requirements and does not provide a metamodel for representing the design space of NLP pipelines.

Other works combine NLP and MDE to improve model transformation synthesis. Lano et al. [7] investigate how model transformation specifications can be derived from requirements and examples using NLP, machine learning, inductive logic programming, search-based software engineering, and metamodel matching. Their work demonstrates that NLP can support requirements analysis and reduce the search space for model transformation synthesis. However, the objective is to improve model transformation development, not to model NLP pipelines as reusable and configurable artifacts.

These LLM-assisted and MDE-oriented studies confirm that metamodels, constraints, validation, and structured representations are increasingly important for reliable AI-supported engineering. However, their focus remains on generating or transforming software and engineering models. The present work adopts a different perspective by using metamodeling to represent the architecture and configuration of NLP pipelines themselves.

2.3 Meta-Level Approaches in NLP

The term “meta-model” has also been used in NLP-related research, but often with a meaning that differs from model-driven metamodeling. In many contexts, a meta-model refers to a higher-level predictive model, an ensemble, or a model that combines the outputs or representations of several machine learning components. This usage should be distinguished from the notion of a metamodel in Model-Driven Engineering, where a meta-model defines the abstract structure, concepts, relationships, and constraints of a family of models.

For example, Xiao et al. [3] propose a real-time sentiment analysis system for social networks using a meta-model and machine learning techniques. Their system integrates textual and visual data from social media posts and combines machine learning and deep learning models such as SVM, CNN, and LSTM to improve sentiment classification. The work is relevant because it shows that the term “meta-model” is used in applied NLP and sentiment analysis. However, the proposed meta-model functions mainly as a predictive or integrative model for sentiment classification. It is not a conceptual metamodel that formalizes the components of NLP pipelines, their relationships, or their configuration constraints.

A different kind of ambiguity appears in the work of Amirhosseini et al. [9], where “NLP” refers to Neuro Linguistic Programming rather than Natural Language Processing. The paper proposes an automated approach to the “Meta Model” used in conversational analysis for identifying deletion, distortion, and generalization patterns. Although this work uses natural language processing techniques, it belongs to a different conceptual and disciplinary context. It is therefore not directly comparable to modern computational NLP pipeline design, but it is useful as a reminder that the expression “NLP meta model” can be terminologically ambiguous.

This distinction is important for the present article. The proposed work does not introduce a meta-classifier, an ensemble model, a meta-learning method, or a conversational meta-model. Instead, it proposes a model-driven metamodel that defines the abstract components and relationships required to describe configurable NLP pipelines at a high level of abstraction.

2.4 ML/NLP Pipeline Standards, Workflow Metadata, and Model-Driven Approaches to ML

Beyond the literature on NLP-based model generation, several standards and platforms have addressed the description, exchange, tracking, and operationalization of machine learning workflows. PMML provides an XML-based standard for representing statistical and data mining models in a vendor-independent manner [18], while ONNX offers an open format for representing machine learning and deep learning models with a focus on interoperability and deployment [19]. These standards are valuable for model exchange and portability. However, they mainly describe trained models and execution structures, rather than the conceptual design of NLP pipelines, including task definition, linguistic preprocessing, representation choices, evaluation settings, explainability requirements, and pipeline-level variability.

Other works focus on metadata, experiment tracking, workflow orchestration, and reproducibility. ML-Schema and MEX provide semantic vocabularies for describing machine learning algorithms, datasets, experiments, and results [20], [21]. MLflow supports experiment tracking and model lifecycle management [22], Kubeflow Pipelines enables the construction and execution of scalable machine learning workflows [23], and DVC supports reproducible machine learning workflows by connecting data versioning, code, models, metrics, and pipeline stages [24]. These tools and vocabularies are useful in practical MLOps contexts, but they remain general-purpose and implementation-oriented. Similarly, CRISP-DM and the Team Data Science Process provide project-level guidance for data mining and data science workflows [25], [26], without defining fine-grained metaclasses for NLP-specific pipeline components.

A distinct and more directly comparable line of research applies model-driven engineering to machine learning systems themselves. Naveed et al. [29] survey this intersection under the label MDE4ML and report that the majority of primary studies propose a new domain-specific language, with a smaller number relying on general-purpose languages such as UML or extending an existing notation. Their review confirms that metamodeling and DSL design are now established instruments for describing machine learning components, and it also identifies recurrent weaknesses in how such proposals are evaluated. Among the primary studies, ML2 (ML-Quadrat) by Moin et al. [30] integrates machine learning concepts into the ThingML modeling language at both the syntactic and the code-generation level, and validates the approach on an IoT case study together with an empirical user evaluation. Closer to the data end of the pipeline, Giner-Miguel et al. [31] define a domain-specific language for describing machine learning datasets in terms of structure, provenance, and social concerns, with tool support intended to make dataset descriptions comparable across projects.

These works establish that a metamodel-based description of machine learning artifacts is both feasible and useful, and they are therefore the closest antecedents of the present proposal. They differ from it in scope rather than in method. ML2 targets smart IoT and cyber-physical services, where the machine learning component is one element within a larger system architecture, and its concern is full code generation for that architecture rather than the internal design space of a language processing workflow. The dataset DSL of [31] is deliberately confined to one component of the pipeline and is, in that respect, considerably more detailed than the DataSource metaclass proposed here; the two are complementary rather than competing, and a refinement of DataSource along those lines is a natural extension. What neither addresses, and what the MDE4ML survey does not report elsewhere, is a metamodel organized around the concerns that distinguish language processing from machine learning in general: linguistic levels, processing granularity, ordered language-dependent preprocessing, representation families ranging from sparse statistical vectors to instruction prompts, and evaluation protocols whose metric families depend on whether the task is classificatory or generative.

In relation to these standards, vocabularies, platforms, process models, and model-driven approaches to machine learning, the present proposal has a narrower role. It is not intended to replace formats for model exchange, tools for experiment tracking, workflow engines, or project-level methodologies. Its role is to add a design-level abstraction for NLP pipelines, before implementation and execution choices are fixed.

The contribution is therefore focused on the pipeline as a modeling object. The metamodel makes explicit the main elements involved in an NLP configuration, including data sources, tasks, linguistic processing, preprocessing, representations, models, training settings, optimization strategies, evaluation protocols, explainability settings, deployment constraints, and results. The novelty claim should be read in this limited sense: the paper does not attempt to formalize machine learning workflows in general, but proposes an NLP-specific structure linking pipeline design, variability, reproducibility, evaluation, and future soft computing-oriented optimization.

2.5 Comparative Synthesis and Research Gap

The reviewed literature approaches the relationship between language technology and model-driven engineering from three directions. The first uses NLP, deep learning, or large language models to generate structured modeling artifacts such as UML, SysML, or DSL models from natural language descriptions [1]–[8]; here language technology is the instrument and a software or system model is the product. The second comprises standards, vocabularies, platforms, and process models supporting model exchange, experiment description, workflow execution, lifecycle management, and project organization in machine learning [18]–[26]; these are indispensable for interoperability and reproducibility, but they are neither NLP-specific nor design oriented. The third applies model-driven engineering to machine learning systems themselves [29]–[31], and is the closest antecedent of the present work. Table 1 summarizes the comparison.

Table 1. Positioning of the proposed metamodel with respect to related work

Line of work	Modeling target	Role of NLP	NLP-specific pipeline concepts	Explicit validity constraints	Refs
Text-to-UML / SysML / DSL generation	Software or system model	Generation instrument	No	DSL grammar conformance	[1], [2], [4], [8]
LLM-assisted MDE	Engineering artifact	Constrained generator	No	Schemas, integrated constraint models	[5], [6], [7]
“Meta-model” in applied NLP	Predictive or ensemble model	Object of study	No	None	[3], [9]
Model exchange standards	Trained model, execution graph	Not applicable	Partial	Format schema	[18], [19]
Metadata vocabularies and MLOps platforms	Experiment, workflow, artifact lineage	Not applicable	Generic, not NLP-specific	Vocabulary or schema level	[20]–[24]
Data science process models	Project and process	Not applicable	No	None	[25], [26]
Model-driven engineering for ML	ML component, dataset, or ML-enabled system	Not applicable	Partial; dataset or component level	DSL grammar, some OCL	[29]–[31]
Proposed metamodel	Configurable NLP pipeline	Modeling target	Explicit metaclasses	Multiplicities, controlled vocabularies, OCL invariants	—

The gap targeted here should therefore be understood more precisely. The problem is not that machine learning lacks workflow descriptions; several standards, platforms, and process models already address that need. It is also not that model-driven engineering has ignored machine learning, since this has become an active research direction. What remains less covered is the design of configurable NLP pipelines as modeling artifacts in their own right.

This paper addresses that narrower gap. It proposes a metamodel whose concepts explicitly cover linguistic levels, processing granularity, preprocessing choices, heterogeneous representation families, task-dependent evaluation protocols, explainability targets, and deployment constraints. The aim is not

only to document these elements, but also to make their combinations checkable through well-formedness constraints.

The novelty claim is therefore limited on purpose. The paper does not present a general formalization of machine learning workflows. It proposes an NLP-specific structure that brings together pipeline design, variability, reproducibility, evaluation, and readiness for future soft computing-oriented optimization within one conceptual artifact.

3 Methodology

3.1 Model-Driven Engineering, MDA, and Metamodeling

Model-Driven Engineering considers models as first-class artifacts for system specification, analysis, design, transformation, and evolution [10]–[13]. Rather than focusing exclusively on implementation code, MDE promotes the use of abstract representations that capture the structural, behavioral, and configuration-related aspects of a system. This abstraction facilitates reasoning, reuse, consistency checking, transformation, and communication independently of low-level technological details.

A model is a purposeful abstraction of a system, while a metamodel defines the language used to construct valid models. More specifically, a metamodel specifies the concepts available in a modeling language, their attributes, relationships, multiplicities, and validity constraints. A model therefore conforms to a metamodel in the same way that a software program conforms to the syntax and semantics of a programming language [11], [12].

Model-Driven Architecture is a specific realization of model-driven principles promoted by the Object Management Group [10]. MDA commonly distinguishes between Computation-Independent Models, Platform-Independent Models, and Platform-Specific Models. This separation makes it possible to define a system independently of implementation technologies before progressively refining it toward a concrete platform.

Although the present work does not implement a complete MDA transformation chain, it adopts the same separation-of-concerns principle. The proposed metamodel describes an NLP pipeline independently of a particular programming language, machine learning library, execution framework, or deployment infrastructure. A concrete pipeline may subsequently be implemented in scikit-learn, TensorFlow, PyTorch, spaCy, Hugging Face, or another environment while preserving the same abstract structure.

