A User Interface to Aid in the Understanding of Rational Closure Entailments

Julia Cotterrell
cttjul001@myuct.ac.za
University of Cape Town
Cape Town, South Africa

ABSTRACT

Defeasible reasoning extends classical logic by allowing conclusions to be withdrawn when exceptions arise, offering a more realistic form of non-monotonic inference. This paper focuses on Rational Closure (RC), a central KLM-framework entailment method, and presents a web-based tool that makes RC reasoning transparent and accessible. Built in Java and TypeScript on the TweetyProject framework, the system partitions knowledge bases into base ranks, evaluates entailments, and explains results in two coordinated modes: a symbolic view tracing formal derivations and a natural language narration clarifying why formulas are removed, why queries hold or fail, and how exceptions are resolved. The novelty lies in this explanatory layer, which translates each algorithmic step into accessible language. The tool serves as both an educational resource and a practical debugger. As AI systems expand into education and governance, interpretable reasoning tools are critical for trust and adoption.

Unit and integration tests confirmed alignment between formal and natural language outputs; benchmarks showed linear scalability locally, with hosting constraints as the main bottleneck. These findings demonstrate that RC can be made interpretable without compromising correctness, lowering barriers to its use in teaching and applied contexts. To our knowledge, this is the first deployed RC system designed primarily to provide natural language explanations of Rational Closure.

KEYWORDS

artificial intelligence, knowledge representation and reasoning, non-monotonic reasoning, defeasible reasoning, rational closure, lexicographic closure, knowledge base generator, user interface, debugger

1 INTRODUCTION

In classical logic, once a conclusion is derived it cannot be retracted. Yet in everyday reasoning, exceptions frequently require us to revise our beliefs. This tension between monotonic formalisms and non-monotonic human reasoning has long motivated research into defeasible reasoning.

Knowledge Representation and Reasoning (KRR) is a branch of Artificial Intelligence (AI) that uses formal logic to encode domain knowledge, allowing automated systems to draw inferences and support decision-making [21]. While classical logic provides sound and transparent inference, its monotonic nature means that adding new information to a knowledge base can never invalidate existing conclusions. In practice this is unrealistic: for example, the rule "cars typically emit CO_2" conflicts with the fact that "Teslas are cars that do not emit CO_2." In classical logic, such exceptions can cause inconsistencies that trivialise inference [18].

Defeasible reasoning addresses this limitation by adopting a non-monotonic approach that allows conclusions to be withdrawn when new evidence arises [25]. The KLM framework provides formal semantics for defeasible reasoning allowing for typicality and exceptions. This approach underlies many modern implementations [18]. This paper focuses on Rational Closure (RC), one of its central entailment methods [9], which will be described in detail in Section 2.2.

Although RC satisfies important postulates and provides a rigorous method for handling exceptions, it remains underutilised in practice. Existing tools typically require expert knowledge, present results in symbolic formats which are difficult to interpret and offer limited support for debugging or explaining inference steps [11]. This restricts their usability in both educational contexts and applied problem solving.

This paper addresses these limitations by designing and implementing an interactive, web-based explainer for RC. The system exposes reasoning processes step-by-step by providing visualisations of base ranks and entailments alongside explanatory text. Its novelty lies in the introduction of a natural language explanation layer that translates each algorithmic step and can be read alongside the formal derivations, creating a dual-modal interface that preserves formal rigour while making defeasible reasoning more accessible. The aims of this work are to examine *whether Rational Closure (RC) can be made more interpretable through natural language explanations aligned with formal derivations; whether a dual-modal interface that combines mathematical and natural language outputs can enhance accessibility for education and debugging; and whether such an explanatory system can maintain correctness and performance while introducing this additional layer.*

The contributions of the paper follow directly from these aims. We present an interactive interface for RC that integrates symbolic and natural language explanations, a visualisation framework that renders base ranks and entailments in step-by-step form, and an evaluation of correctness, performance, and explanation fidelity. While this work forms part of a broader Honours project, this paper focuses on the explanatory interface. The system also incorporates a suite of enhanced entailment strategies (Naïve, Binary, Ternary, Hybrid, and Cached) developed within EXTRC [16], and extends the debugger by merging with a natural language knowledge base generator [12]. This integration enables explanations that connect formal symbols directly to domain-level phrases. By building on these complementary components, the system demonstrates how multiple EXTRC strands can be consolidated in a single front end. By prioritising clarity, accessibility, and interactivity, the paper advances the field by bridging the gap between formal logic and usable reasoning systems.

The remainder of this paper is structured as follows: Section 2 introduces the necessary background in propositional logic, defeasible reasoning, and Rational Closure. Section 3 reviews related work on defeasible reasoning tools. Section 4 presents the design and implementation of the system. Sections 5 and 6 present results and testing. Section 7 discusses limitations and future work, before Section 8 concludes the thesis.

2 BACKGROUND

2.1 Propositional Logic

Propositional logic is a formal system that underpins KRR by assigning truth values (T or F) to statements and combining them with *Boolean Operators* to form complex formulas [3]. The system developed in this paper accepts propositional Knowledge Bases (KB) as input, making familiarity with these basics essential.

2.1.1 Syntax. *Atomic propositions*, $\mathcal{P} = \{p, q, r, ...\}$, are built from *atoms* (lowercase symbols) and combined with *Boolean Operators* such as $\wedge$ (and), $\vee$ (or), $\rightarrow$ (if-then), and $\leftrightarrow$ (if-and-only-if) [3].

2.1.2 Semantics. Semantics are defined by interpretations that assign T or F to each atom, allowing evaluation of formulas [3]. Valuations and operators are object-level, while satisfiability and entailment operate at the meta-level [20].

Although propositional logic provides a tractable foundation, its monotonicity makes it insufficient for realistic reasonings [21]. This limitation directly motivates the turn to defeasible reasoning frameworks considered in the next section [3].

2.2 Defeasible Reasoning

Classical propositional logic is *monotonic* which makes it unsuitable for reasoning with exceptions. To demonstrate this, consider the following KB:

$$\mathcal{K} = \{\ \text{car} \rightarrow \text{emits}CO_2,\ \text{Tesla} \rightarrow \text{car},\ \text{Tesla} \rightarrow \neg\text{emits}CO_2\ \} \quad (1)$$

From the rules Tesla $\rightarrow$ car and car $\rightarrow$ emitsCO_2, classical propositional logic derives Tesla $\rightarrow$ emitsCO_2. However, the KB also states Tesla $\rightarrow$ $\neg$emitsCO_2, which is inconsistent with the conclusion that Teslas emit CO_2. In classical logic, this does not yet yield a contradiction, but it does imply that if "Tesla" is assumed to exist, then both emitsCO_2 and $\neg$emitsCO_2 would follow. This illustrates that simply adding the fact Tesla therefore *trivialises* the KB, since under inconsistency, every formula becomes derivable. As a result, the system collapses and can no longer distinguish valid inferences.

This outcome is unsound, since "Tesla" is assumed to exist. This represents an exception to the general rule that cars emit CO_2. The example illustrates that real-world reasoning often involves such exceptions and requires conclusions to be retractable when conflicting evidence arises.

Defeasible reasoning, a form of non-monotonic logic, addresses this limitation by allowing conclusions to be withdrawn in the presence of new or contradictory information. Numerous formalisations of defeasible reasoning have been proposed; this paper focuses on RC [10], that extends the KLM framework [18]. While its theoretical value is well established, practical tools remain scarce, and existing implementations often lack usability and transparency. This paper aims to address this issue.

2.2.1 The KLM Framework. Kraus, Lehmann, and Magidor (KLM) introduced a formal framework for reasoning with *typicality*, expressed through statements such as "x is typically y" [18]. This is captured as a meta-level consequence relation, denoted $\mid\sim$, and is referred to as "twiddle". These defeasible statements provide the input to a reasoning procedure such as RC, which then determines what follows defeasibly. This form of defeasible entailment is written as $\mid\approx$, so that, given a KB K, we write $K \mid\approx \varphi$ if φ follows defeasibly. For instance, adding Tesla $\rightarrow$ car and Tesla $\rightarrow$ $\neg$emitsCO_2 to the KB, we are able ask whether $K \mid\approx$ (Tesla $\mid\sim$ car) holds.

To ensure such reasoning behaves consistently, KLM proposed a set of six postulates that any *preferential consequence relation* should satisfy. Intuitively, these postulates constrain how defaults can be combined. For example, that if x typically implies y, and y typically implies z, then x should typically imply z. These consequence relations are modelled using *preferential interpretations*, which assign priority orders to different possible worlds (complete and consistent assignments of truth values to all propositions)[30].

Later, Lehmann [19] introduced a seventh postulate to better handle conflicts between defaults. This refinement gave rise to a stronger class of relations known as *rational consequence relations*. Unlike preferential relations, rational ones can resolve conflicts systematically by ranking defaults according to their level of exceptionality. This is formalised using *ranked interpretations*, which assign a rank (or degree of typicality) to valuations, thus defining the semantics of rational consequence [20].

Together, these foundations form the basis of the KLM framework, which underpins many approaches to defeasible reasoning. In this paper, the focus is on RC, a syntactic construction of rational consequence relations that provides a tractable way to reason with defaults. However, the KLM framework's abstract semantics make it difficult for users to understand why particular conclusions hold or fail. This gap between theory and application motivates more interpretable reasoning interfaces.

2.3 Base Rank

Base ranks provide the syntactic foundation for RC. They classify formulas according to their level of exceptionality, and the partitions they produce form the basis for defeasible entailment checks. The RC construction, introduced in Section 2.4, relies directly on these partitions. A formula $\alpha \in \mathcal{L}$ is *exceptional* with respect to a KB $\mathcal{K}$ if $\mathcal{K} \mid\approx_R \top \mid\sim \neg\alpha$, which means that the negation of α is typical in every preferential interpretation that satisfies $\mathcal{K}$ [8]. Equivalently, α is exceptional if its negation follows from the *materialisation* of $\mathcal{K}$, meaning all defeasible implications are translated into classical formulas.

