Collaborative Neural-Symbolic Rule Generation for Explainable Data-to-Text Systems

Gian Marco Simonazzi[1], Alessandro Dal Palù[1,*]

[1]Dipartimento di Scienze Matematiche, Fisiche e Informatiche (Università di Parma), Parco area delle Scienze 53/A, 43124 Parma, Italia

Abstract

We present an extension of the Data2Text framework that introduces a collaborative neural-symbolic approach to explainable data-to-text generation. The original system relies on logic-based rewriting rules to transform abstract concepts into natural language, ensuring full transparency and traceability. However, its main limitation lies in the manual effort required to design and maintain linguistic rules, especially for expressive and multilingual generation.

We address this limitation by integrating Large Language Models (LLMs) as knowledge generators within a supervised pipeline, while preserving the explainable nature of the system. Rather than using LLMs as black-box text generators, we employ them to propose candidate linguistic rules, in particular capturing higher-level semantic equivalences that are not explicitly represented in traditional computational grammars. These include structural and conceptual transformations, such as alternative realizations of comparisons, temporal expressions, and syntactic variations, which go beyond simple lexical synonymy. The extracted patterns are then structured, validated, and incorporated into the symbolic rewriting engine through a multi-stage process involving specialized agents and human supervision.

The resulting framework combines the expressiveness of LLMs with the control and interpretability of symbolic methods. Experimental results show that the approach reduces manual rule engineering effort, increases linguistic variability, and supports adaptation to new domains and languages, while maintaining full explainability.

Overall, LLMs are treated as opaque sources of candidate knowledge, while the symbolic system ensures formalization, execution, and traceability, enabling a principled collaboration between neural and symbolic components.

Keywords

Neural-Symbolic Collaboration, Explainable Data-to-Text, LLM-Assisted Rule Generation

1. Introduction

In recent years, the rapid advancement of Artificial Intelligence has been largely driven by the success of Large Language Models (LLMs). Despite their unprecedented fluency and expressive power, these models operate as complex, non-linear black boxes, raising significant concerns regarding their reliability, accountability, and the interpretability of their outputs. This limitation is particularly critical in domains such as law, medicine, and critical safety systems, where the ability to trace, verify, and explain a system's reasoning process is not merely a feature, but a legal and ethical requirement, as emphasized by recent legislative frameworks like the AI Act.

Simultaneously, the field of neural-symbolic AI has emerged as a promising research direction, aiming to combine the statistical prowess of neural architectures with the rigour and formal guarantees of symbolic logic. Within the specific domain of Data-to-Text (D2T) generation, this integration is essential. Current approaches to D2T are largely polarized: neural methods produce high-quality, natural-sounding text but often suffer from hallucinations and a lack of logical grounding, while traditional symbolic approaches—notably those rooted in Logic Programming—offer full control, transparency, and consistency but are hampered by the inherent difficulty of scaling rule engineering. The "knowledge acquisition bottleneck" remains the primary obstacle: designing, extending, and maintain-

CILC 2026: The 41st Italian Conference on Computational Logic, June 23–25, 2026, Ferrara, Italy

*Corresponding author.

gianmarco.simonazzi@studenti.unipr.it (G. M. Simonazzi); alessandro.dalpalu@unipr.it (A. Dal Palù)

0000-0003-0353-158X (A. Dal Palù)

ing comprehensive rule-sets is an incredibly time-consuming task that requires deep domain expertise and linguistic proficiency.

In this work, we propose to overcome this dichotomy by extending the *Data2Concept2Text* framework, a modular, explainable pipeline for transforming structured data into natural language. The original framework relies on a Prolog-based rewriting process that ensures modularity, multilingual support, and full transparency of the generation process. However, to scale the system's expressivity and adaptability to new domains, we present a novel neural-symbolic approach that integrates LLMs into the symbolic generation loop in a controlled manner.