This perspective differs from most related studies in which NLP is used to generate UML, SysML, DSL, or other structured artifacts from textual specifications [1], [2], [4], [6], [8]. In the present work, the modeling target is the NLP pipeline itself. The proposed metamodel therefore acts as an abstraction layer between high-level pipeline design and concrete technical implementation.

This abstraction provides a common vocabulary for describing pipeline components, defines their structural relationships, and makes configuration choices explicit. It also establishes a basis for future operations such as model validation, pipeline comparison, traceability, transformation, code generation, and optimization. Recent MDE-oriented studies similarly demonstrate the value of explicit metamodels, schemas, constraints, and validation mechanisms for organizing complex AI-assisted engineering workflows [5], [7].

3.2 Design Science Orientation

This study follows a design science orientation because its main result is an artifact: a metamodel for describing configurable NLP pipelines. The goal is not to propose a new NLP algorithm, nor to build an empirical model of performance. Rather, the work aims to define a reusable structure that makes the main decisions of an NLP pipeline explicit and easier to examine [27], [28].

The starting point is a practical difficulty often encountered in NLP studies. Pipeline descriptions are usually scattered across code, informal diagrams, implementation choices, or experiment-specific

settings. As a result, it is not always clear which data source is used, which task is addressed, which representation is consumed by which model, or which configuration produced a given evaluation result. The proposed metamodel addresses this problem by organizing these recurring elements into a common modeling structure.

The construction of the artifact followed three main steps. First, recurrent components of NLP pipelines were identified from the reviewed literature and from common NLP workflow practices. Second, these components were organized into metaclasses and relationships in order to obtain a coherent model-driven structure. Third, the structure was refined through multiplicities, controlled vocabularies, variability mechanisms, and well-formedness constraints, so that pipeline descriptions could be checked rather than only documented.

The evaluation remains illustrative at this stage. To make the artifact more concrete, the paper includes a worked instantiation based on two sentiment analysis pipelines. This instantiation shows how the proposed metaclasses can be mapped to real pipeline elements, how results can be traced back to their producing configuration, and how some malformed configurations can be rejected. In this sense, the article should be read as a first design science contribution toward a more complete tool-supported framework.

3.3 Design Requirements of the Proposed Metamodel

The research gap identified in Section 2 leads to several design requirements for the proposed metamodel. These requirements are needed because configurable NLP pipelines differ not only by task or model family, but also by language, domain, representation choice, evaluation protocol, and deployment context. The metamodel must therefore do more than decompose a pipeline into components. It must also make reproducibility, evaluation, traceability, explainability, deployment, and future optimization visible at the modeling level.

The first requirement is genericity. The metamodel should be able to represent different NLP tasks, including text classification, sentiment analysis, information extraction, machine translation, summarization, question answering, dialogue processing, and text generation. It should also remain usable across languages, learning paradigms, input formats, output structures, and application domains. This genericity is necessary because the proposed artifact is not designed for a single NLP task or for a single type of model.

A second requirement is modularity. An NLP pipeline is rarely a single block. It is usually composed of elements that can be selected, configured, replaced, or extended independently. For this reason, preprocessing, representation, feature engineering, modeling, evaluation, explainability, deployment, and optimization are represented as distinct but connected elements. Training is handled separately as a model-level configuration, since different models may require different training settings, while frozen, pretrained, API-based, or zero-shot models may not require any training configuration at all.

Extensibility is also required. NLP changes quickly, and a metamodel that only fits current architectures would soon become limited. New model families, prompting strategies, retrieval mechanisms, multimodal components, and agent-based architectures should be accommodated without redesigning the whole structure. This is why several metaclasses rely on controlled vocabularies and flexible multiplicities rather than on a closed list of fixed cases.

The metamodel must also support reproducibility and evaluation awareness. In practice, NLP results depend on many decisions: the data source, preprocessing operations, representation method, model choice, training configuration, validation protocol, evaluation metrics, and sometimes resource-related indicators. When these elements remain hidden in code or are only partly described, reproducing or comparing experiments becomes difficult. The proposed metamodel therefore represents them explicitly. `EvaluationConfig` records the evaluation protocol, `EvaluationResult` records the outcome of applying that protocol, and `MetricValue` stores the individual reported measurements. This structure links a result not only to a metric name, but also to the task, dataset, model, and evaluation setting from which it was obtained.

Another requirement concerns ownership. Not all elements of a pipeline have the same status. Some belong to the pipeline, some are attached to a specific model, some are shared resources, and some are produced only after execution. `DataSource` is treated as shared because a dataset or corpus can exist independently of a given pipeline and can be reused in other settings. `TrainingConfig` is attached to `NLPModel` because training parameters are model-specific. By contrast, `OptimizationStrategy`, `EvaluationConfig`, `ExplainabilityConfig`, and `DeploymentConfig` are placed at the pipeline level. `EvaluationResult` and `MetricValue` are treated as execution-generated elements. This separation keeps the graphical metamodel, its formal interpretation, and its later instantiations consistent with one another.

Traceability is a further requirement. A useful pipeline description should not only list components; it should also show how they are connected. The metamodel therefore includes cross-cutting relations between tasks, data sources, representation methods, models, evaluation configurations, explainability configurations, and results. These relations make it possible to record which model addresses which task, which representation is consumed by which model, which data source is used during evaluation, and which evaluation protocol produced a given result. Such information is important for reproducibility, comparison, auditing, and future automated pipeline selection.

The metamodel also needs to account for deployment and explainability. Many NLP systems are now intended for operational settings, including cloud services, APIs, mobile applications, edge devices, and real-time environments. Their design may therefore depend on latency, memory usage, scalability, privacy, interoperability, and target-platform constraints. These aspects are captured through `DeploymentConfig`. Explainability is represented through `ExplainabilityConfig`, which records the explanation method, explanation type, target audience, and expected level of interpretability. Since explanations may concern either the whole pipeline or a specific model, this configuration remains at the pipeline level while allowing an optional link to `NLPModel`.

Finally, the metamodel must be ready for future optimization. The present article does not implement an optimization algorithm; it defines the structure that such an algorithm could later use. NLP pipeline design may involve a large search space that combines preprocessing choices, representation methods, model families, training parameters, evaluation objectives, explainability requirements, and deployment constraints. The `OptimizationStrategy` metaclass provides a place to describe the search method, objective function, candidate space, and constraints.

Future extensions may instantiate this component using grid search, Bayesian optimization, evolutionary algorithms, AutoNLP, or multi-objective strategies that jointly consider predictive performance, computational cost, robustness, explainability, and deployment feasibility. A further requirement concerns verifiability. A description that cannot be checked offers no guarantee that the configuration it records is coherent, and inconsistencies in pipeline descriptions are rarely detected by inspection alone. The metamodel must therefore make it possible to determine whether a pipeline description is well formed, and to reject configurations that violate structural completeness, referential integrity between components, or the semantics of a declared paradigm. This is supported through multiplicities, controlled vocabularies restricting attribute values, and explicit well-formedness constraints expressed in OCL.

Table 2 summarizes the main design requirements and identifies the metamodel components through which they are addressed.

Table 2. Design requirements of the proposed NLP pipeline metamodel

Requirement	Description	Metamodel support
R1. Genericity	Ability to represent different NLP tasks, languages, domains, data sources, and input/output structures	NLPTask, DataSource, LinguisticLevel
R2. Modularity	Ability to decompose an NLP pipeline into independent but connected components	PreprocessingStep, RepresentationMethod, FeatureEngineering, NLPModel, EvaluationConfig, DeploymentConfig

Requirement	Description	Metamodel support
R3. Extensibility	Ability to add new tasks, models, representations, processing steps, and controlled vocabularies	NLPModel, RepresentationMethod, PreprocessingStep, controlled vocabularies
R4. Reproducibility	Ability to explicitly document pipeline choices, training settings, evaluation protocols, and results	DataSource, TrainingConfig, EvaluationConfig, EvaluationResult, MetricValue
R5. Evaluation-awareness	Ability to represent task-specific metrics, evaluation protocols, results, and metric values	EvaluationConfig, EvaluationResult, MetricValue
R6. Ownership clarity	Ability to distinguish pipeline-level, model-level, shared, and execution-generated components	NLPPipeline, DataSource, NLPModel, TrainingConfig, EvaluationConfig, EvaluationResult, MetricValue
R7. Traceability	Ability to link data sources, tasks, representations, models, evaluation settings, and results	Cross-cutting relations among DataSource, NLPTask, RepresentationMethod, NLPModel, EvaluationConfig, EvaluationResult
R8. Deployment-awareness	Ability to represent practical constraints such as latency, memory usage, scalability, privacy, and target platform	DeploymentConfig
R9. Explainability-awareness	Ability to capture interpretability methods, explanation requirements, and optional model-level explanation targets	ExplainabilityConfig, optional explains relation to NLPModel
R10. Optimization-readiness	Ability to support future automatic pipeline configuration and multi-objective optimization	OptimizationStrategy
R11. Verifiability	Ability to determine whether a pipeline description is well formed and to reject inconsistent configurations	Multiplicities, controlled vocabularies, OCL well-formedness constraints

These requirements constitute the conceptual basis of the proposed metamodel. Each requirement is reflected in one or more metaclasses, associations, multiplicities, controlled vocabularies, or variability mechanisms. Sections 4 and 5 explain how these requirements are translated into the structure of the metamodel and into its well-formedness constraints.

4 Proposed NLP Pipeline Metamodel

Natural Language Processing systems are usually described either as running code or as an informal sequence of steps. Neither description travels well. Code ties the description to one library and one version; an informal sequence hides the choices that determine whether two pipelines are comparable at all. What is missing between the two is a conceptual layer at which a pipeline can be specified, compared, and reproduced independently of the tooling that implemented it.

The metamodel proposed in this section occupies that layer. It represents an NLP pipeline not as a linear chain of operations but as a set of interconnected metaclasses, each capturing one dimension of the system: the data it consumes, the tasks it addresses, its linguistic and preprocessing decisions, the representations and models it employs, and the way it is trained, evaluated, explained, and deployed.

Two dimensions organize the proposal, and keeping them apart is what gives it consistency. The first is ownership: whether a component belongs to the pipeline, to a particular model, or to the evaluation process. The second is cross-cutting relations: the functional links connecting tasks to models, models to representations, and evaluation configurations to the artifacts they assess. Conflating the two leaves the status of several components open to interpretation; separating them removes that ambiguity and, as Section 4.6 shows, is what makes evaluation results traceable. No implementation workflow, programming language, library, or deployment environment is prescribed. Classical machine learning pipelines, deep architectures, Transformer-based systems, frozen or zero-shot models, retrieval-

augmented designs, and prospective multimodal or agent-based systems are all expressible within the same framework.