Formally, RC defines a sequence of knowledge bases $\mathcal{E}_0^{\mathcal{K}}, \ldots, \mathcal{E}_\infty^{\mathcal{K}}$ that progressively filter out exceptional formulas. The base rank of a formula α, written $br_{\mathcal{K}}(\alpha)$, is the smallest r such that α is not exceptional in $\mathcal{E}_r^{\mathcal{K}}$. For defeasible implications, $br_{\mathcal{K}}(\alpha \mid\sim \beta) := br_{\mathcal{K}}(\alpha)$ [20].

To illustrate, refer back to the KB:

$$\mathcal{K} = \{\ \text{car} \mid\sim \text{emits}CO_2,\ \text{Tesla} \rightarrow \text{car},\ \text{Tesla} \mid\sim \neg\text{emits}CO_2\ \}.$$

The base rank procedure partitions this KB into layers that reflect how exceptional the antecedents of defeasible rules are. Classical implications (here, Tesla $\rightarrow$ car) are treated as strict knowledge and do not participate in the ranking process. They are assigned to R_∞ by convention.

Rank 0 (R_0): At the lowest level we place the most general default: car $\mid\sim$ emitsCO_2 ("cars normally emit CO_2"). This rule survives because it is not exceptional relative to the KB.

Rank 1 (R_1): The Tesla-specific default Tesla $\mid\sim$ $\neg$emitsCO_2 ("Teslas normally do not emit CO_2") is recognised as an exception to the car-emission rule. It is therefore pushed from ranking level 0 to 1.

Rank ∞ (R_∞): The strict rule Tesla $\rightarrow$ car ("if something is a Tesla, then it is a car") is classical knowledge, not a defeasible default. By convention, all such strict implications are assigned to R_∞. Any defeasible rules that collapse into outright contradiction would also be relegated here, though none occur in this example.

The resulting partition is summarised in Figure 1. Each rank thus reflects a different level of exceptionality: Tesla-related rules are exceptional, while the general car rule remains robust.

R_∞	Tesla $\rightarrow$ car
R_1	Tesla $\mid\sim$ $\neg$emitsCO_2
R_0	car $\mid\sim$ emitsCO_2

Figure 1: The partitions of $\vec{\mathcal{K}}$ produced by `BaseRank`

Base rank partitions offer a tractable way to compute defeasible entailments, yet their intermediate steps are rarely exposed in existing tools. This leaves users with outputs without understanding of the reasoning process. Our system addresses this gap by making base ranks visible and narrated.

2.4 Rational Closure

RC provides a principled method for deriving defeasible entailments from a KB. It can be characterised both semantically, in terms of ranked models, and syntactically, in terms of base ranks.

Semantically, RC is defined through a unique minimal ranked model $R^{RC}_{\mathcal{K}}$. In this model, lower ranks correspond to more typical valuations, while inconsistent or impossible valuations are assigned rank ∞. This ensures that conclusions are drawn from the most typical worlds compatible with $\mathcal{K}$ [20].

Syntactically, RC is defined using base ranks. Formally, $\mathcal{K} \mid\approx_{RC} \alpha \mid\sim \beta$ if-and-only-if $br_{\mathcal{K}}(\alpha) < br_{\mathcal{K}}(\alpha \wedge \neg\beta)$ or $br_{\mathcal{K}}(\alpha) = \infty$ [15]. The base rank of a formula therefore provides a direct method for determining defeasible entailments. This was introduced in Section 2.3. Giordano et al. further showed that RC and minimal ranked entailment produce the same set of inferences [15].

Operationally, RC proceeds in two stages: first, base rank partitions are computed; second, these partitions are used to evaluate whether a defeasible entailment holds. [8]. Given a query $\alpha \mid\sim \beta$, the algorithm:

(1) Computes base rank partitions $R_0, \ldots, R_n, R_\infty$ using `BaseRank`.
(2) Checks if $\alpha \mid\sim \beta$ is exceptional in its current partition.
(3) Iteratively removes lower-ranked components until $\alpha \mid\sim \beta$ is no longer exceptional or only R_∞ remains.
(4) Checks if $R_i \cup \cdots \cup R_\infty \models \alpha \rightarrow \beta$.

If the final entailment holds, the query is in the RC of $\mathcal{K}$, otherwise, it is not [14]. This algorithm ensures that RC remains computationally tractable while capturing the most conservative defeasible conclusions [8].

To illustrate, consider again the Tesla KB.

From the base rank analysis in Section 2.3, *Tesla* is classified as exceptional with respect to the default that cars emit CO_2. RC then evaluates queries by comparing these ranks. For the query

$$\mathcal{K} \mid\approx_{RC} \text{Tesla} \mid\sim \text{car},$$

the entailment holds, since the antecedent Tesla is not more exceptional than cases where it holds. In other words, despite Tesla being exceptional with regard to emissions, it is still treated as a typical instance of a car.

By contrast, the query

$$\mathcal{K} \mid\approx_{RC} \text{Tesla} \mid\sim \text{emits}CO_2$$

does not hold, because the base rank of *Tesla* is higher than that of cases where Teslas fail to emit CO_2.

This shows how RC preserves safe inferences, such as "Teslas are cars," while rejecting unsafe defaults, such as "Teslas emit CO_2," thereby capturing a conservative but reliable notion of defeasible reasoning.

RC has several important properties. It is *conservative*, meaning that any conclusion entailed by RC will also be entailed by stronger closure methods [8]. It satisfies all KLM rationality postulates, thereby ensuring *Lehmann–Magidor rationality* [20]. In terms of performance, RC is *computationally efficient*, requiring only a polynomial number of calls to a classical entailment checker, so it is no harder than classical reasoning [8]. Finally, it exhibits *syntax insensitivity*, as entailment depends only on the logical content of the KB and not on its particular syntactic representation [4].

Despite these strengths, existing implementations of RC often struggle with inefficiency, poor scalability, and, most relevant to this paper, limited usability [8]. This is important because without accessible explanations, RC remains largely confined to theory. To address this, our work develops an explanatory debugger that preserves RC's formal correctness while adding step-by-step, natural language narratives and interactive visualisations, described in Section 4.

2.5 User Interface and Debugger

The usability of defeasible reasoning systems depends on the design of intuitive user interfaces (UIs) and effective debugging tools. These components allow users to construct their own KBs, issue queries, and inspect the steps behind reasoning outputs.

Reasoning with defaults inevitably involves exceptions and conflicting information, which can make results difficult to interpret. This creates a persistent gap between formal representations of logic and the way humans naturally reason about information [19].

The demand for explainable AI (XAI) has grown significantly across both technical and socio-legal domains. Adamson warns that opaque "black box" systems undermine trust, particularly in sensitive domains like law, because post-hoc explanations tend to be partial or misleading [1]. Marques-Silva and Ignatiev argue for formal approaches to XAI, highlighting that many popular techniques

lack guarantees of correctness or relevance, and that trustworthy AI requires explanations that are provably faithful to the underlying reasoning [23]. Complementing these perspectives, Rizzo and Longo show that knowledge-driven methods such as defeasible argumentation can achieve a higher degree of explainability compared to alternative non-monotonic frameworks, making them particularly valuable for domains that require transparency and user trust [26].

This work underlines the importance of designing reasoning systems that are not only correct, but also interpretable. In this context, the present project contributes by providing a natural language explanation layer for RC, outputs remain formally sound while also accessible to non-expert users.

These considerations motivate the system developed in this paper, which integrates RC reasoning with an interactive debugger to improve transparency and accessibility. Specific related systems that address usability and debugging are discussed in Section 3.

3 RELATED WORK

3.1 Defeasible Reasoning Interfaces

A number of systems have been developed to improve the interpretability of non-monotonic reasoning by providing user-facing interfaces. SILK UI, for example, offers a graphical interface for tracing reasoning paths, thereby enhancing interpretability through visualisation of inference structures [17]. While powerful, SILK presents results primarily in formal symbolic form, which can be inaccessible to non-experts.

Protégé is a widely used ontology editor and knowledge management environment that OntoDebug extends to detects missing or inconsistent information by posing clarification questions to users [28]. Subsequent developments expanded OntoDebug to handle exceptions at multiple layers of ontology reasoning [11]. However, OntoDebug does not expose the reasoning process step-by-step for defeasible entailments, as our system does. Moreover, it requires significant ontology expertise and is not generalisable to Rational Closure.

The *Defeasible Inference Platform (DIP)* implements RC for OWL ontologies, while the *KLM Defeasible Justification Tool* highlights minimal subsets of formulas that justify entailments through a graphical interface [32]. Both tools improve transparency, but remain limited to symbolic outputs without natural language support.

Recent work on large language models (LLMs) has explored chain-of-thought explanations as a way to improve interpretability [22]. While such methods can provide intuition, they lack guarantees of consistency with formal entailment. By contrast, our system anchors explanations directly in the RC algorithm: each natural language explanation is generated from the same intermediate structures used in the formal computation. This ensures both interpretability and faithfulness.

3.2 Implementations of Rational Closure

Several systems directly implement RC reasoning. The *TweetyProject* provides a comprehensive Java-based library supporting propositional logic and RC entailment, and is widely used for experimentation and integration in back-end systems [31]. While it provides the reasoning back-end that our work builds upon, it lacks an accessible front-end for non-expert users, motivating our contribution.