Our core insight is that LLMs should be treated as "knowledge discovery agents" rather than direct text generators. By using LLMs to suggest candidate linguistic patterns—which are then formalized, validated, and integrated into our Prolog-based rewriting system—we maintain the integrity of the symbolic framework. In this architecture, the LLM proposes knowledge, but the symbolic system acts as the final arbiter and executor of the rules. This ensures that the generated output remains fully inspectable and that the reasoning path remains logically sound, effectively shielding the system from the opacity typically associated with neural models.

The proposed solution relies on a multi-stage pipeline in which LLMs propose candidate linguistic patterns that are progressively structured and translated into executable Prolog predicates. We also introduce a supervised, human-in-the-loop mechanism that allows developers to validate and refine the candidate rules, ensuring that correctness, consistency, and alignment with the required linguistic quality are maintained throughout the process.

The main contributions of this work are summarized as follows:

1. A methodology for the supervised, LLM-assisted generation of symbolic linguistic rules, which effectively lowers the barrier to entry for extending rule-based D2T systems;
2. A principled neural-symbolic integration that delegates pattern extraction to neural models while delegating execution and reasoning to a transparent Prolog framework;
3. An empirical analysis demonstrating that our method improves the expressivity of the system while reducing the manual effort required for rule engineering, without compromising the explainability of the generation process. Moreover, the LLM-related computational costs remain negligible in practice.

The remainder of the paper is organized as follows. Section 2 provides an essential review of the relevant background in D2T and logic-based generation. Section 3 presents the baseline *Data2Concept2Text* architecture. Section 4 details the proposed neural-symbolic approach and its implementation. Section 5 discusses the experimental evaluation, and Section 6 concludes our findings and suggests directions for future work.

2. Related Work

Data-to-text (D2T) generation has been studied through template-based, grammar-based, neural, and logic-based approaches. Recent work on explainable D2T has shown that Answer Set Programming and Prolog can support transparent control over content selection and realization, while preserving traceability of the generation process [1, 2]. In this line, the Data2Concept2Text and Concept2Text frameworks introduced a modular rewriting architecture that separates conceptual abstraction from linguistic realization, enabling multilingual output and semantic-preserving variants through explicit rules [3, 2]. However, these approaches rely on manually defined rule sets, leading to scalability limitations.

Within symbolic natural language generation, Definite Clause Grammars provide a declarative mechanism for parsing and generation, but they primarily target syntactic well-formedness rather than higher-level semantic rewritings or alternative conceptualizations [4, 5]. This makes them complementary to rewriting-based D2T systems rather than a direct substitute. In our setting, DCGs can support grammatical realization, but they do not by themselves capture the semantic equivalence relations that our framework aims to encode.

In parallel, Large Language Models (LLMs) have significantly advanced Natural Language Generation, enabling fluent and context-aware text generation [6]. However, their black-box nature introduces limitations in terms of controllability, reliability, and explainability [7], making them unsuitable for applications requiring formal guarantees.

Recent work has explored combining LLMs with symbolic systems, including LLM-based agents that generate rule-based NLG systems or structured transformations [8, 9]. While promising, these approaches often rely heavily on neural components, limiting guarantees on correctness and consistency.

Conversely, some hybrid approaches absorb or approximate logical decisions within neural components, reducing transparency at generation time [10, 11]. This difference is crucial for our work: rather than embedding logic inside a neural generator, we keep logic as the final and inspectable layer, so that every transformation remains explicit, traceable, and formally validated.

Our work extends the Data2Concept2Text framework by integrating LLMs as knowledge generators within the rewriting pipeline. In contrast to prior approaches, we focus on the generation of semantic equivalence rules—capturing alternative conceptualizations not represented in DCGs or standard grammars—while maintaining a strict separation between neural suggestion and symbolic execution, thus preserving full explainability.

Our approach is consistent with the position advocated by Rudin [12], as the final system remains inherently interpretable. In contrast to post-hoc explanations of black-box models, we use LLMs only as auxiliary tools for discovering candidate rules, while all reasoning and execution are performed within a transparent symbolic framework.