4.1 Design Principles

The metamodel is organized around a central NLPPipeline metaclass, which represents the pipeline as a design artifact and coordinates the components required to describe it. A principle running through the design is that these components do not share the same conceptual status, and Section 4.2 sets out the three cases that follow from it.

The organization follows from the requirements of Section 3. Genericity comes from DataSource, NLPTask, and LinguisticLevel, which let one structure describe pipelines operating on different domains, languages, data types, and task granularities. Modularity comes from separating concerns into distinct metaclasses rather than folding them into a single configuration object. Extensibility comes from configurable concepts such as PreprocessingStep, RepresentationMethod, NLPModel, and OptimizationStrategy, together with the controlled vocabularies of Section 4.7. Reproducibility rests on explicit training, evaluation, result, and metric descriptions, and on the traceability relations of Section 4.6. Verifiability is supported by multiplicities, controlled vocabularies, and the well-formedness constraints stated in Section 5. Deployment, explainability, and optimization appear as first-class design concerns rather than as afterthoughts.

4.2 Core Ownership Structure

Fig. 2 presents the ownership structure. It shows which components the pipeline owns outright, which it merely references, and which belong instead to a model or to the evaluation process.

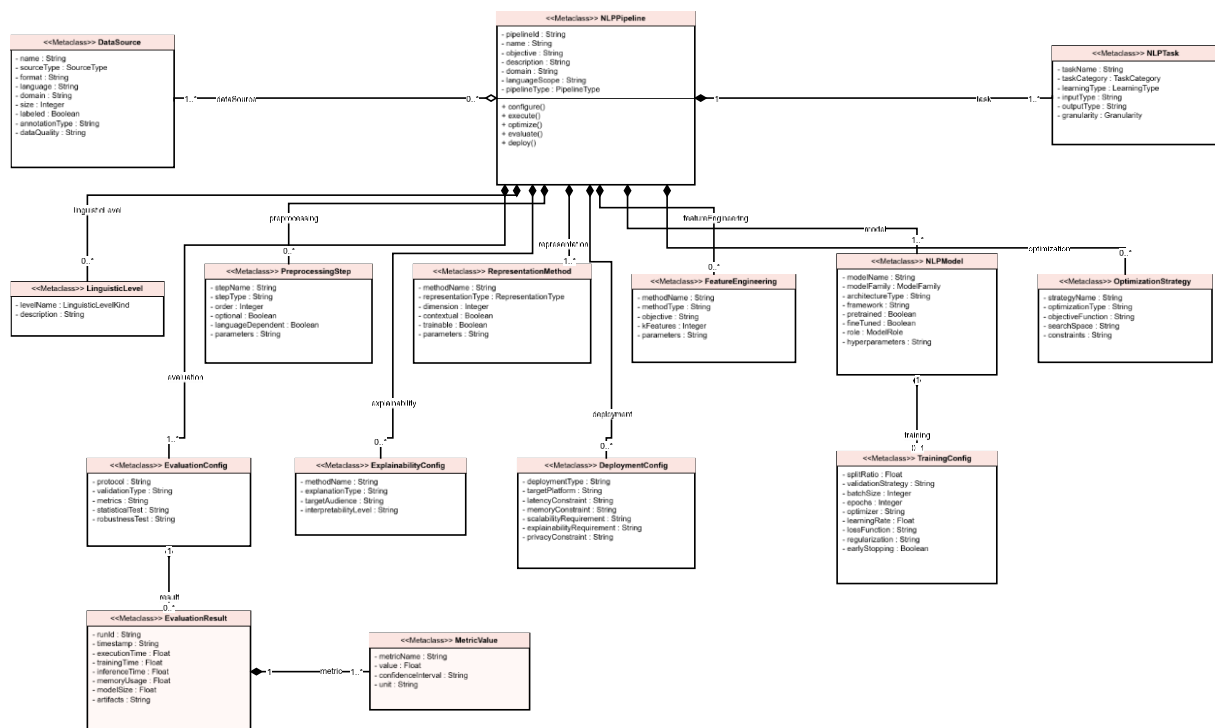


Fig. 2. Ownership structure of the proposed NLP pipeline metamodel

NLPPipeline acts as the central coordinator of the configuration. In the diagram, filled diamonds denote composition, while the hollow diamond represents shared aggregation. A valid pipeline must be associated with at least one DataSource, one NLPTask, one RepresentationMethod, one NLPModel, and one EvaluationConfig. Together, these elements define the minimum conceptual basis required to describe an NLP pipeline. A configuration that omits one of these elements does not satisfy the minimal structural conditions defined by the metamodel.

Most components are attached through composition, which expresses exclusive ownership and a dependent lifecycle: a `PreprocessingStep`, a `RepresentationMethod`, or an `EvaluationConfig` is defined for one pipeline, has no independent existence, and is deleted with it. `DataSource` is the exception. It is attached through shared aggregation, because a corpus, dataset, knowledge base, or retrieval source exists independently of any pipeline and is routinely reused across several; the pipeline references a resource it does not own. The asymmetry matters in practice: it is what makes benchmark reuse and cross-pipeline comparison expressible within the metamodel rather than left to convention, and it is why a data source carries its own quality and annotation description instead of inheriting one from whichever pipeline consumes it.

Multiplicities at the pipeline level remain deliberately flexible, since the presence of a component depends on the kind of system being modeled. Explicit preprocessing may be indispensable in a classical pipeline and entirely absorbed into the tokenizer of a pretrained Transformer. Deployment information is essential for an operational service and irrelevant to an exploratory experiment. Requiring these components would force analysts to populate metaclasses that carry no information for their pipeline, which is the kind of uniformity the proposal avoids.

`TrainingConfig` is treated differently, and this is a consequential design decision. It is attached to `NLPModel` rather than to `NLPPipeline`, because training parameters are properties of a model and not of the workflow containing it. In a pipeline holding several models — an ensemble, a retriever paired with a generator, a teacher and a student — each may require its own optimizer, learning rate, batch size, epoch count, loss function, or validation strategy, and a single pipeline-level training block cannot express that. Conversely, frozen pretrained models, externally hosted APIs, and zero-shot configurations have no training parameters to record at all. Each `NLPModel` therefore owns zero or one `TrainingConfig`, and the zero case is as meaningful as the one.

The same reasoning applies at the other end. `EvaluationResult` is owned by `EvaluationConfig` because results are produced by applying a protocol, not by declaring one; `MetricValue` is owned by `EvaluationResult` because a single run yields several measurements at once, predictive and computational alike. This three-level chain keeps design-time configuration and execution-generated artifacts structurally distinct, which is a precondition for the traceability discussed in Section 4.6.

4.3 Data and Task Specification

`DataSource` describes the textual, multimodal, or external resources a pipeline processes, recording source type, format, language, domain, size, annotation status and type, and data quality. These attributes accommodate training, validation, and test partitions as well as unlabeled corpora, knowledge bases, and retrieval sources. Because a pipeline may draw on several simultaneously — a labeled training corpus alongside an unlabeled retrieval index, for instance — the multiplicity is one or more.

`NLPTask` specifies what the pipeline is for, recording the task category, the learning paradigm, the expected input and output, and the processing granularity. It covers the standard range from classification and named-entity recognition through machine translation and summarization to question answering and text generation. Contemporary systems frequently pursue more than one objective at once, combining retrieval with generation or classification with extraction, so several `NLPTask` instances may attach to the same pipeline.

`LinguisticLevel` names the linguistic dimensions the processing engages, from lexical and morphological through to pragmatic and discourse level. Its presence is context-dependent: some pipelines model these levels explicitly, particularly in morphologically rich or low-resource languages where the choice is consequential, while others rely on representations that neural architectures learn implicitly, and declaring a level would be a fiction.

Together these three metaclasses establish the context in which every later decision is made. Preprocessing choices, representation methods, model selection, evaluation protocols, explainability needs, and deployment constraints all follow from what the data is, what the task requires, and which linguistic levels are in play.

4.4 Processing, Representation, Learning, and Optimization

`PreprocessingStep` covers the linguistic and data-cleaning operations applied before representation, training, or inference — tokenization, normalization, lemmatization, noise filtering, and comparable transformations. An `order` attribute records position in the sequence, and an optional flag distinguishes steps that may be skipped from those the pipeline depends on.

Preprocessing is modeled as context-dependent rather than mandatory, for empirical rather than theoretical reasons. In classical pipelines it is explicit and often decisive for performance. In Transformer-based systems much of it is absorbed into the pretrained tokenizer and is no longer a design choice the analyst makes. In zero-shot and API-based systems it may be handled entirely by an external service and be invisible from outside. A metamodel requiring preprocessing to be declared would force the third case to be described as though it were the first.

`RepresentationMethod` defines how text becomes something a model can consume, spanning statistical representations such as TF-IDF, distributed embeddings such as Word2Vec or FastText, contextual representations from pretrained language models, and representations produced by generative systems. Several may coexist in one pipeline, since combining sparse with dense vectors is now common practice rather than an exception. `FeatureEngineering` covers selection, dimensionality reduction, transformation, aggregation, and fusion; it is optional and repeatable, because such operations are often applied in sequence and because many neural pipelines apply none.

`NLPModel` represents the learning algorithm, pretrained model, external model, or architecture employed, and is deliberately generic: it accommodates classical learners, recurrent and convolutional networks, and Transformer-based architectures within one metaclass. A pipeline may hold several, which is what allows ensembles, hybrid architectures, retrieval-generation systems, teacher-student configurations, and multi-agent designs to be described without distortion. The `role` attribute distinguishes a model's function — primary, auxiliary, retriever, generator, reranker, classifier, or ensemble member — and it is this attribute, with the `addressedBy` relation of Section 4.6, that prevents a multi-model pipeline from being reduced to an undifferentiated list.

`TrainingConfig` records partition strategy, validation method, batch size, epoch count, optimizer, learning rate, loss function, regularization, and early stopping. As explained in Section 4.2, it is owned by `NLPModel` with a zero-or-one multiplicity rather than by the pipeline.

`OptimizationStrategy` prepares the metamodel for automatic configuration, recording the search mechanism, objective function, search space, and constraints. It can represent grid and random search, Bayesian optimization, evolutionary algorithms, neural architecture search, and multi-objective formulations. The present article does not implement an optimization process; the metaclass reserves a well-defined location for that extension and identifies which parts of a configuration constitute the search space.