The DIP extends RC to description logics, implementing it for ALC and thereby enabling defeasible reasoning in OWL ontologies [7]. The *KLM Defeasible Justification Tool* similarly offers a graphical interface for RC in propositional logic, visualising minimal subsets of formulas that justify entailments [32]. Although these tools make justifications visible, they do not provide step-by-step tutorials that guide users through the reasoning process.

Together, these implementations demonstrate the applicability of RC across different logical formalisms and contexts, and they provide a foundation for further developments in scalability and usability. However, users are expected to hold expert knowledge on RC and they expose reasoning primarily through symbolic artefacts. Our contribution is complementary: a dual-modal interface that combines mathematically rendered definitions natural language explanations.

4 SYSTEM DESIGN AND IMPLEMENTATION

4.1 Aim

4.1.1 Design and implement a web-based reasoning tool. The primary aim of this project was to design and implement an interactive, web-based environment for defeasible reasoning, focussing on RC. The system was developed to present reasoning steps in a transparent and accessible manner by combining both textual and logical explanations. This dual-modal approach is intended to lower the barrier to entry for non-experts[7].

4.1.2 Support debugging and educational use. A further aim of the system was to support debugging and educational contexts by incorporating natural language explanations. The interface allows users to trace how a RC conclusion was derived, exposing intermediate stages including base rank partitions and the removal of formulas. In this way, the tool is intended not only as a reasoning engine but also as a tool for exploring KBs [11].

4.1.3 Integration with Other EXTRC Project Components. Where feasible, the user interface should support integration with additional modules from the broader EXTRC project, enabling a more unified system. These include the optimisation of rational closure and a natural language knowledge base generator.

4.2 Requirements

4.2.1 Functional Requirements. The system is required to accept KBs composed of both classical and defeasible formulas, and to allow users to query entailments. It must not only return whether an entailment holds under RC, but also present the intermediate stages of reasoning, such as the computed base rank partitions and the formulas that are removed at each stage. This ensures that users are able to trace the path from the initial KB to the final conclusion. In addition, the tool will provide an accessible interface for entering KBs, submitting queries, and reviewing results. This ensures support for both expert use and educational exploration.

4.2.2 Non-Functional Requirements. Beyond its core functionality, the system is expected to meet several non-functional requirements. To maintain usability, the interface should remain intuitive and

responsive, even when reasoning over moderately complex KBs. Performance is also an essential requirement as the system should deliver results efficiently without imposing excessive delays on the user. Another requirement is extendability: the architecture should permit the addition of further reasoning algorithms, such as Lexicographic Closure, without a major redesign. Finally, transparency and clarity should guide all aspects of the system, ensuring that the output remains interpretable. To illustrate this requirement, the design also anticipates optional extensions such as natural language KB integration and RC optimisation, demonstrating the intended extendability of the system.

In summary, the functional and non-functional requirements of the system reflect its dual purpose: to serve as both a practical reasoning tool and an educational aid. By combining algorithmic transparency with usability and extensibility, the system aims to bridge the gap between formal logic and user understanding, offering a platform where defeasible reasoning can be explored, debugged, and explained in an accessible manner.

4.3 Architecture

The system followed a layered architecture. In the back end, a Spring Boot application in Java exposed RESTful APIs for reasoning services. Spring Boot was chosen because it integrated seamlessly with the Java-based TweetyProject library, avoiding cross-language complexity. Its mature ecosystem also supported modular service development for future extensions. Logical inference relied on the TweetyProject library, which provides established implementations of defeasible entailment algorithms and is well supported in the research literature [31], ensuring both correctness and extendability. The front end was developed in React and TypeScript to provide a responsive user interface for entering KBs and submitting queries. These technologies were selected for their support of interactive design, type safety, and modular components. The layered structure, illustrated in Figure 2, promoted modularity, simplified maintenance, and created a clear pathway for future extensions.

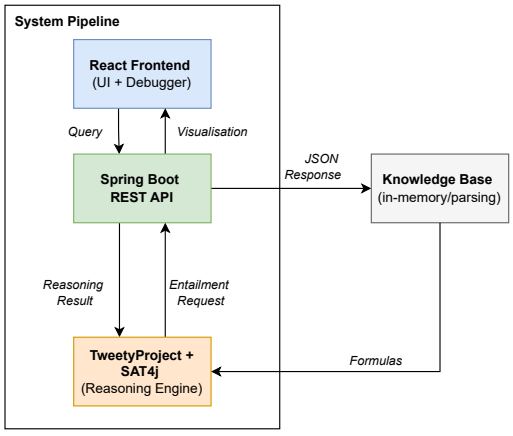

Figure 2: System architecture of the defeasible reasoning tool.

4.3.1 Back-end Implementation. The back end was implemented in Java with Spring Boot, which exposed reasoning functionality via REST endpoints. Controllers handled HTTP requests and delegated them to service classes, which then invoked the *TweetyProject* library for logical inference. This design ensured separation of concerns: controllers remained lightweight, while services encapsulated reasoning logic and delegated computations to the TweetyProject. Endpoints were provided for both base-rank computation and RC entailment, and responses were returned in JSON format for consumption by the front end.

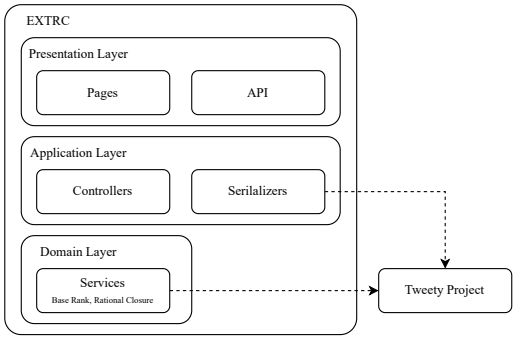

Figure 3: Layered architecture of the EXTRC system.

4.3.2 Front-end Implementation. The front end was implemented in React and TypeScript to deliver a responsive and interactive interface. The layout consisted of three main panels: a KB panel for entering formulas, a query panel for specifying entailment queries, and an output panel that displayed results. Outputs were presented in multiple forms, including tabular partitions of base-rank creation, RC outcomes, and textual explanations. Tabs allowed users to switch between raw logical outputs and explanatory views, supporting both debugging and educational exploration. Figure 5 illustrates how a user entered queries and KBs.

Algorithm 1 RC Debugger Pipeline (simplified)

KB $\mathcal{K}$, query $\alpha \mid\sim \beta$ Verdict (entailed / not entailed), Explanation
$(R_0, \ldots, R_n) \leftarrow \texttt{BaseRank}(\mathcal{K})$
for $i \leftarrow 0$ n **do**
if $R_i \cup \cdots \cup R_n \models \alpha \rightarrow \beta$ **then return** Entailed,
Explanation$(R_0 \ldots R_n, \alpha \mid\sim \beta)$ **return** Not entailed,
Explanation$(R_0 \ldots R_n, \alpha \mid\sim \beta)$

This high-level pseudocode summarises the reasoning pipeline. Full technical details of the `BaseRank` and `RationalClosure` algorithms are given in Appendix A (Algorithms 1 and 2).

4.4 Worked Example: Tesla KB

To demonstrate the debugger in practice, we revisit the Tesla KB introduced in Section 2.2:

$$\mathcal{K} = \{\ car \mid\sim emitsCO_2,\ \ Tesla \rightarrow car,\ \ Tesla \mid\sim \neg emitsCO_2\ \}.$$

The BaseRank procedure partitions the KB into three levels: the general rule that cars emit CO_2 in R_0, the Tesla exception in R_1, and the strict rule $Tesla \longrightarrow car$ in R_∞ (Figure 4). RC then evaluates queries by iteratively testing whether the antecedent is exceptional, and finally checking classical entailment over the remaining rules.

Debugger output. For the query $Tesla \mid\sim car$, the algorithm proceeds as follows:

(1) Compute the partition: $R_0 = \{car \mid\sim emitsCO_2\}$, $R_1 = \{Tesla \mid\sim \neg emitsCO_2\}$, $R_\infty = \{Tesla \longrightarrow car\}$.
(2) Check whether *Tesla* is exceptional at $R_0 \cup R_1 \cup R_\infty$. It is not.
(3) Perform the classical entailment test: $\overline{R_0 \cup R_1 \cup R_\infty} \models Tesla \longrightarrow car$. This holds because the strict rule is present.

The debugger therefore reports that the entailment holds, with the accompanying explanation: *"Teslas are (typically) cars. No counterexamples remain, and the strict rule ensures Teslas are classified as cars."*

For the query $Tesla \mid\sim emitsCO_2$, the process differs:

(1) Compute the same base-rank partition.
(2) At R_0, *Tesla* is exceptional with respect to the rule that cars emit CO_2.
(3) Remove R_0 and continue with $R_1 \cup R_\infty$.
(4) In this reduced set, $\overline{R_1 \cup R_\infty} \not\models Tesla \longrightarrow emitsCO_2$, since the Tesla-specific exception blocks the general rule.

The debugger therefore explains why the entailment does not hold: *"Although cars normally emit CO_2, Teslas are an exception. The specific Tesla rule overrides the general one, so Teslas are not treated as typical CO_2 emitters."*

R_0	$car \mid\sim emitsCO_2$
R_1	$Tesla \mid\sim \neg emitsCO_2$
R_∞	$Tesla \longrightarrow car$

Figure 4: Base-rank partitions for the Tesla KB as displayed in the debugger.

This worked example illustrates how the debugger mirrors the Rational Closure algorithm: it exposes the partitioning into ranks, checks exceptionality iteratively, and then performs a final entailment test. The system not only reproduces RC's results but also narrates the reasoning process step by step in natural languag

4.5 Interface Design for Usability

The interface was structured into three clear panels: KB input, query entry, and results. This separation mirrors the reasoning pipeline and reduces cognitive load. Results are shown in both mathematical definitions and natural language explanations, supporting both expert and novice users. Colour coding highlights entailments, and collapsible views in the Base Rank Explorer let users expand detail only when needed. Concise text, consistent terminology, and a responsive layout were adopted to ensure readability across devices.

