Machine Learning (2026) 115:135
https://doi.org/10.1007/s10994-026-07075-1

Predicate Renaming via Large Language Models

Elisabetta Gentili[1,4] · Tony Ribeiro[2,4] · Fabrizio Riguzzi[3] · Katsumi Inoue[4]

Received: 24 March 2025 / Revised: 11 May 2026 / Accepted: 12 May 2026
© The Author(s) 2026

Abstract
In this paper, we address the problem of giving names to predicates in logic rules using Large Language Models (LLMs). In the context of Inductive Logic Programming, various rule generation methods produce rules containing unnamed predicates, with Predicate Invention being a key example. This hinders the readability, interpretability, and reusability of the logic theory. Leveraging recent advancements in LLMs development, we explore their ability to process natural language and code to provide semantically meaningful suggestions for giving a name to unnamed predicates. The evaluation of our approach on some hand-crafted logic rules indicates that LLMs hold potential for this task.

Keywords Large language models · Inductive logic programming · Predicate invention · Code refactoring

1 Introduction

Logic is an effective approach to modeling domains with complex relationships among entities. Logic Programming (LP) has been a powerful declarative paradigm based on formal logic since the 70 s (Kowalski, 1988). Contrary to the traditional imperative programming

Editor: Steven Schockaert.

✉ Elisabetta Gentili
elisabetta.gentili1@unife.it

Tony Ribeiro
tony.ribeiro@ls2n.fr

Fabrizio Riguzzi
fabrizio.riguzzi@unife.it

Katsumi Inoue
inoue@nii.ac.jp

[1] Department of Engineering, University of Ferrara, Via Saragat 1, 44122 Ferrara, Italy

[2] Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, 44000 Nantes, France

[3] Department of Mathematics and Computer Science, University of Ferrara, Via Machiavelli 30, 44121 Ferrara, Italy

[4] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan

Published online: 23 May 2026

paradigm, where the focus is on *how* the problem should be solved, in logic programming the program is a description of *what* is the problem. Logic programs represent knowledge of a domain using facts and rules to state what is true. An LP system can derive new knowledge by means of logical inference and answer users' queries about knowledge not explicitly encoded in the program. Examples of LP languages include Prolog (Colmerauer & Roussel, 1996), Datalog (Abiteboul et al., 1995), and Answer Set Programming (Gelfond & Lifschitz, 1988). Since the inference procedure is independent of the knowledge base, logic programs are generally flexible, reusable, and easy to maintain and adapt as new requirements arise. New knowledge can be added without changing the inference algorithm. Despite the many advantages, LP also has some challenges. The knowledge base needs to be carefully structured. To mitigate this problem, Inductive Logic Programming (ILP) aims at automatically inducing knowledge bases from data (Muggleton et al., 2012). Various ILP systems have been proposed and have found successful applications (Cropper & Dumančić, 2022) in bioinformatics and drug design (Srinivasan et al., 1996, 1997; Inoue et al., 2013), natural language processing (Mooney, 1996), games (Law et al., 2014; Basu et al., 2021), data curation and transformation (Gulwani, 2011), explainable Artificial Intelligence (AI) (Zhang et al., 2024), and learning from trajectories (Inoue et al., 2014).

Some systems are capable of performing predicate invention, i.e., introducing new unnamed predicates into the learned program to adhere to a language bias or create simpler rules. Examples of such systems are POPPI (Cropper & Morel, 2021b), PHIL (Nguembang Fadja et al., 2021), Nesy-π (Sha et al., 2024), and many others. The presence of such predicates poses a big challenge for both programmers and final users, since it is difficult to understand their intended meaning, especially when the output theory is composed of a large number of rules. Additionally, manually renaming them is usually unfeasible, since it requires the availability of a domain expert. In fact, unnamed predicates not only complicate human understanding of the logic program but also hinder the debugging process for programmers.