4.5 Evaluation, Explainability, Deployment, and Results

`EvaluationConfig` specifies how a pipeline is assessed, and its separation from the results it produces is one of the structural commitments of the proposal. Its attributes describe validation type, selected metrics, statistical tests, robustness tests, and evaluation conditions. A pipeline may carry several configurations, which is necessary whenever it addresses multiple tasks, compares models, or is assessed under more than one protocol.

The separation matters because evaluation practice varies sharply by task: classification and generation rely on different metric families, and both may be accompanied by confidence intervals, significance testing, robustness analysis, fairness indicators, and resource measurements. Collapsing protocol and outcome into a single metaclass makes it impossible to express that the same protocol was applied twice with different results, or that two protocols were applied to the same model.

`EvaluationResult` records what an evaluation produced, and `MetricValue` decomposes that record into individual measurements, each with its name, value, confidence interval, and unit. Both are execution-

generated: they describe what happened rather than what was designed, and attaching them to EvaluationConfig rather than to NLPPipeline keeps that boundary visible in the structure itself.

ExplainabilityConfig represents interpretability requirements and the mechanisms chosen to meet them, identifying the explanation method and type, the intended audience, and the expected level of interpretability. The metaclass sits at the pipeline level, since explainability is a property demanded of a system, but it may optionally target a specific NLPModel through the explains relation, so that one metaclass expresses both a system-level requirement and a model-specific target.

DeploymentConfig captures the operational context: deployment type, target platform, latency and memory constraints, scalability requirements, privacy constraints, and interoperability needs. It carries real weight for systems intended as real-time services, APIs, mobile applications, or edge deployments, and none for a purely experimental pipeline — hence its flexible multiplicity. A pipeline deployed in more than one environment may hold several instances, each with its own constraint profile.

4.6 Cross-Cutting Relations and Traceability

Ownership tells us what a pipeline is made of. It does not tell us how the parts relate to one another, and a metamodel that stops at ownership yields a configuration tree rather than a description of a system. The relations introduced here supply what is missing, specifying dependencies, usage links, evaluation targets, and explanation scope. Fig. 3 presents them and Table 3 summarizes them.

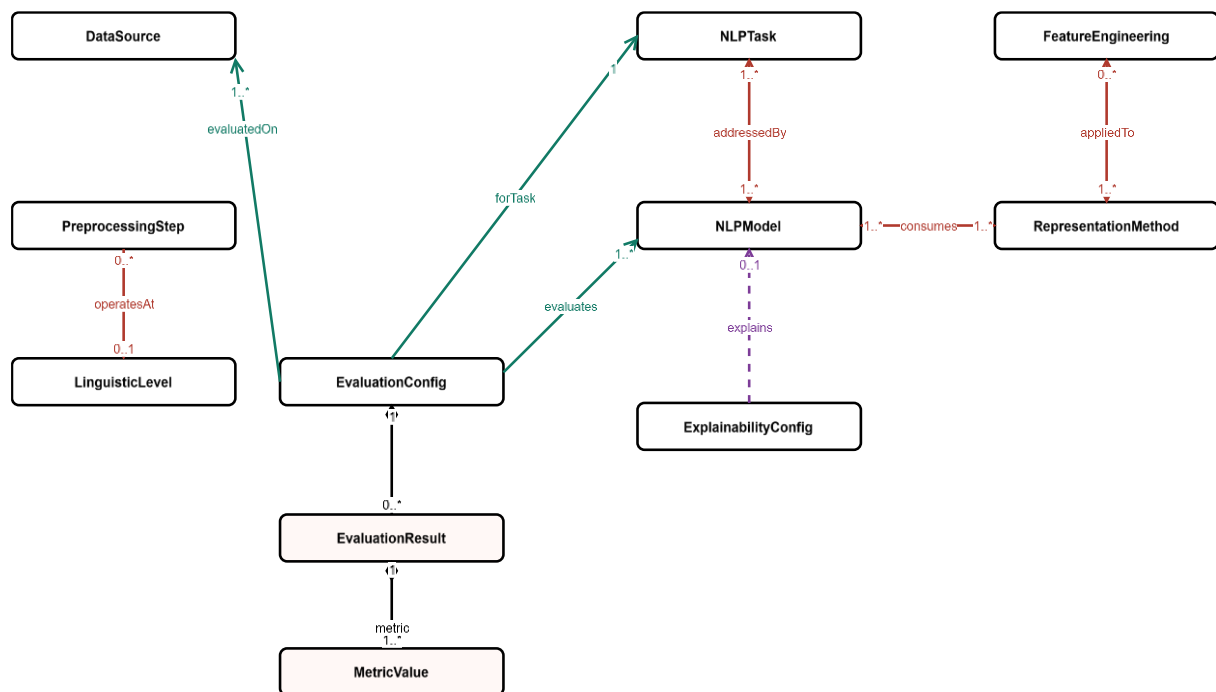


Fig. 3. Cross-cutting relations supporting traceability and consistency in the proposed NLP pipeline metamodel

operatesAt links PreprocessingStep to LinguisticLevel and states the level at which an operation applies: tokenization at the lexical level, lemmatization at the morphological level, dependency parsing at the syntactic level. Beyond documentation value, this relation is what turns LinguisticLevel from a declarative label into a functional part of the model.

consumes links NLPModel to RepresentationMethod and states which representation feeds which model. It is necessary because the same model family may take quite different inputs: a Logistic Regression classifier consuming TF-IDF vectors and a Transformer classifier consuming contextual embeddings are both classifiers, and nothing else in the structure distinguishes their input requirements. In hybrid pipelines combining sparse and dense representations, consumes is the only element preventing an ambiguous description. *appliedTo* plays the analogous role for FeatureEngineering.

addressedBy links *NLPTask* to *NLPModel* and states which model addresses which task. In single-task pipelines this is trivial; in multi-task, ensemble, retrieval-generation, and multi-agent settings it is indispensable, since several models with different functions coexist and the mapping between them and the objectives is not recoverable from the containment structure alone.

Three relations together establish evaluation traceability: *evaluates* identifies the models an *EvaluationConfig* assesses, *forTask* the task under which the assessment is performed, and *evaluatedOn* the data sources used. Combined with the ownership chain from *EvaluationConfig* through *EvaluationResult* to *MetricValue*, they allow any reported figure to be traced back to the exact model, task, data source, and protocol that produced it. This is not a refinement but a requirement: without these relations an *EvaluationResult* cannot be attributed to a specific configuration, and the reproducibility objective stated in Section 3 would remain an assertion with no structural support.

Finally, *explains* links *ExplainabilityConfig* to an optional *NLPModel* target. Its zero-or-one multiplicity is semantically loaded rather than merely permissive: with no model named, the configuration expresses a system-level requirement applying to the pipeline as a whole; with a model named, the method applies to that model in particular. Both readings are common in practice — a regulatory requirement attaches to the system, a SHAP analysis to a classifier — and one relation expresses each without introducing a second metaclass.

Table 3. Cross-cutting relations in the proposed metamodel

Relation	Source	Target	Multiplicity (source–target)	Purpose
operatesAt	PreprocessingStep	LinguisticLevel	0..* – 0..1	Linguistic level at which a preprocessing operation applies
consumes	NLPModel	RepresentationMethod	1..* – 1..*	Representation used as input by a given model
appliedTo	FeatureEngineering	RepresentationMethod	0..* – 1..*	Representation to which a feature operation is applied
addressedBy	NLPTask	NLPModel	1..* – 1..*	Model or models addressing a given task
evaluates	EvaluationConfig	NLPModel	0..* – 1..*	Model or models assessed by an evaluation protocol
forTask	EvaluationConfig	NLPTask	0..* – 1	Task under which the assessment is performed
evaluatedOn	EvaluationConfig	DataSource	0..* – 1..*	Data source or sources used for assessment
explains	ExplainabilityConfig	NLPModel	0..* – 0..1	Explanation target; unset denotes pipeline-level scope

4.7 Roles, Multiplicities, and Controlled Vocabularies

Table 5 consolidates the preceding discussion, assigning each metaclass an ownership status, a role, and a multiplicity. Six statuses are distinguished: core structural, shared resource, context-dependent, model-level, extension-oriented, and execution-generated. The classification is not a ranking. A context-dependent metaclass is not less important than a core structural one; its necessity is determined by the pipeline being described, and it is this variability that allows different pipeline families to be represented without imposing a uniformity none of them shares.

The metamodel also replaces free-text attributes with controlled enumerations wherever the domain admits a bounded set of values, and the substitution is more than cosmetic. A String-typed attribute accepts anything and therefore expresses no constraint; an enumeration restricts the instance space and

makes a configuration checkable at design time. Nine enumerations are introduced (Table 4). Attributes that remain String-typed are those whose value space is genuinely open — framework names, hyperparameter settings, free-form descriptions — where enumeration would impose a closure the domain does not have.

Table 4. Enumerations introduced in the proposed metamodel

Enumeration	Applies to	Values
PipelineType	NLPPipeline.pipelineType	CLASSICAL, NEURAL, TRANSFORMER, HYBRID, ZERO_SHOT, RETRIEVAL_AUGMENTED, MULTI_AGENT
SourceType	DataSource.sourceType	CORPUS, DATABASE, API, STREAM, KNOWLEDGE_BASE, WEB
TaskCategory	NLPTask.taskCategory	CLASSIFICATION, SENTIMENT_ANALYSIS, NER, INFORMATION_EXTRACTION, TRANSLATION, SUMMARIZATION, QUESTION_ANSWERING, DIALOGUE, GENERATION
LearningType	NLPTask.learningType	SUPERVISED, UNSUPERVISED, SEMI_SUPERVISED, SELF_SUPERVISED, ZERO_SHOT, FEW_SHOT, REINFORCEMENT
Granularity	NLPTask.granularity	CHARACTER, TOKEN, PHRASE, SENTENCE, PARAGRAPH, DOCUMENT, CORPUS
RepresentationType	RepresentationMethod.representationType	STATISTICAL, DISTRIBUTED, CONTEXTUAL, GENERATIVE, MULTIMODAL
ModelFamily	NLPModel.modelFamily	PROBABILISTIC, LINEAR, KERNEL_BASED, TREE_BASED, ENSEMBLE, NEURAL, RECURRENT, TRANSFORMER, LARGE_LANGUAGE_MODEL
ModelRole	NLPModel.role	PRIMARY, AUXILIARY, RETRIEVER, GENERATOR, RERANKER, CLASSIFIER, ENSEMBLE_MEMBER, TEACHER, STUDENT
LinguisticLevelKind	LinguisticLevel.levelName	LEXICAL, MORPHOLOGICAL, SYNTACTIC, SEMANTIC, PRAGMATIC, DISCOURSE