These design choices reflect established usability principles. For example, Nielsen's heuristics *emphasise consistency and standards* as well as *visibility of system status*, both of which guided the use of consistent terminology and real-time highlighting [24]. Similarly, Shneiderman's Eight Golden Rules highlight the importance of *reducing short-term memory load* and *striving for consistency*, principles applied in the separation of panels and collapsible views [29].

In this way, the interface balances functionality for debugging with accessibility for teaching.

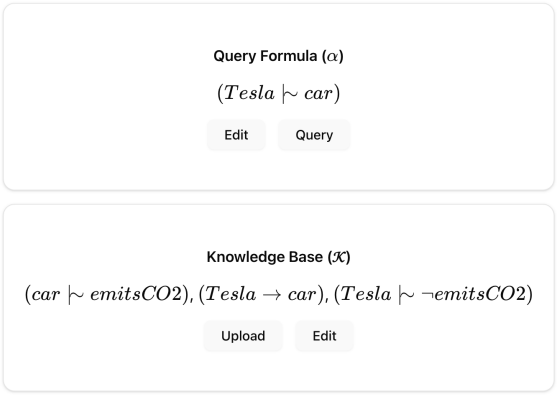

Figure 5: User input for Query and KB

4.5.1 Considered Alternatives. During development we evaluated alternative frameworks and libraries, including Angular for the front end and Javalin for the back end. Angular was ultimately rejected in favour of React because React's component model offered greater flexibility for incremental UI development and easier integration with TypeScript [27]. Similarly, Javalin was considered but Spring Boot was chosen because it provides stronger support for integration, robustness, and long-term maintainability [5]. These decisions allowed us to prioritise clarity and maintainability while keeping the system accessible to future contributors.

5 RESULTS

5.1 Output Examples

Once the user has entered data as seen in Figure 5, the system generates interactive outputs that include base rank partitions, defeasible entailments, and explanatory text. These outputs are organised into three main views: *Summary*, *Base Rank*, and *RC*. An example of the *Summary* view is shown in Figure 6, where entailment results, base rank partitions, and explanatory text are presented together. Information is presented in complementary forms including raw logical partitions, symbolic entailment results, and natural language explanations. This helps support both debugging for expert users and pedagogical exploration for those engaging with defeasible reasoning for the first time. In addition to custom inputs, the system provides a library of preloaded KBs the user can select when exploring the system. As shown in Appendix Figure 11, these examples are expressed in more natural language, allowing users to experiment with reasoning tasks immediately while simultaneously illustrating common patterns of defeasible inference.

5.2 Algorithm Outputs

Algorithm outputs include the partitions generated by the base rank procedure and the corresponding RC entailments. The base

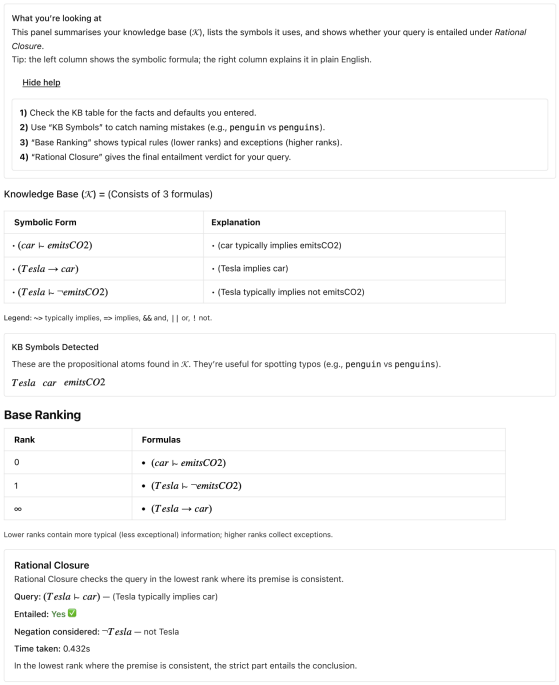

Figure 6: Summary view combining entailment results, base ranks, and explanations

rank splits the KB into $R_0, R_1, \ldots, R_\infty$, and these partitions are then used to test defeasible entailments. The system also shows the intermediate steps of this process, such as which formulas were removed and which were kept. This allows users to determine how the final RC result was reached [8]. This is supported by the "Base Rank Explorer," which walks through each stage of the ranking step-by-step, making the reasoning both transparent and easy to follow.

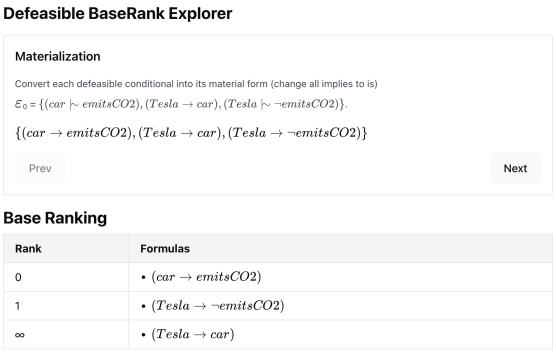

Figure 7: Interactive Base Rank Explorer showing the step-by-step construction of ranks

5.3 Explanatory Features

A distinctive feature of the system is its ability to link algorithmic steps with explanatory text. For example, when a formula is removed during ranking, the system highlights the conflict and generates a short rationale for why that step was necessary. Figure 7 illustrates this in the Base Rank Explorer: when the Tesla default conflicts with the general car rule, the interface explicitly marks the Tesla rule as exceptional and explains why it is lifted to a higher rank.

Similarly, Rational Closure outputs are accompanied by detailed explanations that clarify why an entailment does or does not hold. Figure 8 shows this process for the query $Tesla \mid\sim car$. The debugger presents the computed ranks, identifies whether the antecedent is exceptional, and then explains the outcome of the final entailment check in plain language. If the entailment holds, the system reports which rules guarantee it; if not, it highlights the specific exceptions that block it.

These explanatory features reinforce transparency by connecting the formal reasoning process to an accessible narrative, and support the system's dual purpose as both a debugger and a teaching tool.

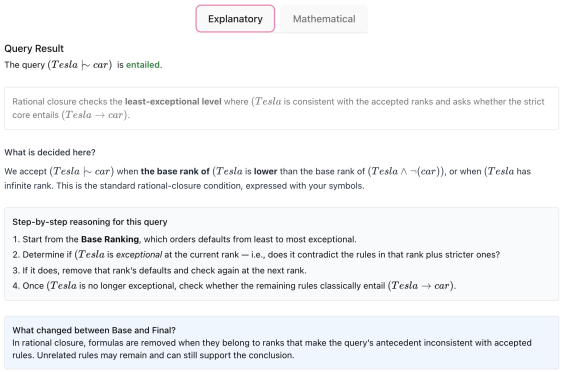

Figure 8: Natural language explanations accompanying Rational Closure results

5.4 Mathematical Features

A further distinctive feature is the use of dual-modal explanations, combining natural language with formal mathematical renderings of base rank and RC definitions. In addition to natural language explanations, the system provides a *mathematical explanation mode* that allows users to start understanding the algorithm alongside the maths. This mode displays the definitions of base rank and RC in their formal notation, with the user's query embedded directly in the symbolic framework (Appendix Figures 14 and 15), following the formal presentations in the literature [15]..

There are two purposes of this feature: first, it offers expert users a precise and rigorous representation of the reasoning process that aligns directly with the formal definitions in the literature; second, it enables learners to compare natural language explanations with their mathematical counterparts, supporting a multimodal understanding of defeasible reasoning. By providing both accessible and formal perspectives, the system accommodates diverse user needs

and strengthens its role as both a teaching tool and a reasoning debugger.

5.5 User Interaction

The system guides the user through a structured workflow that mirrors the process of defeasible reasoning. On entering the application, the user can navigate to the tutorial page that provides an introduction to the interface and explains the purpose of each panel. This acts as an onboarding step, allowing new users to understand the reasoning concepts and system features before engaging with the tool directly. This reflects established usability principles such as Nielsen's heuristic of providing visibility of system status [**?**] and Shneiderman's rule of reducing short-term memory load [29].

From there, the syntax page offers a concise reference for how to encode KBs and queries, ensuring that users can translate reasoning tasks into the system's supported formalism without ambiguity. It also provides definitions for unfamiliar symbols. These all reduce the barrier to entry, particularly for those unfamiliar with the exact logical notation.

To further support exploration, the application includes a set of preloaded KBs expressed in more natural language. These examples allow users to quickly test the system without needing to construct their own KB from scratch, while also serving as pedagogical demonstrations of common reasoning patterns.

Once equipped with this context, the user moves to the three main functional panels: the KB editor, the query input panel, and the output explorer. This sequence reflects the logical flow of reasoning: specifying assumptions, formulating queries, and interpreting results. The output explorer integrates the features described above, including algorithm outputs such as the base rank partitions and RC entailments, as well as explanatory text which highlights removed formulas and clarifies why specific steps were taken. Switching between the summary, base rank, and RC tabs provides multiple perspectives on the reasoning process. This enables users to move fluidly between high-level overviews and detailed justifications.

As a whole, this walk-through design not only supports advanced debugging but also functions as a learning pathway, gradually building user understanding from syntax and examples toward full reasoning tasks. This emphasis on usability and pedagogy reinforces the system's role as both an educational tool and a reasoning debugger.