3. The Overall Pipeline

The framework as described in [2] follows a two-stage transformation pipeline, referred to as *Data2Concept2Text*. Given an input dataset, the first stage extracts a structured conceptual representation using Answer Set Programming, while the second stage produces a natural language description from such representation through Prolog encoding. The overall process is compositional and preserves a clear separation between semantic abstraction and linguistic realization.

Concepts are modeled as finite rooted trees, where each node encodes either a class, a relation, or an attribute. Formally, a concept is represented as a term of the form

$$T = [r, T_1, \ldots, T_n]$$

where r is the root label and T_i are subtrees. This structure provides a compact and interpretable representation of semantic information, enabling hierarchical decomposition and manipulation.

The mapping from concepts to text is defined as a sequence of rewriting transformations over trees. Each transformation is specified by a rule of the form:

$$\texttt{rule}(L, \tau, \nu, T, \mathcal{T})$$

where L denotes the target language, τ the transformation stage, ν a rule identifier, T the input tree, and $\mathcal{T}$ a finite set of output trees. The rewriting process is nondeterministic, allowing the generation of multiple semantically equivalent variants. The rewriting process is stratified into ordered stages, each operating at a specific level of abstraction. Given a stage τ, rules are applied exhaustively until a fixpoint is reached before proceeding to the next stage. The main stages are:

- semantic equivalence transformations;
- structural organization into syntactic roles;
- lexical realization;
- agreement propagation;
- inflection and ordering (via CLP);
- syntactic refinement.

This stratification ensures a controlled progression from abstract concepts to surface forms.

The pipeline distinguishes between language-independent and language-dependent transformations. Early stages operate on abstract representations, while later stages introduce language-specific constraints. This separation enables the reuse of semantic transformations across languages and supports modular extensibility.

Each rewriting step is explicitly recorded, associating generated structures with the rules applied. This yields a traceable transformation sequence from input concepts to output text, ensuring transparency and interpretability of the overall process.

The framework relies on a predefined set of rewriting rules whose construction and maintenance require substantial manual effort, limiting scalability and adaptability. In particular, extending the system to new domains or languages demands significant human intervention.

4. Methodology and Implementation

The goal of this work is to reduce the manual burden of rule construction while preserving the explainability of the pipeline. LLMs are used as auxiliary components to propose candidate rewriting rules rather than final text. Traditional grammar formalisms, such as context-free grammars and Definite Clause Grammars (DCGs), are designed to model well-formed sentences, but they do not explicitly capture rewriting equivalence across semantic and structural levels.

By contrast, LLMs can infer such equivalences from examples and usage patterns, proposing candidate transformations over intermediate representations. These candidates are then structured, validated, and translated into Prolog predicates within the symbolic rewriting framework. This separation allows the system to benefit from LLM expressivity while keeping formalization, execution, and traceability entirely within the symbolic layer.

4.1. Human-in-the-loop orchestration

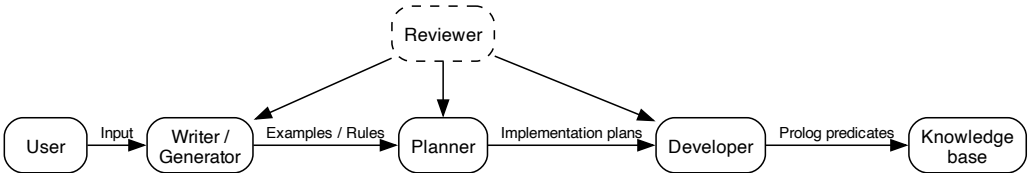

Figure 1: Pipeline orchestration

Let $\mathcal{R}$ denote the set of rewriting rules in the form `rule/5`. The proposed methodology extends $\mathcal{R}$ through a neural-symbolic pipeline composed of three agents: the `Writer`, the `Planner`, and the `Developer`, each operating at a different level of abstraction (Figure 1).