Table 5. Metaclasses, ownership, roles, and variability in the proposed metamodel

Metaclass	Ownership or conceptual status	Role in the metamodel	Multiplicity and design role
NLPPipeline	Central design artifact	Represents and coordinates the complete NLP pipeline	Exactly one
DataSource	Shared resource aggregated by NLPPipeline	Describes data origin, format, language, domain, annotation, size, and quality	One or more per pipeline; may be referenced by zero or more pipelines
NLPTask	Pipeline-level component by composition	Defines task category, learning type, input, output, and granularity	One or more
LinguisticLevel	Pipeline-level, context-dependent component by composition	Identifies the linguistic dimensions involved in processing	Zero or more
PreprocessingStep	Pipeline-level, context-dependent component by composition	Represents ordered text-cleaning and linguistic operations	Zero or more

Metaclass	Ownership or conceptual status	Role in the metamodel	Multiplicity and design role
RepresentationMethod	Pipeline-level component by composition	Defines the statistical, distributed, contextual, or generative representation	One or more
FeatureEngineering	Pipeline-level, context-dependent component by composition	Represents feature selection, reduction, transformation, aggregation, or fusion	Zero or more
NLPModel	Pipeline-level component by composition	Identifies the learning model, family, architecture, role, and hyperparameters	One or more
TrainingConfig	Model-level configuration by composition in NLPModel	Specifies training, adaptation, or fine-tuning settings for a specific model	Zero or one per NLPModel
OptimizationStrategy	Pipeline-level, extension-oriented component by composition	Defines pipeline-search and optimization mechanisms for future extension	Zero or more
EvaluationConfig	Pipeline-level component by composition	Specifies the evaluation protocol, metrics, statistical tests, and robustness tests	One or more
EvaluationResult	Execution-generated element owned by EvaluationConfig	Records the outcome of applying an evaluation protocol	Zero or more per EvaluationConfig
MetricValue	Execution-generated element owned by EvaluationResult	Records individual metric names, values, intervals, and units	One or more per EvaluationResult
ExplainabilityConfig	Pipeline-level, context-dependent component with optional model target	Represents explanation methods, audiences, interpretability requirements, and targets	Zero or more; may explain zero or one NLPModel
DeploymentConfig	Pipeline-level, context-dependent component by composition	Describes platform, latency, memory, scalability, privacy, and operational constraints	Zero or more

Table 5 also specifies the multiplicities that a structurally valid pipeline must respect. At the minimum, an NLPPipeline has to refer to one DataSource and include one NLPTask, one RepresentationMethod, one NLPModel, and one EvaluationConfig. Other components are not mandatory in all cases; they are included only when they are meaningful for the pipeline being described. A model may therefore have no TrainingConfig, for example in a frozen or zero-shot setting, while an EvaluationConfig may later produce several EvaluationResult instances. Each result, in turn, contains one or more MetricValue instances.

The intention is to keep two requirements together. The metamodel must be strict enough to identify an incomplete or inconsistent pipeline description, but flexible enough to cover different NLP settings without forcing artificial components into the model. This includes classical machine learning pipelines, neural and Transformer-based systems, multilingual configurations, explainable or deployable pipelines, zero-shot and hybrid designs, and future AutoNLP scenarios. Section 5 gives the formal interpretation of this structure and defines the constraints that valid pipeline instances must satisfy.

5 Formal Definition and Well-Formedness Constraints

The metamodel of Section 4 admits a compact formal reading, and giving it one serves two purposes. It states precisely what a pipeline configuration consists of, and it provides the vocabulary in which validity conditions can be expressed. A structure described only by a class diagram may be notationally correct and still admit semantically inconsistent configurations; the constraints in Section 5.2 are what distinguish a well-formed pipeline from one that merely satisfies its multiplicities.

5.1 Formal Definition of a Pipeline Configuration

Let $\mathcal{D}$ denote the universe of available data sources. Because a data source is aggregated rather than owned, $\mathcal{D}$ is defined independently of any particular configuration, which is what allows the same corpus to appear in several pipelines.

A pipeline p is defined as $p = \langle \delta, D, T, L, \Pi, R, F, M, O, E, X, \Delta \rangle$

where δ is the pipeline descriptor, comprising identifier, name, objective, domain, language scope, and pipeline type; $D \subseteq \mathcal{D}$ is the set of referenced data sources, with $|D| \geq 1$; T is the set of tasks, $|T| \geq 1$; L the set of linguistic levels, $|L| \geq 0$; Π the ordered sequence of preprocessing steps, $|\Pi| \geq 0$; R the set of representation methods, $|R| \geq 1$; F the set of feature engineering operations, $|F| \geq 0$; M the set of models, $|M| \geq 1$; O the set of optimization strategies, $|O| \geq 0$; E the set of evaluation configurations, $|E| \geq 1$; X the set of explainability configurations, $|X| \geq 0$; and Δ the set of deployment configurations, $|\Delta| \geq 0$.

The distinction between D and the remaining components is not a notational convenience. Every component other than D is owned by p and has no existence outside it, whereas D is a subset of a pre-existing universe. Π is a sequence rather than a set because preprocessing steps are ordered; the other components are unordered.

Two components are defined at a lower level than the pipeline. Each model $m \in M$ is a pair $m = \langle \mu, \theta(m) \rangle$, where μ is the model descriptor and $\theta: M \rightarrow \mathcal{T} \cup \{\perp\}$ assigns to each model either a training configuration drawn from the set $\mathcal{T}$ of admissible configurations, or the symbol $\perp$ where the model requires no local training. The value $\perp$ is meaningful rather than merely absent: it characterizes frozen pretrained models, externally hosted services, and zero-shot configurations, and distinguishes them from models whose training is simply undocumented.

Each evaluation configuration $e \in E$ is likewise a pair $e = \langle \varepsilon, \mathcal{R}(e) \rangle$, where ε is the protocol descriptor and $\mathcal{R}(e)$ the finite, possibly empty set of results produced by executing that protocol. Each result $r \in \mathcal{R}(e)$ is in turn a pair $r = \langle \rho, V(r) \rangle$, with ρ the execution descriptor and $V(r)$ the set of individual metric values, $|V(r)| \geq 1$. The chain

$E \rightarrow \mathcal{R} \rightarrow V$ separates the protocol, its execution, and the individual measurement: a protocol may be executed several times and yield distinct results, and each execution yields several measurements at once.

The cross-cutting relations of Section 4.6 are expressed as functions over the components of p :

operatesAt : $\Pi \rightarrow L \cup \{\perp\}$; consumes : $M \rightarrow \mathcal{P}(R) \setminus \{\emptyset\}$; appliedTo : $F \rightarrow \mathcal{P}(R) \setminus \{\emptyset\}$; addressedBy : $T \rightarrow \mathcal{P}(M) \setminus \{\emptyset\}$; evaluates : $E \rightarrow \mathcal{P}(M) \setminus \{\emptyset\}$; forTask : $E \rightarrow T$; evaluatedOn : $E \rightarrow \mathcal{P}(D) \setminus \{\emptyset\}$; and explains :

$X \rightarrow M \cup \{\perp\}$.

When these functions are interpreted over the components of a given pipeline, referential integrity follows from their intended codomains: a protocol cannot evaluate a model the pipeline does not contain, and a model cannot consume a representation the pipeline does not define. Constraints C4 to C7 in Section 5.2 make that restriction explicit, so that it can be checked in an implemented metamodel rather than assumed.

The nesting is what makes results attributable. Given a metric value $v \in V(r)$ with $r \in \mathcal{R}(e)$, the configuration that produced it is recovered as evaluates(e), forTask(e), evaluatedOn(e), the protocol descriptor ε , and, for each model $m \in$ evaluates(e), the pair $\langle \mu, \theta(m) \rangle$ together with the representations consumes(m). No additional bookkeeping is required. A flat structure in which training and results were attached directly to the pipeline could not support this recovery, since it would leave unstated which model a training configuration governs and which protocol produced a given result.

5.2 Well-Formedness Constraints

Multiplicities and controlled vocabularies restrict the instance space but do not exhaust what it means for a configuration to be valid. A pipeline may satisfy every multiplicity and remain incoherent: it may evaluate a model against a task no model addresses, declare itself zero-shot while attaching training configurations, or number its preprocessing steps 1, 2, 4. Conditions of this kind cannot be drawn in a class diagram and are expressed here in the Object Constraint Language, the standard companion notation for MOF-based metamodels.

Thirteen invariants are defined, grouped into four families. They are not exhaustive; they cover the errors most likely to arise when a configuration is written by hand or generated automatically. One representative invariant from each family is given in full below, and Table 6 states all thirteen.

Structural completeness. A pipeline must specify data, task, representation, model, and assessment. The remaining invariants in this family require that a recorded evaluation result carry at least one measurement, and that preprocessing positions be unique and contiguous.

context NLPPipeline inv MinimalConfiguration:

```
self.dataSource->size() >= 1 and
self.task->size() >= 1 and
self.representation->size() >= 1 and
self.model->size() >= 1 and
self.evaluation->size() >= 1
```

Referential integrity. Cross-cutting relations may only target elements the pipeline owns or references. The same pattern applies to consumed representations, evaluated tasks and data sources, and explanation targets.

context EvaluationConfig inv EvaluatedModelsAreLocal:

```
self.evaluates->forAll(m | self.pipeline.model->includes(m))
```

Semantic coherence. A protocol may only evaluate models that actually address the task it evaluates them under. Without this invariant, a reported score can be attributed to a task the model was never assigned — an error invisible to multiplicity checking and common in multi-model experimental reporting.

context EvaluationConfig inv TaskModelConsistency:

```
self.evaluates->forAll(m | self.forTask.addressedBy->includes(m))
```

Paradigm coherence. These invariants tie the declared pipelineType to the rest of the configuration, so that a stated paradigm cannot contradict the structure that implements it.

context NLPPipeline inv ZeroShotHasNoTraining:

```
self.pipelineType = PipelineType::ZERO_SHOT implies
self.model->forAll(m | m.training->isEmpty())
```

Table 6. Well-formedness constraints and the configuration errors they exclude

Id	Context	Invariant	Error prevented
C1	NLPPipeline	At least one data source, task, representation, model, and evaluation configuration	Pipeline missing an essential component
C2	EvaluationResult	At least one metric value per result	Result recorded without any measurement