5.6 Incorporation of Wider EXTRC Project

A further contribution of this work has been the incorporation of components from the wider EXTRC project. First, the system integrates a suite of enhanced entailment strategies (Naïve, Binary, Ternary, Hybrid, Cached, and SAT-based), with their execution times surfaced directly in the user interface. These strategies were originally developed as part of ongoing optimisation work within EXTRC [16], and their inclusion transforms the debugger into both an educational tool and a lightweight benchmarking environment, illustrating performance trade-offs between alternative approaches.

In addition, the debugger has been extended to interface with the EXTRC natural language knowledge base generator [12]. This enhancement maps symbolic formulas to human-readable phrases, enabling explanations that are more accessible and domain-aware. Although not part of the original specification, this integration demonstrates the modularity of the design and highlights the potential for richer, user-facing explanations.

Together, these incorporations show how the interface can act as a unifying front end for multiple EXTRC components, consolidating symbolic reasoning, optimisation strategies, and natural language explanations within a single interactive environment.

Overall, the system demonstrates how Rational Closure can be made accessible through a combination of formal outputs, natural language explanations, and pedagogical design. By surfacing intermediate steps, supporting dual-modal views, and integrating with related EXTRC components, the debugger advances beyond existing implementations to provide both a practical reasoning engine and an educational resource.

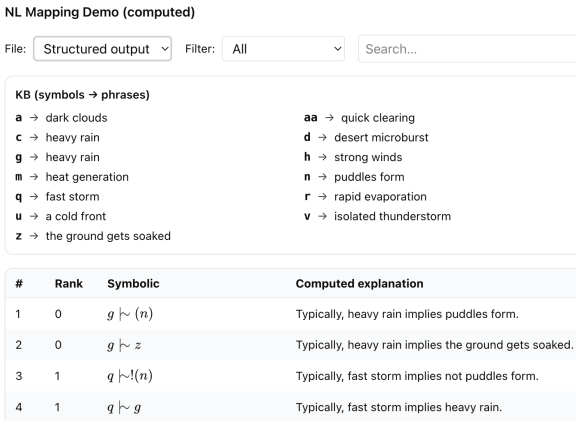

Figure 9: Integration with Natural Language Knowledge Base

6 TESTING AND EVALUATION

6.1 Correctness

Correctness of the system was verified through a combination of unit and integration testing. For the back-end level, JUnit tests were written for the service layer to ensure that entailment logic and base rank computation were implemented correctly. Controller tests were used to validate the REST API, checking that endpoints responded as expected and that input was correctly parsed and passed to the reasoning service (see Appendix Figure 18). For the front-end, integration tests confirmed that queries submitted through the user interface produced the same results as direct calls to the TweetyProject reasoning engine. Together, these tests ensured that both individual components and the overall pipeline behaved as intended.

6.2 Performance

Performance was evaluated by measuring execution time for RC reasoning across increasing KB sizes, in both local and deployed environments. Figure 10 compares average response times for queries executed on a local machine and in the deployed application on Render.

On the local host, execution times remained consistently low, well under 100 ms even for KBs of up to 100 formulas. This indicates

that the RC implementation itself scaled efficiently and does not incur significant overhead as the KB size increases. By contrast, the deployed version exhibited markedly slower response times, ranging between 300–800 ms per query, with variability independent of KB size. This suggests that the primary bottleneck lies in network latency and resource constraints imposed by the hosting environment rather than the reasoning algorithm.

The graph shows that both environments handle small to medium KBs without a noticeable degradation in performance, but that the deployed environment introduces substantial fixed overhead. Memory monitoring further confirmed that the Render deployment was limited by free-tier quotas, preventing larger-scale tests beyond 50 formulas.

Overall, these results demonstrate that RC is computationally efficient, but that practical scalability in a deployed setting is highly dependent on infrastructure choices. For research and educational use, local execution is sufficient, while production deployment would require more robust hosting resources.

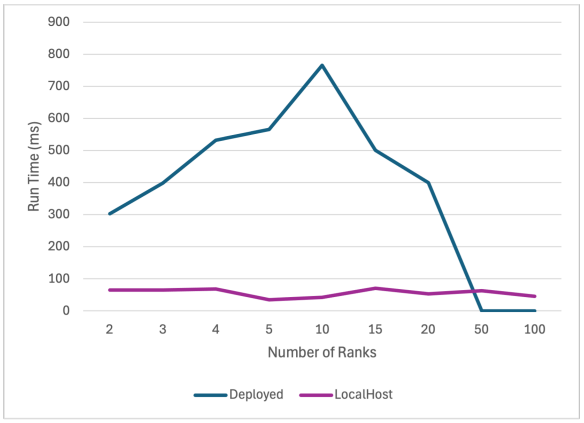

Figure 10: Performance of RC on Local Host and Deployment

6.3 Usability

Usability was assessed through qualitative feedback obtained from expert users, including the project supervisor and colleagues. Evaluation focused on the clarity of the user interface, the transparency of explanations provided by the debugger, and the accessibility of RC reasoning for non-experts. Feedback suggested that the dual-modal presentation of reasoning steps (textual and logical) improved interpretability, although further refinement of natural language explanations remains a potential area for improvement.

6.3.1 Evaluation of the Natural Language Layer. The natural language explanation mode was evaluated qualitatively, since no formal user study was undertaken. Instead, validation focused on faithfulness: each explanation is generated directly from the intermediate structures produced by the RC algorithm, ensuring that textual rationales remain consistent with the underlying formal outputs. This distinguishes the system from approaches such as LLM-generated chain-of-thought, where explanations may diverge from the formal semantics.

6.4 Integration Testing

Integration testing evaluated the system as a complete pipeline, from KB input to reasoning output and debugger explanations. End-to-end tests confirmed that inputs were consistently parsed, processed by the back-end, and displayed correctly in the user interface. Error handling was also tested, with invalid formulas and malformed queries producing appropriate error messages rather than system failures. These evaluations confirmed the reliability of communication between the React front-end, Spring Boot REST API, and the TweetyProject reasoning engine.

6.5 Alternative Evaluation Approach

No formal usability study was undertaken, as this would have required ethical clearance outside the scope of the project. Instead, evaluation focused on methods consistent with the system's aims. Correctness was verified through unit and integration testing at both the back-end and front-end, ensuring outputs matched the formal definitions of base rank and RC. Performance was assessed by measuring execution times across increasing KB sizes, confirming that algorithmic scalability was sound and that bottlenecks arose mainly from hosting constraints. Usability was evaluated indirectly: natural language explanations were checked for alignment with formal reasoning steps, and informal expert feedback helped refine clarity. While less systematic than a user study, this approach provided evidence of correctness, transparency, and accessibility, with a structured user evaluation left as a clear avenue for future work. Overall, the evaluation confirmed that the system is correct, performant, and transparent in its outputs. While informal expert feedback validated the clarity of natural language explanations, a structured user study remains an important next step. This would allow systematic measurement of the tool's educational impact, complementing the technical evaluation presented here.

7 DISCUSSION

This project demonstrates the feasibility of combining RC reasoning with an interactive, web-based interface. By leveraging a layered architecture, the implementation integrates the TweetyProject reasoning engine with a Spring Boot back-end and a React front-end, providing a full pipeline from KB input to reasoning output. Testing confirmed that the system correctly implemented RC, with outputs consistent with theoretical definitions. The addition of debugging functionality and dual-modal presentation of reasoning steps represents a significant step towards improving the transparency and accessibility of defeasible reasoning. Unlike existing systems such as OntoDebug or DIP, which either assume expert knowledge or present symbolic results only, this tool anchors its explanations in the RC derivation itself. This avoids the faithfulness issues associated with LLM-generated narratives while lowering the barrier for non-experts

The system contributes to the broader conversation on usability in defeasible reasoning. Prior work has emphasised the gap between formal reasoning and human understanding; this interface operationalises RC not only as a reasoning service but also as an educational and debugging tool. By exposing base ranks and entailment steps, it clarifies the mechanisms behind conclusions, supporting both expert analysis and student learning. The inclusion

of both natural language and mathematical explanations tailors outputs to different audiences: narration improves accessibility for non-specialists, while formal definitions ensure precision and reproducibility for experts. Unlike symbolic-only tools or LLM-based explanations, this system aligns narration directly with derivations, achieving both faithfulness and interpretability. Tutorial-style pages (syntax guides, connective definitions, example KBs) further enhance its role as a teaching resource. Beyond these core aims, the integration of rational closure optimisation strategies and the prototyping of a natural language knowledge base generator illustrate the system's modularity and demonstrate its ability to incorporate components from the wider EXTRC project.

Overall, the results suggest that RC can be embedded effectively in interactive environments, but further work is required to improve scalability, refine natural language explanations, and broaden empirical evaluation. These findings provide a foundation for systems that balance formal rigour with usability and interpretability.

7.1 Limitations

While the system demonstrates the feasibility of providing an interactive and transparent interface for defeasible reasoning, several limitations remain. Reasoning capabilities are restricted to RC, which captures an important form of non-monotonic inference but not alternative approaches such as Relevant and Lexicographic Closure. This constrains the tool's generalisability. Deployment on free-tier platforms (Render and Vercel) also introduced delays and memory limits, though these did not undermine correctness. While debugging is supported through exposure of base ranks and entailment steps, natural language support is still template-based: connector words (e.g. "is typically") may not always sound natural, and singular/plural mismatches can occur. Although faithful, these explanations sometimes lack fluency. The system also depends heavily on the TweetyProject library, which ensures reliability but complicates porting to other reasoning engines.