The `Writer` generates representative natural language examples for a target phenomenon (e.g., temporal complements or comparisons), providing an implicit dataset of linguistic patterns. The `Planner` abstracts these examples into structured rule specifications by identifying recurring patterns and underlying semantic conditions. The `Developer` then formalizes these specifications into executable Prolog predicates compliant with the `rule/5` schema.

Overall, the process transforms unstructured linguistic knowledge into explicit symbolic rules through four steps: generation, abstraction, formalization, and validation.

A human-in-the-loop mechanism controls rule integration into $\mathcal{R}$. The user specifies the target phenomenon and can inspect, filter, and refine intermediate outputs at each stage, ensuring correctness and preventing inconsistencies. Validated rules are incorporated without altering the operational

semantics of the rewriting system, preserving modularity and compatibility. Particular care is required to avoid cyclic dependencies among rules that may lead to unintended interactions during Prolog execution.

Within this framework, LLMs are treated as opaque components that provide candidate knowledge, while the symbolic system remains solely responsible for formalization, execution, and explainability. This separation allows the framework to leverage LLM expressivity while maintaining full transparency and control.

LLM Selection. The pipeline employs a heterogeneous set of LLMs tailored to the specific task. In particular, GPT-5 Mini (OpenAI) is used for reasoning-intensive stages, such as the Planner and the Generator, due to its stronger capability in abstraction and structured rule synthesis. For generation-oriented tasks, including the Writer and the Developer, Gemini 3 Flash and Gemini 2.5 Flash (Google) are adopted, as they provide adequate quality for text generation and code synthesis at significantly lower cost and latency.

This separation enables an efficient allocation of computational resources: GPT-5 Mini handles reasoning-intensive stages, while Gemini 3 Flash and Gemini 2.5 Flash are sufficient for producing linguistic examples and Prolog predicates at lower cost. Experimental results confirm that this combination achieves high-quality outputs while keeping the overall computational cost low, making the approach suitable for iterative and interactive rule development.

4.2. Agent Prompting

The behavior of each agent in the rule generation pipeline is defined through task-specific prompt templates, designed to constrain the model toward its intended functional role within the generation process. We illustrate the approach through cases of semantic equivalence rules and/or complement rules. For the `Writer`, the prompt includes a concise description of the relevant structural and semantic characteristics of the target transformation.

```
You are an expert in logical semantics.
Your task is to generate new semantic rules for a NLG system.

A semantic equivalence rule operates at the abstract, conceptual level to restructure the
underlying logic of a message without altering its core meaning.
Being strictly language-agnostic, it avoids specific vocabulary, syntax, or grammatical
features, focusing entirely on "what to say" rather than "how to say it".

By leveraging domain ontologies and knowledge bases, these rules perform deep logical
transformations-such as aggregating multiple narrow concepts into a
broader category, explicating implicit relationships, or converting static attributes into
dynamic, abstract event nodes.
```

We experiment with two `Writer` configurations, which are specialized through additional implementation-dependent instructions.

Mode 1 - starting from an input spanning tree

Mode 1 operates in a data-driven manner, deriving semantic abstractions directly from a given input tree and generalizing them into reusable equivalence patterns:

```
The user will provide a tree structure as a string.
Your goal is to infer abstractions starting from the given tree.
Once you find an abstraction, describe it from the other direction, so from the more
abstract tree to the more complex tree. You should also generalize the abstractions found if
 possible from the specific instance found in the current tree.
```

Mode 2 - starting from existing equivalence rules

In contrast, Mode 2 is knowledge-driven, expanding the existing rule base by identifying and proposing new equivalence rules that are not yet represented:

```
The user will provide a list of current equiv_class rules.
Your task is to generate new semantic equivalence rules that are not currently covered.
```

The following are the prompts for the `Planner` and `Developer` agents for the complement rules generation task. Note that the prompt is similar for semantic equivalence rules generation.