Id	Context	Invariant	Error prevented
C3	NLPPipeline	Preprocessing positions are unique and contiguous	Duplicated or missing preprocessing step positions
C4	NLPModel	Consumed representations belong to the pipeline	Model consuming a representation defined outside the pipeline
C5	EvaluationConfig	Evaluated models belong to the pipeline	Protocol evaluating a model outside the pipeline
C6	EvaluationConfig	Evaluated task and data sources belong to the pipeline	Assessment performed on a task or data source outside the pipeline
C7	ExplainabilityConfig	Explanation target, when set, belongs to the pipeline	Explanation targeting a foreign model
C8	EvaluationConfig	Evaluated models address the task under assessment	Score attributed to a task the model does not address
C9	NLPModel	Fine-tuning implies pretraining	Fine-tuning declared without pretraining
C10	NLPModel	A model that is not pretrained carries a training configuration	Model with undefined parameter origin
C11	TrainingConfig	Split ratio is strictly between 0 and 1	Invalid data partition
C12	NLPPipeline	A zero-shot pipeline carries no training configuration	Declared paradigm contradicting the structure
C13	NLPPipeline	A retrieval-augmented pipeline contains at least one retriever and at least one generator	Declared paradigm without the roles that define it

Together, the definition of Section 5.1 and these invariants give the metamodel a checkable notion of validity. A configuration satisfying the multiplicities of Table 5, the enumerations of Table 4, and the invariants of Table 6 is a well-formed pipeline instance. One that violates any of them can be rejected before execution, with a diagnosis identifying the specific condition that failed. Section 6 applies both the definition and the constraints to two contrasting pipeline configurations.

6 Case Study Instantiation

6.1 Purpose and Scenario

A metamodel may be internally coherent while remaining difficult to apply to concrete pipeline descriptions. This section therefore instantiates the proposed metamodel on two sentiment analysis pipelines and checks both against the well-formedness constraints of Section 5.2. The objective is not to report a new empirical benchmark, nor to argue that one configuration outperforms the other. The purpose is to show how the metaclasses, ownership relations, cross-cutting associations, multiplicities, and constraints apply to realistic settings.

Sentiment analysis is suitable for this instantiation because the task definition can remain stable while the implementation choices vary considerably [14]. Binary polarity classification over movie reviews may be addressed by a classical supervised pipeline built on sparse statistical representations and a locally trained classifier. It may also be addressed by a zero-shot pipeline in which an instruction prompt is submitted to a pretrained language model without local training. Representing both configurations within the same structure provides an initial indication of the metamodel's flexibility.

Instantiation A describes a pipeline implemented in earlier work on the effect of preprocessing strategies on sentiment classification [32]. This grounds the instantiation in an executed configuration rather than a purely constructed one. The source study combines TF-IDF representation with Logistic Regression over the IMDb Large Movie Review Dataset and compares six preprocessing strategies under a single evaluation protocol. Instantiation B describes a zero-shot configuration on the same corpus. It is not grounded in an executed experiment; it is included as a structural test of whether the metamodel

accommodates a paradigm unlike the first. Sharing a single DataSource instance between the two pipelines is deliberate, since DataSource is the only shared aggregated element in the metamodel. Composed elements such as NLPTask, EvaluationConfig, and DeploymentConfig are represented as distinct instances for each pipeline, even where they carry identical attribute values.

In the tables that follow, $\perp$ denotes the absence of a component or relation, Π denotes the set of preprocessing steps associated with a pipeline, and $\theta(m)$ denotes the training configuration associated with a model m . Attributes marked as not reported were not documented in the source experiment. They are retained in the instantiation rather than omitted, because a description that records an attribute as undocumented is more useful than one in which the attribute simply does not appear.

6.2 Instantiation A: A Classical Supervised Pipeline

Table 7 presents the configuration at the object level. Each instance receives an identifier, a metaclass, and concrete attribute values.

Table 7. Instantiation A — object-level configuration

Instance	Metaclass	Attribute values
p_A	NLPPipeline	SA-001; IMDb-TFIDF-LR; domain = movie reviews; languageScope = english; pipelineType = CLASSICAL
d_imdb	DataSource	IMDb Large Movie Review Dataset; sourceType = CORPUS; language = english; size = 50,000 labeled reviews, with 25,000 training reviews and 25,000 test reviews; labeled = true; annotationType = binary polarity; dataQuality = balanced positive/negative classes; predefined split
t_sent_A	NLPTask	taskCategory = SENTIMENT_ANALYSIS; learningType = SUPERVISED; inputType = review text; outputType = polarity label; granularity = DOCUMENT
l_lex_A, l_morph_A	LinguisticLevel	levelName = LEXICAL; levelName = MORPHOLOGICAL; instantiated from the reported preprocessing operations
pp_minimal	PreprocessingStep	minimal cleaning; removal of URLs, character repetitions, and excessive punctuation
pp_basic	PreprocessingStep	basic cleaning; extended to non-alphanumeric characters and redundant spaces
pp_stopwords	PreprocessingStep	stopword removal; elimination of common English stopwords
pp_lemma	PreprocessingStep	lemmatization; canonical form reduction using spaCy
pp_stem	PreprocessingStep	stemming; morphological reduction using NLTK Porter Stemmer
pp_emoji	PreprocessingStep	emoji normalization; conversion of emojis into standardized textual equivalents
r_tfidf	RepresentationMethod	TF-IDF; representationType = STATISTICAL; contextual = false; dimension not reported; n-gram range not reported; min_df/max_df not reported; fitting parameters not reported
m_lr	NLPModel	Logistic Regression; modelFamily = LINEAR; pretrained = false; fineTuned = false; role = PRIMARY; library not reported; version not reported; hyperparameters not reported
tc_lr	TrainingConfig	predefined train/test split = 25,000/25,000; explicit splitRatio attribute not directly reported; validationStrategy not reported; optimizer not reported; regularization not reported
e_clf_A	EvaluationConfig	held-out test evaluation; metrics = accuracy and F1-macro; statisticalTest not reported; robustnessTest not reported

Instance	Metaclass	Attribute values
er_1–er_6	EvaluationResult	one result per preprocessing strategy; execution and resource attributes not reported
v_1–v_12	MetricValue	accuracy and F1-macro per result; values reported in Table 8
—	FeatureEngineering	not instantiated; no feature selection or reduction reported
—	ExplainabilityConfig	not reported
—	DeploymentConfig	not instantiated; the configuration is experimental and declares no operational constraints
—	OptimizationStrategy	not reported; the source study compares six preprocessing strategies but defines no search mechanism or objective function

The six preprocessing strategies are represented as six independent PreprocessingStep instances rather than as a single ordered chain, because the source experiment applied each strategy separately in order to isolate its effect. Therefore, the order attribute does not apply in this instantiation: no preprocessing sequence is declared, and each pipeline execution activates one of the six strategies. The metamodel accommodates both arrangements, and recording which one holds is itself part of describing the pipeline accurately. Linguistic levels appear as two instances, since levelName admits a single enumeration value per instance, and the morphological level is represented because lemmatization and stemming operate at that level.

A single EvaluationConfig generates six EvaluationResult instances, one per preprocessing strategy. This directly exercises the multiplicity defined in Section 4.2: the protocol is declared once and executed six times, and each execution carries its own measurements. Table 8 records these metric values.

Table 8. Instantiation A — metric values recorded under e_clf_A [32]

Result	Preprocessing strategy	Accuracy	F1-macro
er_1	stemming	0.836	0.835871
er_2	basic cleaning	0.832	0.831828
er_3	emoji normalization	0.832	0.831828
er_4	minimal cleaning	0.830	0.829867
er_5	lemmatization	0.829	0.828925
er_6	stopword removal	0.827	0.826834

The cross-cutting relations are instantiated in Table 9.

Table 9. Instantiation A — cross-cutting relation instances

Relation	Instance
operatesAt	pp_stopwords → l_lex_A; pp_lemma, pp_stem → l_morph_A; pp_minimal, pp_basic, pp_emoji → ⊥
consumes	m_lr → { r_tfidf }
appliedTo	not instantiated; no feature engineering operation declared
addressedBy	t_sent_A → { m_lr }

Relation	Instance
evaluates / forTask / evaluatedOn	$e_clf_A \rightarrow \{ m_lr \} / \{ t_sent_A \} / \{ d_imdb \}$
explains	not instantiated; no explainability configuration reported

These relations make the provenance of each measurement explicit. The accuracy of 0.836 recorded in `er_1` belongs to the metric-value set of that result, which was generated under `e_clf_A`. The protocol assessed `m_lr` for the task `t_sent_A` on `d_imdb`, and `m_lr` consumed `r_tfidf` under the training configuration `tc_lr`, with `pp_stem` as the active preprocessing strategy. Reported in isolation, the figure 0.836 is not interpretable. Recovered through this chain, it is attached to a specific corpus, task, representation, model, protocol, and preprocessing strategy. This is the traceability discussed in Section 4.6, demonstrated at the instance level rather than asserted conceptually.

Table 7 also shows what the source experiment did not document. Vector dimension, model hyperparameters, validation strategy, statistical and robustness testing, and all execution and resource measurements are marked as not reported, as are the explainability and optimization concerns. The metamodel supplies none of these values, but it does make their absence visible and locates each gap in the structure. A configuration may therefore be well formed with respect to every checkable condition while remaining incompletely documented, and distinguishing these two situations is part of what the reproducibility requirement of Section 3 asks for.

6.3 Instantiation B: A Zero-Shot Pipeline

The second configuration addresses the same classification problem on the same shared data source but follows a different paradigm. Rather than relying on explicit preprocessing, feature extraction, and local training, it submits an instruction prompt to a pretrained language model. Table 10 presents the object-level differences from Instantiation A. This configuration is conceptual and is used only as a structural stress-test of the metamodel; it is not reported as an executed experiment.