7.2 Future Work

Future work should extend support beyond RC to frameworks such as Relevant and Lexicographic Closure, enabling comparative analysis of different approaches to defeasible reasoning. More advanced natural language generation, informed by cognitive science and XAI research [2, 6], would improve fluency while retaining faithfulness. Scaling experiments on larger KBs would further validate performance. Integration across group components also remains a priority: optimisation strategies have already been incorporated, but the connector module should be revisited to complete end-to-end workflows. Finally, the integration of the natural language knowledge base generator could be developed further. At present it remains a separate feature, but embedding it more directly into the query workflow would allow explanations to flow more naturally from symbolic input to human-readable output that allow explanations to vary their phrasing automatically (e.g. "usually," "normally," "in most cases"). Overall, future work may therefore focus on optimising reasoning performance, exploring closer integration of natural language interfaces, undertaking more formal user evaluations, and extending support to additional entailment algorithms.

Incorporating these extensions would not only refine the tool's usability, but also strengthen its role as a unifying front-end for the EXTRC project, where diverse reasoning strategies and explanation methods can be compared in a single environment.

Overall, the project shows the value of visualising defeasible reasoning in an interactive environment, while leaving open multiple avenues for refinement and extension.

8 CONCLUSIONS

This project has presented the design and implementation of an interactive web-based interface for RC reasoning. Building on the KLM framework, the system exposes the reasoning process step-by-step, providing visualisations of base ranks, entailments, and conflicts within a KB. By prioritising clarity and accessibility, the tool demonstrates how defeasible reasoning, often regarded as abstract and inaccessible, can be made transparent for both researchers and learners.

The central contribution of this work lies in bridging the gap between formal theory and practical usability. Existing systems typically require an expert level of knowledge, present results in symbolic formats, or lack support for debugging. In contrast, this interface integrates explanation, interactivity, and visualisation, thereby offering both an educational resource and a practical aid for those working with defeasible reasoning. In doing so, it contributes to broader efforts in XAI by showing that formal reasoning can be made interpretable without sacrificing rigour. All primary deliverables were achieved: a deployed debugger, a dual-mode explanation system, performance evaluation, and alignment verification. In addition, the system was extended to incorporate two components from the wider EXTRC project: a suite of enhanced entailment strategies and the natural language knowledge base generator. Surfacing execution times makes the interface a benchmarking tool, while the KB generator enriches explanations. These extensions, though not part of the original specification, demonstrate the modularity of the framework and highlight its potential to consolidate different strands of the EXTRC project into a unified interactive environment.

There are, however, limitations. The current implementation remains constrained by the scalability of RC and the reliance on a classical entailment back end. Usability testing has so far been limited to informal evaluation, and a more systematic user study would strengthen the assessment of its educational and practical value. The natural language knowledge base generator could be tied into the queries rather than being a separate page. These limitations suggest directions for future work, including optimisation of reasoning performance, deeper integration of natural language knowledge base generator, broader empirical evaluation, and the extension of additional entailment algorithms.

Despite these constraints, the work demonstrates that RC can be implemented in a way that is both theoretically faithful and understandable. By providing a concrete tool that makes defeasible reasoning more accessible, this paper takes a step toward narrowing the gap between logical formalisms and real-world reasoning needs. In this way, it highlights the value of combining technical precision with human-centred design in the ongoing development of knowledge representation and reasoning systems.

REFERENCES

[1] Greg Adamson. Explainable artificial intelligence (xai): A reason to believe? *Law in Context: A Socio-Legal Journal*, 37(3):23–44, 2022.

[2] Clayton Kevin Baker, Claire Denny, Paul Freund, and Thomas Meyer. Cognitive defeasible reasoning: the extent to which forms of defeasible reasoning correspond with human reasoning. In *Southern African Conference for Artificial Intelligence Research*, pages 199–219. Springer, 2020.

[3] Mordechai Ben-Ari. *Mathematical logic for computer science*. Springer Science & Business Media, 2012.

[4] Salem Benferhat, Didier Dubois, and Henri Prade. Some syntactic approaches to the handling of inconsistent knowledge bases: A comparative study part 1: The flat case. *Studia Logica*, 58(1):17–45, 1997.

[5] M. Blaszczyk, Marko Puček, and P. Kopniak. Comparison of lightweight frameworks for java by analyzing proprietary web applications. *Journal of Computer Sciences Institute*, 2021.

[6] Giovanni Casini, Michael Harrison, Thomas Meyer, and Reid Swan. Arbitrary ranking of defeasible subsumption. 2019.

[7] Giovanni Casini, Thomas Meyer, Kody Moodley, Uli Sattler, and Ivan Varzinczak. Introducing defeasibility into owl ontologies. In *The Semantic Web-ISWC 2015: 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II 14*, pages 409–426. Springer, 2015.

[8] Giovanni Casini, Thomas Meyer, and Ivan Varzinczak. Taking defeasible entailment beyond rational closure. In *Logics in Artificial Intelligence: 16th European Conference, JELIA 2019, Rende, Italy, May 7–11, 2019, Proceedings 16*, pages 182–197. Springer, 2019.

[9] Giovanni Casini and Umberto Straccia. Rational closure for defeasible description logics. In *Logics in Artificial Intelligence: 12th European Conference, JELIA 2010, Helsinki, Finland, September 13-15, 2010. Proceedings 12*, pages 77–90. Springer, 2010.

[10] Giovanni Casini and Umberto Straccia. Lexicographic closure for defeasible description logics. In *Proc. of Australasian Ontology Workshop*, volume 969, pages 28–39, 2012.

[11] Simone Coetzer and Katarina Britz. Debugging classical ontologies using defeasible reasoning tools. In *Formal Ontology in Information Systems (FOIS 2022)*, volume 344, pages 97–106. IOS Press, 2022.

[12] Jethro Dunn. Implementing natural language into knowledge base generation. Honours Project, University of Cape Town, 2025.

[13] Lloyd Everett, Emily Morris, and Thomas Meyer. Explanation for KLM-style defeasible reasoning. In Edgar Jembere, Aurona J. Gerber, Serestina Viriri, and Anban Pillay, editors, *Artificial Intelligence Research*, pages 192–207. Springer International Publishing, Cham, 2022.

[14] Michael Freund. Preferential reasoning in the perspective of poole default logic. *Artificial Intelligence*, 98(1-2):209–235, 1998.

[15] Laura Giordano, Valentina Gliozzi, Nicola Olivetti, and Gian Luca Pozzato. Semantic characterization of rational closure: From propositional logic to description logics. *Artificial Intelligence*, 226:1–33, 2015.

[16] Nevaniah Gounden. Investigation of optimisation techniques for rational closure in defeasible reasoning. Honours Project, University of Cape Town, 2025.

[17] Benjamin Grosof, Mark Burstein, Mike Dean, Carl Andersen, Brett Benyo, William Ferguson, Daniela Inclezan, and Richard Shapiro. A silk graphical ui for defeasible reasoning, with a biology causal process example. In *9th International Semantic Web Conference ISWC 2010*, page 113. Citeseer, 2010.

[18] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial intelligence*, 44(1-2):167–207, 1990.

[19] Daniel Lehmann. Another perspective on default reasoning. *Annals of mathematics and artificial intelligence*, 15:61–82, 1995.

[20] Daniel Lehmann and Menachem Magidor. What does a conditional knowledge base entail? *Artificial intelligence*, 55(1):1–60, 1992.

[21] Hector J Levesque. Knowledge representation and reasoning. *Annual review of computer science*, 1(1):255–287, 1986.

[22] Zhaoqun Li, Chen Chen, Mengze Li, and Beishui Liao. Exploring formal defeasible reasoning of large language models: A chain-of-thought approach. *Knowledge-Based Systems*, page 113564, 2025.

[23] Joao Marques-Silva and Alexey Ignatiev. Delivering trustworthy ai through formal xai. In *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI-22)*, pages 12342–12350. AAAI Press, 2022.

[24] Jakob Nielsen and Robert L. Mack. *Usability Inspection Methods*. John Wiley & Sons, 1994.

[25] John L Pollock. Defeasible reasoning. *Cognitive science*, 11(4):481–518, 1987.

[26] Lucas Rizzo and Luca Longo. A qualitative investigation of the degree of explainability of defeasible argumentation and non-monotonic fuzzy reasoning. In *Proceedings of the 26th Irish Conference on Artificial Intelligence and Cognitive Science (AICS)*, pages 138–149, 2018.

[27] E. Saks. Javascript frameworks: Angular vs react vs vue. 2019.

[28] Konstantin Schekotihin, Patrick Rodler, and Wolfgang Schmid. Ontodebug: Interactive ontology debugging plug-in for protégé. In *Foundations of Information and Knowledge Systems: 10th International Symposium, FoIKS 2018, Budapest, Hungary, May 14–18, 2018, Proceedings 10*, pages 340–359. Springer, 2018.

[29] Ben Shneiderman, Catherine Plaisant, Maxine Cohen, Steven Jacobs, Niklas Elmqvist, and Nicholas Diakopoulos. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Pearson, 6th edition, 2016.

[30] Y. Shoham. A semantical approach to nonmonotonic logics. In M. L. Ginsberg, editor, *Readings in Non-Monotonic Reasoning*, pages 227–249. Morgan Kaufmann, 1987.

[31] Matthias Thimm. The tweety library collection for logical aspects of artificial intelligence and knowledge representation. *KI-Künstliche Intelligenz*, 31(1):93–97, 2017.

[32] Steve Wang. Defeasible justification for the klm framework. Master's thesis, Faculty of Science, University of Cape Town, 2022.