Planner

```
You are an expert Prolog Software Architect and a Senior Computational Linguist.
Your task is to analyze grammatical examples provided by an NLG system (in a specific target
 language) and design the architectural plan to implement a new grammatical feature (usually
 a prepositional complement).

You will NOT write the final Prolog code. Your job is to observe the examples, abstract the
linguistic rules (e.g., when to use which preposition based on the noun type), and write
clear, detailed instructions for a Prolog Developer who will write the actual code.
```

Developer

```
You are an expert Prolog developer and Computational Linguist.
Your task is to analyze a Natural Language Generation (NLG) system and generate new semantic
and grammatical rules.

### Task
The user is an expert Prolog System Architect and will give you a list of change suggestions.
 Your task is to implement the requested changes. DO NOT MAKE ANY OTHER CHANGES! You should
also add comments your code in order to make it more readable.
Describe every case you implement.
```

Both agents also receive a detailed context on the Concept2Text, covering the rule predicates interface, input tree structure and rewrite conventions.

5. Experimental Results

Evaluation Goals. The evaluation aims to assess the impact of the proposed neural-symbolic methodology along three main dimensions: (i) expressivity of the generated language, (ii) reduction of manual rule engineering effort, and (iii) preservation of explainability within the generation process.

Experimental Setting. We evaluate the proposed approach by comparing the baseline Concept2Text pipeline, based on manually defined rules, with an extended version incorporating LLM-generated rules, using the same set of input concepts. The results show that the integration of LLM-generated rules increases the diversity of syntactic and lexical realizations while preserving semantic equivalence. In particular, the number of distinct output variants grows significantly due to the combinatorial interaction between rewriting stages, confirming the impact of rule diversity on expressivity.

A key result is that explainability is fully preserved. All generated rules, once validated, are explicitly represented within the symbolic system. The rewriting trace remains accessible, and each output sentence can be justified in terms of applied rules, independently of the neural component.

We observe that candidate rules proposed by LLMs may occasionally introduce inconsistencies or overly general patterns. These issues are mitigated by the supervision step, which filters out invalid rules before integration. This confirms the necessity of maintaining a controlled interface between neural and symbolic components.

5.1. Examples

5.1.1. Generation of Complement Rules.

A set of experiments evaluates the generation of grammatical rules for complements within the *structure2grammar* stage. In this setting, complements are modeled as rewriting rules associated with `rel` nodes, and their correct realization depends on both semantic context and linguistic conventions.

The LLM-based pipeline is initialized by the `Writer`, which generates representative example sentences for the target phenomenon. For temporal complements, the first produced examples are:

> I have been studying English for five years.
> They have been living in this city since 2010.

These examples implicitly encode different temporal interpretations (duration vs. starting point), which are not explicitly formalized in the symbolic system.

The `Planner` analyzes such examples and abstracts them into structured rule specifications. For instance, it identifies that calendar dates and time points require a starting-point relation expressed by "since", while durations are expressed by "for". This leads to instructions such as:

> Add relation: start-point = "since" for calendar dates.
> Add relation: duration = "for" for temporal intervals.

These specifications are then translated by the `Developer` into executable Prolog predicates. A simplified example of the resulting rules is:

```
relation(en, pp, tempo_continuato, SubTree,
    [[[prep, "for"], modifier_np([attribute(no_art)])]]) :-
    member_non_var([class1(C) | _], SubTree),
    (common_knowledge_isa(C, duration_unit);
     member(C, [week, month, year, time, period])).

relation(en, pp, tempo_punto_iniziale, SubTree,
    [[[prep, "since"], modifier_np([attribute(no_art)])]]) :-
    member_non_var([attribute(data(_,_,_)) | _], SubTree).
```

These rules explicitly encode the conditions under which different prepositions are selected, making the underlying linguistic decision process transparent and inspectable.