Table 10. Instantiation B — object-level differences from Instantiation A

Instance	Metaclass	Attribute values
p_B	NLPPipeline	SA-002; IMDb-LLM-ZeroShot; domain = movie reviews; languageScope = english; pipelineType = ZERO_SHOT
d_imdb	DataSource	same shared instance as in Instantiation A
t_sent_B	NLPTask	same task values as <code>t_sent_A</code> ; distinct instance owned by <code>p_B</code>
—	LinguisticLevel	absent; linguistic dimensions handled implicitly by the pretrained model
—	PreprocessingStep	none; $\Pi = \emptyset$; tokenization and formatting handled internally by the hosted model
r_prompt	RepresentationMethod	instruction prompt; representationType = GENERATIVE; contextual = true; trainable = false
—	FeatureEngineering	absent
m_llm	NLPModel	instruction-tuned language model; modelFamily = LARGE_LANGUAGE_MODEL; pretrained = true; fineTuned = false; role = PRIMARY
—	TrainingConfig	absent; $\theta(m_llm) = \perp$
e_clf_B	EvaluationConfig	held-out test evaluation; metrics = accuracy and F1-macro
—	EvaluationResult	none; the configuration has not been executed

Instance	Metaclass	Attribute values
x_zs_B	ExplainabilityConfig	methodName = output rationale prompting; explanationType = local; targetAudience = end user; interpretabilityLevel = textual; explains $\rightarrow \perp$
dc_api_B	DeploymentConfig	deploymentType = REST API; targetPlatform = hosted service; latencyConstraint = application-defined threshold; privacyConstraint = no retained review text

Note: Only d_imdb is reused as a shared DataSource instance, while the remaining non-absent elements are treated as distinct instances owned by p_B.

This second pass exercises parts of the metamodel that the classical pipeline leaves invisible. An empty preprocessing set is a valid configuration rather than an omission. The absent training configuration gives $\theta(m) = \perp$ a concrete referent and confirms that the zero-or-one multiplicity on NLPModel carries meaning rather than merely permitting absence. The absence of linguistic levels is consistent with a paradigm in which such dimensions are handled implicitly. An EvaluationConfig with no results also shows that a protocol may be declared before it is executed, which is precisely what separating protocol from outcome was intended to allow.

The contrast between the two configurations also shows how the structure would surface operational conflicts. dc_api_B declares a latency requirement, whereas Instantiation A declares no deployment configuration because the source study was experimental. Once such a requirement is given a concrete threshold and results are recorded for p_B, an inference time exceeding it becomes a checkable inconsistency between an operational requirement and a measured outcome rather than an informal observation.

6.4 Verification Against the Constraints

Table 11 records the outcome of checking both configurations against C1–C13. Constraints conditioned on the presence of preprocessing sequences, local training, or retrieval augmentation are vacuously satisfied where the condition does not hold, which is their intended behavior: optional components should not force artificial objects into a pipeline instance.

Table 11. Constraint verification for both instantiations

Id	Property	Instantiation A	Instantiation B
C1	Minimal configuration	satisfied	satisfied
C2	Non-empty results	satisfied; two metric values per result	vacuous; no result recorded
C3	Preprocessing order	vacuous; strategies are alternatives, no ordered sequence is declared	vacuous; $\Pi = \emptyset$
C4–C7	Referential integrity	satisfied	satisfied
C8	Task–model consistency	satisfied	satisfied
C9	Fine-tuning presupposes pretraining	satisfied	satisfied
C10	Training required when not pretrained	satisfied; tc_lr is present	vacuous; m_llm is pretrained
C11	Valid split ratio	partially checkable; train/test counts are reported, but the splitRatio attribute is not directly reported	vacuous; no TrainingConfig
C12	Zero-shot carries no training configuration	vacuous; pipelineType = CLASSICAL	satisfied; $\theta(m_llm) = \perp$

Id	Property	Instantiation A	Instantiation B
C13	Retrieval-augmented role consistency	vacuous; not retrieval-augmented	vacuous; not retrieval-augmented

Satisfaction alone establishes little, since a permissive constraint set is satisfied by anything. The four configurations below were constructed to fail, each isolating one invariant, and none is contrived.

Declaring the six preprocessing strategies of Instantiation A as an ordered sequence while assigning `pp_lemma` and `pp_stem` the same position would violate C3. The metamodel would then reject the configuration rather than allowing a chain in which two operations claim the same place, leaving the order of execution undefined.

Adding a second model `m_nb` to `p_A` and extending `evaluates(e_clf_A)` to cover it without extending `addressedBy(t_sent_A)` would associate a reported score with a model never assigned to the task. C8 catches this; multiplicity checking does not, and the error is familiar in multi-model experimental reporting.

Assigning `pretrained = false` to `m_llm` while leaving $\theta(m_llm) = \perp$ describes a model that is neither pretrained nor trained and whose parameters therefore have no declared origin, which C10 rejects. Attaching a `TrainingConfig` to `m_llm` while `p_B` retains `pipelineType = ZERO_SHOT` sets the declared paradigm against the model structure, violating C12. This case shows that the class diagram alone is insufficient and that constraints are required to capture semantic consistency.

Each rejection names the invariant that failed, which turns the constraint set into a validation mechanism rather than a descriptive addition to the metamodel.

6.5 What the Instantiation Establishes

The two instantiations show that the metamodel represents configurations sharing a data source and a task while remaining structurally distinct pipeline instances. A sparse statistical pipeline trained locally and a zero-shot prompted model are described with the same metaclasses, relations, multiplicities, and constraints; their differences are carried by attribute values, optional components, ownership relations, and explicit absence cases such as $\perp$ rather than by structural exceptions.

Traceability holds at the instance level. Each of the twelve metric values recorded in Instantiation A resolves to a specific evaluation result, evaluation configuration, model, task, data source, representation method, training configuration, and preprocessing strategy. The six results generated under one evaluation configuration also show that separating protocol from outcome is not a formality: a single declared protocol accounts for six executions that would otherwise appear as six unrelated experiments.

The exercise further confirms the distinction between shared aggregation and composition, since only the `DataSource` instance is reused while task, evaluation, and deployment elements remain pipeline-owned. The rejected configurations also show that the constraints are not merely documentary: they detect duplicate preprocessing positions, evaluation of a model not assigned to a task, undefined model training status, and inconsistency between a declared zero-shot paradigm and a local training configuration.

This section provides illustrative rather than empirical validation. It shows that the artifact represents concrete NLP pipelines, preserves coherent ownership semantics under instantiation, supports traceability, and applies well-formedness constraints to valid and invalid configurations alike. It does not show that the metamodel is complete, that its decomposition into metaclasses is the right one, or that practitioners would find it usable without tool support. Those questions call respectively for broader coverage studies, comparison with alternative decompositions, and user-oriented evaluation.

7 Discussion and Limitations

7.1 Contribution of the Proposed Metamodel

The proposal shifts the modeling focus from individual NLP techniques to the structure of the pipeline itself. Work at the intersection of NLP and modeling has largely used natural language as an input from which modeling artifacts are generated [1], [2], [4], [6], [8]; here the pipeline is the artifact being modeled, and it is treated not as an implementation sequence but as a configuration of interrelated components.

The practical consequence concerns documentation. NLP results depend on decisions about data, preprocessing, representation, model, training, and evaluation that are frequently distributed across source code, narrative description, and supplementary material, when they are recorded at all. Representing these decisions as metaclasses gives them a single structured location, and the instantiation of Section 6 shows what this makes visible: of the components declared for a published experiment, several — vector dimension, model hyperparameters, validation strategy, statistical testing, resource measurements — were marked as not reported. The metamodel does not recover missing information, but it locates each gap in the structure rather than leaving absence indistinguishable from omission. This is a modest capability and a consequential one, since undocumented configuration is among the more persistent obstacles to reproducing published NLP results.

Comparison benefits similarly. Multi-metric comparative studies of sentiment analysis also illustrate the value of evaluating models through multiple performance measures [15]. Two pipelines can be contrasted through their internal configuration rather than through final performance alone, which is consistent with comparative studies showing that model selection should weigh adaptability and operational constraints alongside predictive accuracy [16], [17]. The traceability relations of Section 4.6 are what make this possible at the level of individual measurements: a reported figure resolves to the model, task, data source, representation, protocol, and preprocessing strategy that produced it, rather than standing as an isolated number.

7.2 Soft Computing Relevance and Optimization Potential

The relevance of this work to soft computing lies in what it prepares rather than in what it computes. An NLP pipeline is a configurable structure combining statistical learning, approximate representation, and decision-oriented processing under competing objectives, and its design involves a search space that is large, discrete, heterogeneous, and only partially ordered: preprocessing strategies, representation families, model families, training settings, evaluation objectives, and deployment constraints interact in ways that resist analytical solution. Such spaces are well suited to metaheuristic and multi-objective methods commonly associated with soft computing.

What has been missing is not the search machinery but an explicit, machine-readable description of what is being searched over. A metaheuristic requires a defined space, an objective function, and constraints; a pipeline described only by code or by narrative supplies none of these in a form a search procedure can consume. The metamodel provides that description. OptimizationStrategy identifies the search mechanism and objective; the enumerations of Table 4 bound the discrete alternatives; the multiplicities of Table 5 determine which components may vary and which must be present; and the constraints of Section 5.2 exclude regions of the space that are structurally invalid, so that a search need not evaluate configurations that could never be executed.

The multi-objective dimension follows from the same structure. EvaluationResult can record predictive metrics alongside inference time, memory use, and model size when such measurements are available, while DeploymentConfig records latency, memory, scalability, and privacy requirements. An objective function defined over these components can weigh accuracy against operational feasibility, which matters because the appropriate model for a deployed system is rarely the most accurate one but the one best satisfying a combination of performance and constraint. Section 6 shows this in outline: the classical configuration declares no operational constraints because it was experimental, whereas the zero-shot

configuration declares a latency requirement that its own inference cost may not satisfy — a conflict the structure would surface once both are quantified.

This is a claim about foundations, not about results. The article defines the representation over which soft computing methods could operate; it does not implement a search, and Section 7.3 states this among the limitations.

7.3 Limitations and Validity Threats

The evaluation is illustrative rather than empirical. Section 6 shows that the metamodel expresses two structurally dissimilar pipelines and rejects four malformed configurations, which establishes representability and internal coherence. It does not establish that practitioners find the metamodel usable, that its metaclasses are correctly grained, or that it improves reproducibility in practice — claims that would require user studies, comparison against alternative decompositions, and longitudinal assessment respectively.

Coverage is argued rather than demonstrated. The metamodel is presented as accommodating classical, neural, Transformer-based, zero-shot, hybrid, and retrieval-augmented pipelines, but only two configurations were instantiated, and only one of them corresponds to an executed experiment. Retrieval-augmented and multi-agent designs are supported through the role attribute and constraint C13, yet neither has been instantiated. A systematic coverage study, sampling pipelines from published work across paradigms and attempting to express each, would test the claim properly.

The decomposition into fifteen metaclasses is one defensible partition among several. Alternative groupings are conceivable — merging feature engineering into representation, or separating inference from evaluation — and nothing in this article demonstrates that the chosen partition is superior rather than merely workable. The same applies to the enumerations of Table 4, whose values reflect current practice and will require revision as the field moves.