A ALGORITHMS

Algorithm 2 BaseRank adapted from [13]

Require: A knowledge base $\mathcal{K}$
Ensure: An ordered tuple $(\mathcal{R}_0, \ldots, \mathcal{R}_{n-1}, \mathcal{R}_\infty, n)$

$i := 0$
$\mathcal{R}_\infty := \mathcal{K} \setminus \{\alpha \mid\sim \beta \in \mathcal{K}\}$
$\mathcal{E}_0 := \{\alpha \mid\sim \beta \in \mathcal{K}\}$
repeat
    $\mathcal{E}_{i+1} := \{\alpha \mid\sim \beta \in \mathcal{E}_i \mid \mathcal{R}_\infty \cup \mathcal{E}_i \models \neg\alpha\}$
    $\mathcal{R}_i := \mathcal{E}_i \setminus \mathcal{E}_{i+1}$
    $i := i + 1$
until $\mathcal{E}_{i-1} = \mathcal{E}_i$
$\mathcal{R}_\infty := \mathcal{R}_\infty \cup \mathcal{E}_{i-1}$
$n := i - 1$
return $(\mathcal{R}_0, \ldots, \mathcal{R}_{n-1}, \mathcal{R}_\infty, n)$ =0

Algorithm 3 RationalClosure adapted from[13]

Require: A knowledge base $\mathcal{K}$ and a DI $\alpha \mid\sim \beta$
Ensure: **true** if $\mathcal{K} \mid\sim_{\mathrm{RC}} \alpha \vdash \beta$, otherwise **false**

$(\mathcal{R}_0, \ldots, \mathcal{R}_{n-1}, \mathcal{R}_\infty, n) \leftarrow$ BaseRank($\mathcal{K}$)
$i := 0$
$\mathcal{R} := \bigcup_{j=0}^{n-1} \mathcal{R}_j$
while $\mathcal{R}_\infty \cup \mathcal{R} \not\models \neg\alpha$ **and** $\mathcal{R}_\infty \neq \emptyset$ **do**
    $\mathcal{R} := \mathcal{R} \setminus \mathcal{R}_i$
    $i := i + 1$
end while
return $\mathcal{R}_\infty \cup \mathcal{R} \models \alpha \rightarrow \beta$

B SUPPLEMENTARY INFORMATION

B.1 Screenshots

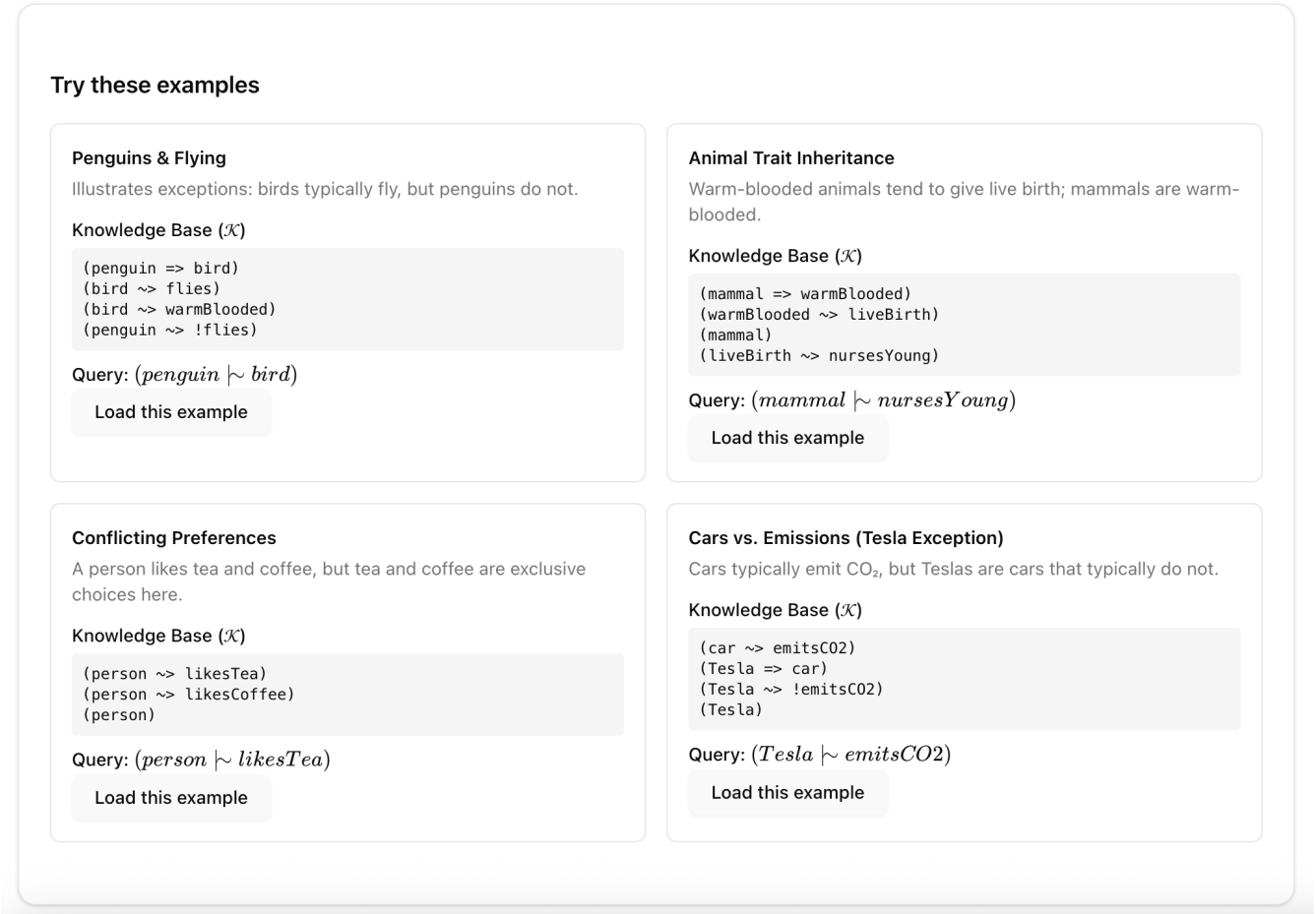

Figure 11: Preloaded KBs expressed in natural language, enabling quick experimentation

Explanatory | Mathematical

The BaseRank algorithm takes your knowledge base $\mathcal{K}$ with **3** formulas. The algorithm organises them into levels $R_0, R_1, \ldots, R_\infty$ by performing an iterative process to assign a base rank to each statement, which indicates its level of typicality or specificity. Valuations with a lower rank are considered more normal or typical than those with a higher rank.

Let's start with unpacking how the base rank is computed

Your current query: $(Tesla \mid\sim car)$

In other words, you are trying to find out if *(Tesla typically are car)* holds, given your current knowledge base.

This means we look for the lowest level r where the knowledge base $\mathcal{E}_r^{\mathcal{K}}$ supports the defeasible inference that $\neg\alpha$ is not typically true.

Defeasible BaseRank Explorer

Materialization

Convert each defeasible conditional into its material form (change all implies to is)

$\mathcal{E}_0 = \{(car \mid\sim emitsCO2), (Tesla \rightarrow car), (Tesla \mid\sim \neg emitsCO2)\}.$

$$\{(car \rightarrow emitsCO2), (Tesla \rightarrow car), (Tesla \rightarrow \neg emitsCO2)\}$$

Prev | Next

Figure 12: Explanation of Base Rank

Initial Base Ranking

Rank	Formulas
0	• $(car \rightarrow emitsCO2)$
1	• $(Tesla \rightarrow \neg emitsCO2)$
∞	• $(Tesla \rightarrow car)$

Removed Formulas

Rank	Formulas
0	• $(car \rightarrow emitsCO2)$

Final Retained Ranking

Rank	Formulas
1	• $(Tesla \mid\sim \neg emitsCO2)$
∞	• $(Tesla \rightarrow car)$

Why this verdict?
The retained rules still allow us to derive $(Tesla \rightarrow car)$ from the strict core at the relevant rank.

Figure 13: Rational Closure output view displaying final defeasible entailments

Explanatory | Mathematical

A propositional sentence α is said to be *exceptional* with respect to $\mathcal{K}$ if $\mathcal{K} \models_R \top \mid\sim \neg\alpha$.

A sequence of knowledge bases $\mathcal{E}_0^{\mathcal{K}}, \mathcal{E}_1^{\mathcal{K}}, \ldots, \mathcal{E}_\infty^{\mathcal{K}}$ is defined as follows:

$$\mathcal{E}_0^{\mathcal{K}} \equiv_{\text{def}} \mathcal{K}$$
$$\mathcal{E}_i^{\mathcal{K}} \equiv_{\text{def}} \varepsilon(\mathcal{E}_{i-1}^{\mathcal{K}}), \quad 0 < i < n$$
$$\mathcal{E}_\infty^{\mathcal{K}} \equiv_{\text{def}} \mathcal{E}_n^{\mathcal{K}} \text{ where } n \text{ is the smallest index such that } \mathcal{E}_n^{\mathcal{K}} = \mathcal{E}_{n+1}^{\mathcal{K}}.$$

The **base rank** $br_{\mathcal{K}}(\alpha)$ of a propositional statement α with respect to $\mathcal{K}$ is defined to be the smallest r such that α is *not exceptional* with respect to $\mathcal{E}_r^{\mathcal{K}}$:

$$br_{\mathcal{K}}(\alpha) \equiv_{\text{def}} \min\{r \mid \mathcal{E}_r^{\mathcal{K}} \not\models_R \top \mid\sim \neg\alpha\}$$

For your current query:

$\alpha = (Tesla \mid\sim car)$
$\mathcal{K} = \{(car \mid\sim emitsCO2), (Tesla \rightarrow car), (Tesla \mid\sim \neg emitsCO2)\}.$

So we are evaluating: $br_{\mathcal{K}}((Tesla \mid\sim car)) = \min\{r \mid \mathcal{E}_r^{\mathcal{K}} \not\models_R \top \mid\sim \neg((Tesla \mid\sim car))\}$

Observation

$\mathcal{K} \models_{RC} \alpha \mid\sim \beta \quad \text{iff} \quad br_{\mathcal{K}}(\alpha) < br_{\mathcal{K}}(\alpha \wedge \neg\beta)$ or $br_{\mathcal{K}}(\alpha) = \infty$.