5.1.2. Semantic Equivalence Rules

We evaluate the generation of semantic equivalence rules within the *equiv_class* stage, which captures alternative conceptualizations preserving meaning while enabling diverse linguistic realizations.

Generator. The `Generator` explores the space of possible abstractions and produces alternative representations of the same input semantics. Two complementary modes are considered.

Mode 1 (tree-driven). The Generator operates directly on an input concept tree:

```
[root, [class(serie), [rel(attribute), attribute(ordinale(1))],
[rel(attribute), attribute(singolare)],
[rel(verbo), class(essere),
   [rel(attribute), class(maggiore)],
   [rel(paragone), [class(serie),
      [rel(attribute), attribute(ordinale(2))],
      [rel(attribute), attribute(singolare)]
   ]]
]]]
```

corresponding to:

La prima serie è maggiore rispetto alla seconda serie
(*The first series is greater than the second series*)

From this input, the Generator proposes abstractions such as the *comparatore_binario*, enabling alternative realizations:

La prima serie supera la seconda serie
(*The first series surpasses the second series*)

This mode produces precise, context-dependent rules, but with limited generality.

Mode 2 (rule-driven). The Generator instead operates on existing rule descriptions to discover new equivalences not yet represented. This produces more abstract rules (e.g., active/passive alternation), increasing expressivity but also requiring stronger validation because of the larger search space.

Planner. The `Planner` refines candidate abstractions into structured rule specifications, explicitly identifying alternative conceptual encodings of the same semantic relation. In the comparison example, it derives three distinct output structures corresponding to different ways of representing the same information.

- **Relational abstraction.** The comparison is modeled as an explicit relation between two entities, introducing a node such as `rel(comparativo_binario)`. This form emphasizes the interaction between the two objects and supports verbal realizations (e.g., "supera" / "surpasses") or more descriptive constructions (e.g., "il confronto mostra...").
- **Attributive (subject-centric) form.** The comparison is expressed as a property of the subject, resulting in a nominal predicate (e.g., "è maggiore", i.e., "is greater"). This corresponds to the simplest and most direct encoding, where the relation is implicit in the attribute.
- **Inverted comparative form.** The comparison is reformulated by swapping subject and object and inverting the comparison (e.g., using "più"/"meno", i.e., "more"/"less"). This produces equivalent statements such as transforming "A is greater than B" into "B is less than A", preserving semantics while altering the conceptual perspective.

These alternatives represent different but equivalent conceptual structures, which are later translated into distinct rewriting rules. By explicitly enumerating them, the `Planner` enables systematic exploration of the equivalence space while keeping the transformations interpretable and controllable.

Developer. The `Developer` translates these specifications into executable Prolog rules. Due to their complexity, we report a representative excerpt:

```
% Output 1: Relation-centric
append([[rel(attribute), attribute(tipo_comparazione, Tipo)],
EntityA, EntityB, stop-comparativo_binario_elaborazione |Modifiers], C_clean, Out1Children),
Out1 = [rel(comparativo_binario) | Out1Children],

% Output 2: Subject-centric
append([[rel(paragone), EntityB]], Modifiers, PredArgs),
append([stop-comparativo_binario_elaborazione,
[rel(verbo), class(comparativo_adgettivale),
[rel(attribute), class(Tipo)] | PredArgs]], C_clean, Out2Children),

% Output 3: Reciprocal
(Tipo = maggiore -> InvTipo = minore ;
 Tipo = minore -> InvTipo = maggiore ;
 InvTipo = Tipo),
append([[rel(attribute), attribute(tipo_comparazione, InvTipo)],
EntityB, EntityA | Modifiers], C_clean, Out3Children),
RewList = [Out1, Out2, Out3].
```

This example illustrates how multiple semantic equivalences are encoded as alternative rewriting structures. Overall, the pipeline enables the discovery of non-trivial equivalences, balancing precision (Mode 1) and generality (Mode 2), while maintaining full control through symbolic validation.