OptimizationStrategy remains uninstantiated. It reserves a location for automatic configuration and identifies which components constitute the search space, but no search is defined, executed, or evaluated here, and the soft computing argument of Section 7.2 is therefore a statement about what the structure enables rather than a demonstration that it does.

Finally, the metamodel has no tool support. Constraints are expressed in OCL but were verified by inspection rather than by a constraint checker, and no editor, validator, or transformation exists. Manual verification is feasible for configurations of the size shown in Section 6, but it would not scale to larger industrial or multi-component NLP pipelines. This confirms the need for future tool support, including a modeling editor, an automated constraint checker, and possible model-to-configuration transformations.

8 Conclusion and Future Work

This article has proposed a model-driven metamodel for the systematic design of configurable NLP pipelines. Its central design principle is the distinction between three kinds of ownership: components belonging to the pipeline, components belonging to a particular model, and components belonging to the evaluation process. This ownership structure is complemented by cross-cutting relations that make evaluation results traceable to the configurations that produce them. The proposed structure is given a formal reading, supported by a set of thirteen well-formedness constraints, and instantiated on two contrasting sentiment analysis pipelines, one drawn from an executed experiment and one serving as a structural test.

The contribution is conceptual and methodological. What the article offers is a reusable abstraction through which heterogeneous pipeline configurations can be described, compared, and checked, together with evidence that the abstraction can be applied to concrete pipelines and can reject malformed configurations. What it does not offer is empirical validation, tool support, or an implemented optimization procedure.

Four directions follow from this work. The first is implementation in a model-driven environment, using EMF/Ecore or a dedicated domain-specific modeling language, so that pipeline models can be created, checked automatically against the constraints, and transformed toward executable configurations. The second is a coverage study across paradigms, testing whether the metamodel can express pipelines it was not designed around and revising the metaclasses where it fails. The third is to fully instantiate OptimizationStrategy by defining a search space over the components already isolated by the metamodel and evaluating whether metaheuristic or multi-objective methods operating over that space can yield configurations that would be difficult to identify manually.

The fourth is extension toward paradigms whose structure the current version accommodates only in outline: prompt engineering, retrieval-augmented generation, instruction tuning, multimodal processing, and orchestration of multiple language models.

The last direction is also a caution. NLP practice moves faster than the abstractions built to describe it, and a metamodel that fits current pipelines will require revision as the field changes. The design attempts to anticipate this evolution through controlled variability and explicit extension points, but the real test of that anticipation lies in whether the structure can accommodate future paradigms that were not considered during its design.

DECLARATIONS

Funding. This research received no external funding.

Conflicts of interest. The authors declare that they have no competing interests.

Ethics approval. Not applicable. This study did not involve human participants or animal experimentation.

Informed consent. Not applicable. No human participants were recruited for this study.

Data availability. No new datasets were generated during this study. The supervised sentiment analysis example draws on previously published experimental work, while the zero-shot example is conceptual.

Author contributions. Kamal Walji: Conceptualization, methodology, formal analysis, investigation, visualization, and writing – original draft. Allae Erraissi, Abdelali Zakrani, and Mouad Banane: Supervision, methodological guidance, validation, and writing – review and editing. All authors contributed to the scientific development and writing of the manuscript, reviewed its content, and approved the final version.

Use of artificial intelligence. Artificial intelligence tools were used solely to assist with linguistic proofreading, sentence reformulation, and improving the clarity of the manuscript. All scientific content, analyses, interpretations, and conclusions were reviewed and validated by the authors, who take full responsibility for the final manuscript.

Acknowledgements. The authors have no acknowledgements to declare.

REFERENCES

- [1] Babaalla, Z., Abdelmalek, H., Jakimi, A., Oualla, M.: Extraction of UML class diagrams using deep learning: Comparative study and critical analysis. *Procedia Computer Science* 236, 452–459 (2024).
- [2] Zhong, S., Scarinci, A., Cicirello, A.: Natural Language Processing for systems engineering: Automatic generation of Systems Modelling Language diagrams. *Knowledge-Based Systems* 259, 110071 (2023).
- [3] Xiao, S., Alobaedy, M.M., Goyal, S.B., Singla, S., Kang, S., Chadha, R.: Real-time sentiment analysis on social networks using meta-model and machine learning techniques. *Scalable Computing: Practice and Experience* 24(4), 819–834 (2023).
- [4] Hendricks, M.A., Cicirello, A.: Text to model via SysML: Automated generation of dynamical system computational models from unstructured natural language text via enhanced System Modeling Language diagrams. arXiv preprint arXiv:2507.06803v3 (2026).
- [5] Ma, T., Lai, H., Wang, H., Tian, Z., Li, C., Xu, F., Fang, L.: ATLAS: A Layered Constraint-Guided Framework for Structured Artifact Generation in LLM-Assisted MDE. arXiv preprint arXiv:2510.25890v3 (2026).
- [6] Giannouris, P., Ananiadou, S.: NOMAD: A Multi-Agent LLM System for UML Class Diagram Generation from Natural Language Requirements. arXiv preprint arXiv:2511.22409v2 (2026).
- [7] Lano, K., Fang, S., Umar, M.A., Yassipour-Tehrani, S.: Enhancing model transformation synthesis using natural language processing. In: *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS '20 Companion)*, pp. 1–10. ACM (2020).
- [8] Baber, J., Hili, N., Schwab, D., Challier, L., Satrin, C.: From Text to DSL: Evaluating Grammar-Based Model Generation Using Open LLMs. In: *New Trends in Intelligent Software Methodologies, Tools and Techniques*, pp. 96–109. IOS Press (2025).
- [9] Amirhosseini, M.H., Kazemian, H.B., Ouazzane, K., Chandler, C.: Natural Language Processing approach to NLP Meta model automation. *IEEE Conference Paper*, pp. 186–193 (2018).

- [10] OMG: MDA Guide Version 1.0.1. Object Management Group (2003).
- [11] Bézivin, J.: On the unification power of models. *Software and Systems Modeling* 4, 171–188 (2005).
- [12] Schmidt, D.C.: Model-Driven Engineering. *IEEE Computer* 39(2), 25–31 (2006).
- [13] Selic, B.: The pragmatics of model-driven development. *IEEE Software* 20(5), 19–25 (2003).
- [14] Walji, K., Erraissi, A., Zakrani, A., Banane, M.: Optimization of Sentiment Analysis: Comparison Between Classical Models and Neural Architectures. In: Mejdoub, Y., Elamri, A., Kardouchi, M. (eds) *Connected Objects, Artificial Intelligence, Telecommunications and Electronics Engineering. COCIA 2025. Lecture Notes in Networks and Systems*, vol. 1584. Springer, Cham (2026). https://doi.org/10.1007/978-3-032-01536-5_97
- [15] Walji, K., Erraissi, A., Zakrani, A., Banane, M.: Sentiment Analysis Revisited: A Multi-Metric Comparative Study. *International Journal of Advanced Computer Science and Applications* 16(9) (2025). <http://dx.doi.org/10.14569/IJACSA.2025.0160965>
- [16] Idrissi Khaldi, M., Erraissi, A., Hain, M., Banane, M.: In-Depth Comparison of Supervised Classification Models: Performance and Adaptability to Practical Requirements. *International Journal of Advanced Computer Science and Applications* 16(8) (2025). <http://dx.doi.org/10.14569/IJACSA.2025.0160862>
- [17] Elbayed, Z., Qadi Elidrissi, A.: Deep Learning Algorithms for Solving the Black-Scholes Equation: A Comparative Study. In: Mejdoub, Y., Elamri, A., Kardouchi, M. (eds) *Connected Objects, Artificial Intelligence, Telecommunications and Electronics Engineering. COCIA 2025. Lecture Notes in Networks and Systems*, vol. 1584. Springer, Cham (2026). https://doi.org/10.1007/978-3-032-01536-5_91
- [18] Data Mining Group: Predictive Model Markup Language (PMML). Available at: <https://dmg.org/pmml>
- [19] ONNX: Open Neural Network Exchange Intermediate Representation Specification. Available at: <https://onnx.ai/onnx/repo-docs/IR.html>
- [20] Publio, G.C., Esteves, D., Ławrynowicz, A., Panov, P., Soldatova, L., Soru, T., Vanschoren, J., Zafar, H.: ML-Schema: Exposing the semantics of machine learning with schemas and ontologies. *arXiv preprint arXiv:1807.05351* (2018).
- [21] Esteves, D., Moussallem, D., Neto, C.B., Soru, T., Usbeck, R., Ackermann, M., Lehmann, J.: MEX Vocabulary: A lightweight interchange format for machine learning experiments. In: *Proceedings of the 11th International Conference on Semantic Systems*, pp. 169–176 (2015).
- [22] MLflow: MLflow Documentation. Available at: <https://mlflow.org/docs/latest/>
- [23] Kubeflow: Kubeflow Pipelines Documentation. Available at: <https://www.kubeflow.org/docs/components/pipelines/>
- [24] DVC: Data Version Control Documentation. Available at: <https://dvc.org/doc>
- [25] Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., Wirth, R.: CRISP-DM 1.0: Step-by-step Data Mining Guide. CRISP-DM Consortium (2000).
- [26] Microsoft: Team Data Science Process Documentation. Microsoft Learn. Available at: <https://learn.microsoft.com/>
- [27] Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. *MIS Quarterly* 28(1), 75–105 (2004).
- [28] Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A design science research methodology for information systems research. *Journal of Management Information Systems* 24(3), 45–77 (2007).
- [29] Naveed, H., Arora, C., Khalajzadeh, H., Grundy, J., Haggag, O.: Model driven engineering for machine learning components: A systematic literature review. *Information and Software Technology* 169, 107423 (2024). <https://doi.org/10.1016/j.infsof.2024.107423>
- [30] Moin, A., Challenger, M., Badii, A., Günnemann, S.: A model-driven approach to machine learning and software modeling for the IoT. *Software and Systems Modeling* 21, 987–1014 (2022). <https://doi.org/10.1007/s10270-021-00967-x>
- [31] Giner-Migueluez, J., Gómez, A., Cabot, J.: A domain-specific language for describing machine learning datasets. *Journal of Computer Languages* 76, 101209 (2023). <https://doi.org/10.1016/j.col.2023.101209>
- [32] Walji, K., Erraissi, A., Zakrani, A., Banane, M.: The Impact of Preprocessing on Sentiment Classification: An Empirical Evaluation on IMDb and Yelp. In: *Smart City Application Conference, SCA 2025, Lecture Notes in Networks and Systems*, vol. 1890, Chapter 43. Springer, Cham (2026).