Figure 14: Mathematical explanation of the Base Rank procedure with formal definitions

Explanatory | Mathematical

Query Result

The query $(Tesla \mid\sim car)$ is **entailed**.

Rational closure can be characterised in terms of base ranks. For a knowledge base $\mathcal{K}$ and formula α, let $br_{\mathcal{K}}(\alpha)$ be the smallest r such that $\mathcal{E}_r^{\mathcal{K}} \not\models_R \top \mid\sim \neg\alpha$, or ∞ if no such r exists.

$$br_{\mathcal{K}}(\alpha) \equiv_{\text{def}} \min\{r \mid \mathcal{E}_r^{\mathcal{K}} \not\models_R \top \mid\sim \neg\alpha\}$$

Then the rational-closure entailment condition is:

$$\mathcal{K} \models_{RC} \alpha \mid\sim \beta \quad \text{iff} \quad br_{\mathcal{K}}(\alpha) < br_{\mathcal{K}}(\alpha \wedge \neg\beta) \;\text{ or }\; br_{\mathcal{K}}(\alpha) = \infty$$

For your current query:
$\alpha = (Tesla$, $\beta = car)$, and
$\mathcal{K} = \{(car \mid\sim emitsCO2), (Tesla \to car), (Tesla \mid\sim \neg emitsCO2)\}$.

Hence we evaluate:
$br_{\mathcal{K}}(Tesla) < br_{\mathcal{K}}(Tesla \wedge \neg(car)) \;\text{ or }\; br_{\mathcal{K}}(Tesla) = \infty$

Figure 15: Mathematical explanation of Rational Closure, embedding user queries into formal definitions

Tutorial

This app helps you explore **defeasible reasoning** — reasoning with typical rules and exceptions. You'll enter a Knowledge Base ($\mathcal{K}$) of statements, choose a query, and see how different reasoning closures handle it.

How the app works

1. **Enter your Knowledge Base ($\mathcal{K}$):** one statement per line. Use $\mid\sim$ for *typical/defeasible* rules (e.g., "Birds ~`>` Fly"). Use $\rightarrow$ for *strict* rules (always true).
2. **Pick a query** you want to test — e.g.,*"If something is a penguin, does it typically fly?"*
3. Click **Query**. The system will compute:
   - **Base Rank:** sorts your rules into levels from most certain (rank 0) to least certain (higher ranks), with impossible ones at ∞.
   - **Rational Closure:** draws cautious conclusions that hold in the most specific consistent scenarios.
4. Check the **Summary**, **Base Rank**, and **Rational Closure** tabs.

Figure 16: Tutorial of Web Application

Definitions

$p,\ q,\ r, \ldots$ Atoms

Lowercase letters (p, q, r, ...) that stand for simple statements that are either true or false.

Example: p

$\top$ Tautology

A formula that is always true in every situation (e.g., 'True'). Useful as a neutral truth.

Example: $\top$

$\bot$ -

Always false in every situation. Represents an impossible or contradictory statement.

Example: $\bot$

$\neg$ Negation

'Not'. The statement ¬p is true exactly when p is false, and false when p is true.

Example: $\neg p$

$\wedge$ Conjunction

'And'. The statement (p ∧ q) is true only when both p and q are true.

Example: $p \wedge q$

$\vee$ Disjunction

'Or' (inclusive). The statement (p ∨ q) is true when at least one of p or q is true (or both).

Example: $p \vee q$

$\rightarrow$ Material implication

'If ... then ...' under classical logic. (p → q) is false only when p is true and q is false; otherwise true.

Example: $p \rightarrow q$

$\leftrightarrow$ Equivalence

'If and only if'. (p ↔ q) is true when p and q have the same truth value (both true or both false).

Example: $p \leftrightarrow q$

$\mid\!\sim$ Defeasible implication

A defeasible rule inside the knowledge base. 'Typically ...'. (p ~> q) says q normally follows from p unless there are stronger exceptions.

Example: $p \mid\!\sim q$

$\mid\!\approx$ Defeasible entailment

Used in queries to ask whether something follows defeasibly from the ranked knowledge base.

Example: $K \mid\!\approx (p \mid\!\sim q)$

Figure 17: List of Definitions on Syntax page

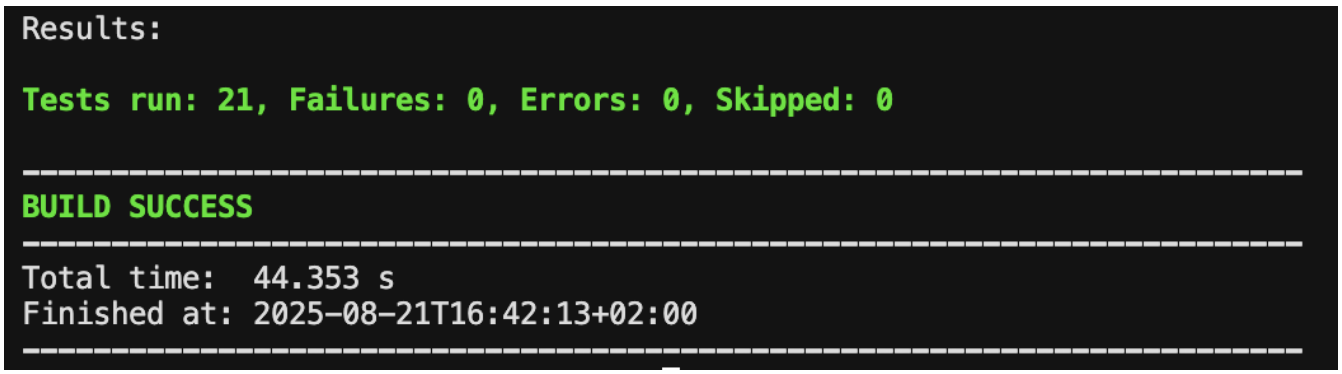

Figure 18: Screenshot of JUnit tests performed

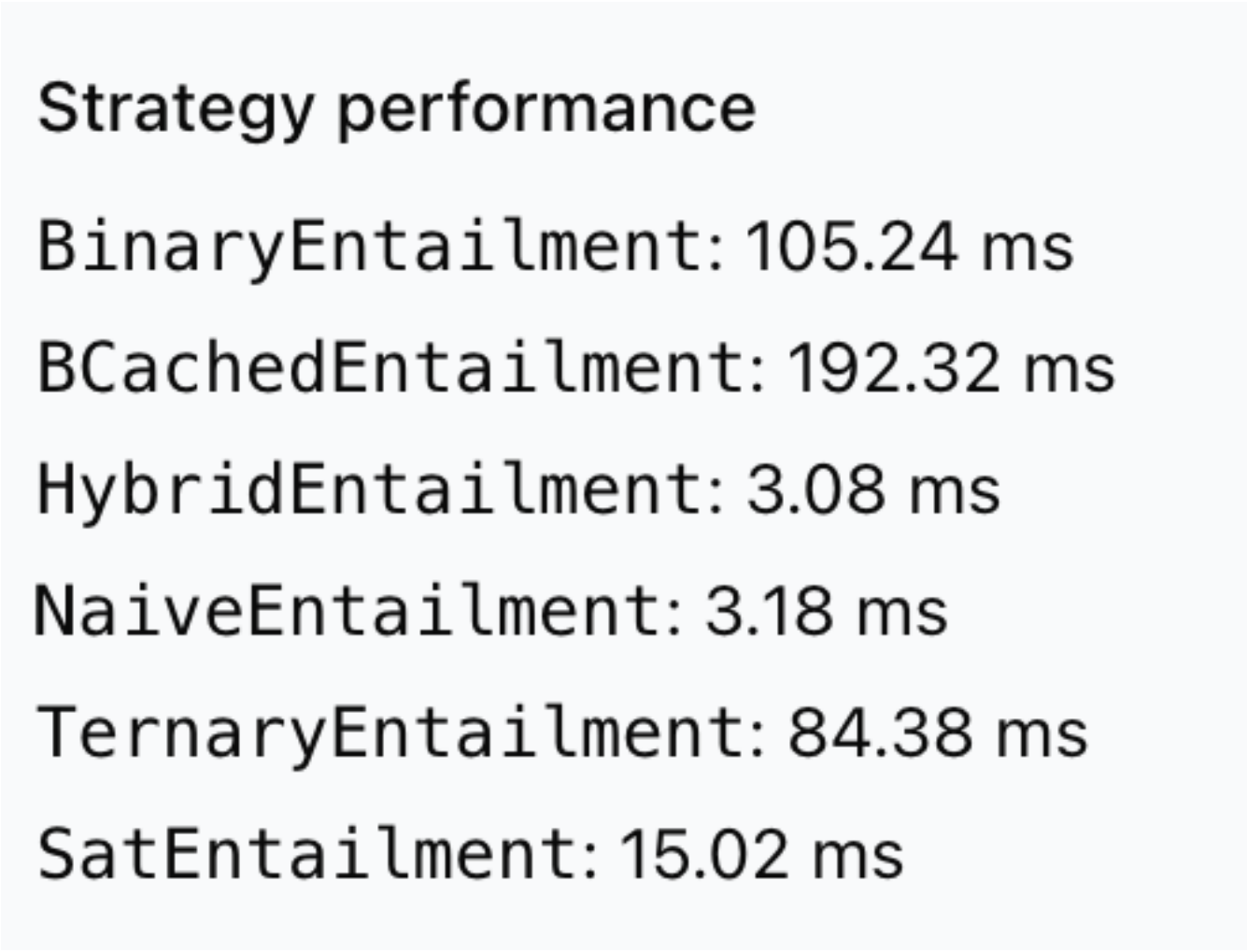

Figure 19: Execution times of enhanced entailment strategies developed in greater EXTRC project