Discussion. The experiments on semantic equivalence rule generation highlight both the potential and the limitations of the proposed LLM-based approach. On one hand, the pipeline is able to produce meaningful and often non-trivial abstractions, capturing higher-level semantic transformations that would be difficult to design manually. This confirms the effectiveness of LLMs as tools for exploring the space of possible equivalence rules.

On the other hand, the quality and usefulness of the generated rules depend strongly on the level of abstraction. More specific rules are precise but limited in scope, while more abstract rules increase generality at the cost of requiring stricter validation. This trade-off emphasizes the importance of human supervision in selecting and refining candidate rules.

Overall, the results suggest that the combination of LLM-based generation and symbolic validation provides a viable strategy to expand the rule base, while maintaining control, correctness, and explainability.

5.2. Multilingual support

Additional experiments evaluate the multilingual extension of the system through the adaptation of ordering constraints modeled as Constraint Satisfaction Problems (CSP) within a Constraint Logic Programming (CLP) framework. In Concept2Text, phrase linearization is not fixed but derived from declarative precedence constraints among lexical elements, solved at runtime by a CLP solver.

The LLM-based methodology operates directly on these constraints, proposing language-specific adaptations while preserving the CSP structure. Instead of assigning fixed positions, it modifies relative precedence relations, ensuring compatibility with the existing solver. A simplified example is:

```
% Variables represent positions in the phrase
Vars = [V_art, V_adj, V_noun],
Vars ins 1..3,

% Relative ordering constraints
V_art #< V_adj,
V_adj #< V_noun,

% Labeling
label(Vars).
```

Multilingual adaptation is achieved by updating these relations. For example:

```
% English-like ordering
V_art #< V_adj,
V_adj #< V_noun

% Italian-like ordering
V_art #< V_noun,
V_noun #< V_adj
```

These transformations are generated automatically by the LLM pipeline, without manual intervention. The same approach extends to verb phrases, ensuring correct ordering of auxiliaries, verbs, and complements, as in:

I have been studying English for five years

Experiments show successful adaptation across languages (e.g., English and French), with all rules remaining explicit and interpretable. The CSP formulation guarantees consistency, ensuring that multilingual extensions preserve both grammatical correctness and explainability.

Task	Agent	Model	Input	Output	Cost (€)
Complements	Writer	Gemini Flash	183	246	0.0006
	Planner	GPT-5 Mini	6204	6303	0.0141
	Developer	Gemini Flash	8309	1031	0.0072
Equivalence	Writer Mode1	GPT-5 Mini	2006	2479	0.0054
	Writer Mode2	GPT-5 Mini	3561	4361	0.0096
	Planner	GPT-5 Mini	8408	2702	0.0075
	Developer	Gemini Flash	10055	1050	0.0082

Table 1
Detailed cost breakdown for selected rule generation tasks.

5.3. Choice of LLMs and costs

The experiments show that reasoning-intensive stages, such as planning and abstraction, are better handled by GPT-5 Mini, while generation-oriented stages, such as example production and code synthesis, can be effectively managed by Gemini 3 Flash and Gemini 2.5 Flash at lower cost.

In practice, this heterogeneous model allocation provides a good trade-off between quality, latency, and cost. This modular choice allows the system to maintain high performance while keeping the overall computational cost low, making the approach suitable for iterative and interactive rule development.

From a development perspective, the proposed methodology significantly reduces the need for manual rule specification. Instead of designing rewriting rules from scratch, developers can rely on candidate rules suggested by LLMs and selectively integrate them into the system. This shifts the effort from construction to validation, resulting in improved scalability and faster iteration during rule development.

The experimental results further show that this gain is achieved with a limited computational overhead. As summarized in Table 1, the generation of rules requires only a moderate number of tokens and incurs a cost of a few cents per task. This confirms that the approach is not only effective in reducing human effort, but also practical for interactive rule development.

6. Conclusion

This work investigated the use of Large Language Models to support the extraction and formalization of linguistic knowledge in explainable data-to-text systems. The proposed methodology demonstrates that LLMs can effectively assist in the generation of semantic equivalence rules, complement structures, and multilingual adaptations, significantly improving the expressivity of the generated text while reducing manual development effort.

At the same time, the approach preserves the core properties of explainable AI by maintaining a symbolic, rule-based architecture in which all transformations remain explicit and traceable. The results highlight the potential of neural-symbolic integration as a practical strategy to combine the linguistic capabilities of LLMs with the transparency and control of logic-based systems.

Overall, the experiments confirm that LLMs can be successfully used as tools for knowledge extraction rather than direct text generation, enabling scalable and controllable extensions of rule-based Natural Language Generation systems. This positions LLMs not as generators, but as controlled sources of symbolic knowledge.

Future work will focus on extending multilingual support to a broader range of languages, improving the automatic identification of a representative set of complement rules, and further expanding the space of semantic equivalence transformations. These directions aim at increasing coverage and expressivity while preserving the explainability of the symbolic framework.

Links

Example of actual Python code that runs the LLMs calls, as well as Prolog implementation described in the paper, can be found at https://ahead-lab.unipr.it/dalpalu-cilc2026/.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT-5.3 EDU in order to: Grammar and spelling check. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] A. Dal Palù, A. Dovier, A. Formisano, An xai approach for data-to-text processing with asp, in: Proceedings of the 39th International Conference on Logic Programming (ICLP 2023), EPTCS 385, 2023, pp. 353–366. doi:10.4204/EPTCS.385.38.

[2] F. Bertini, A. Dal Palù, F. Zaglio, F. Fabiano, A. Formisano, Data2concept2text: An explainable multilingual framework for data analysis narration, in: Proceedings of the 26th Italian Conference on Theoretical Computer Science (CILC 2025), EPTCS 416, 2025, pp. 139–152. doi:10.4204/EPTCS.416.13.

[3] F. Bertini, A. Dal Palù, F. Fabiano, A. Formisano, F. Zaglio, Concept2text: An explainable multilingual rewriting of concepts into natural language, in: CILC 2024, 2024.

[4] F. C. N. Pereira, D. H. D. Warren, Definite clause grammars for language analysis, Artificial Intelligence 13 (1980) 231–278.

[5] F. C. N. Pereira, S. M. Shieber, Prolog and Natural-Language Analysis, Microtome Publishing, 2002.

[6] W. X. Zhao, K. Zhou, J. Li, et al., A survey of large language models, arXiv preprint arXiv:2303.18223 (2023).

[7] Y. Wang, W. Zhong, et al., Aligning large language models with human values, arXiv preprint arXiv:2307.12966 (2023).

[8] J. Warczyński, M. Lango, O. Dušek, Leveraging large language models for building interpretable rule-based data-to-text systems, in: Proceedings of the 17th International Natural Language Generation Conference (INLG 2024), 2024, pp. 622–630. doi:10.18653/v1/2024.inlg-main.48.

[9] M. Lango, O. Dušek, Llm agents implement an nlg system from scratch: Building interpretable rule-based rdf-to-text generators, in: Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track, 2025, pp. 2025–2040. doi:10.18653/v1/2025.emnlp-industry.142.

[10] X. Lin, H. Li, T. Huang, F. Wang, L. Chao, F. Zhuang, T. Wang, T. Zhang, A logic aware neural generation method for explainable data-to-text, Proceedings of the ACM on Management of Data (2022). doi:10.1145/3534678.3539082.

[11] R. Panchendrarajan, A. Zubiaga, Synergizing machine learning & symbolic methods: A survey on hybrid approaches to natural language processing, Expert Systems with Applications 251 (2024) 124097.

[12] C. Rudin, Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead, Nature Machine Intelligence 1 (2019) 206–215.