In this paper, we thus propose to use Large Language Models (LLMs) to provide names to these predicates. In the last few years, LLMs (Alammar & Grootendorst, 2024), a type of generative AI, have proven to be highly performing in a wide array of tasks across many domains, including natural language processing, art, and data science, often producing human-like results (Brown et al., 2020; Kaddour et al., 2023). LLMs are based on deep learning architectures, typically using transformer models (Liu et al., 2024b; Vaswani et al., 2017), and are pre-trained on massive amounts of data, enabling them to process and generate natural language and other types of contents to perform a wide range of tasks, such as text, audio, and video generation, text summarization, code generation, chatbots and query answering. Therefore, the goal of this work is to exploit LLMs to generate semantically meaningful predicate names, which should be naturally understood by the user. To our knowledge, no systems currently perform automatic renaming of unnamed predicates.

The rest of this paper is structured as follows: Sect. 2 provides an overview of the relevant background; Sect. 3 presents a review of related work; Sect. 4 outlines the methodology used in the study; Sect. 5 describes the experimental setup and approach, and presents the results of the experiments; Sect. 6 discusses the findings and limitations of this work. Finally, Sect. 7 summarizes key findings and suggests directions for future research.

2 Preliminaries

2.1 Unnamed Predicates

ILP systems may sometimes introduce invented predicates that are not explicitly defined in the knowledge base or problem description. There are many reasons to invent new predicates, such as capturing intermediate relationships, simplifying complex rules, enhancing the program's modularity and flexibility, expressing a new concept, improving the accuracy while reducing the complexity of a hypothesis, or when the given vocabulary is insufficient to express the target theory. However, invented symbols have a placeholder name, such as "*inv*". The lack of a meaningful name makes it difficult to understand the intended use of the predicate and its reusability.

In ILP (Muggleton, 1991), Predicate Invention (PI) was first introduced in inverse resolution (Muggleton & Buntine, 1988), and early works have been summarized in Stahl (1993). PI refers to those techniques that enable systems to automatically discover new predicates, which are not part of the initial knowledge base (Stahl, 1995). One of the most straightforward examples of PI, shown in Example 1, is the invention of the predicate `parent` to learn the definition of `grandparent`, given only the predicates `mother` and `father`:

Example 1 Example of PI, where the invented predicate `inv` represents the parent relation.

```
grandparent(A,B) :- inv(A,C),inv(C,B).
inv(A,B) :- mother(A,B).
inv(A,B) :- father(A,B).
```

In this way, a more compact theory has been created. Despite gaining increasing attention in recent developments, PI is still not widely spread (Cropper & Dumančić, 2022). As a matter of fact, the invention of new, useful predicates is not a trivial task (Muggleton et al., 2012), as it is often challenging to determine when and if they are necessary, how many arguments they should have (i.e., their arity), and what their order and type should be (Stahl, 1995; Kramer, 1995).

METAGOL (Muggleton et al., 2015) is a metarule-based approach that seeks to overcome this limitation by using metarules to determine and restrict the syntax of rules, although they should be predefined by the user.

POPPER (Cropper & Morel, 2021a) is an ILP system based on the Learning From Failures (LFF) framework, which decomposes the ILP problem into three steps: generate, test, and constrain. In each cycle, POPPER generates a logic program based on current constraints, evaluates it against examples, and, if the hypothesis is incomplete or inconsistent, derives new constraints from the failure. This iterative refinement continues until a consistent and complete program is found. POPPI (Cropper & Morel, 2021b) is an ILP method that supports the creation of new predicates without needing human intervention to specify the arity or metarules. POPPI thus extends POPPER by enabling automatic PI, casting the ILP problem as an Answer Set Programming (ASP) task.

Sha et al. (2024) proposed NeSy-π, a Neuro-Symbolic (NeSy) approach for the classification of complex visual scenes, that processes visual inputs with deep neural networks and invents new concepts.

Nevertheless, the discovery of new predicates is not limited to PI.

In the context of Probabilistic Logic Programming (PLP) (Riguzzi, 2023), Hierarchical Probabilistic Logic Programs (HPLPs) (Fadja et al., 2017) are a restriction of Logic Programs with Annotated Disjunctions (LPADs) (Vennekens et al., 2004). In HPLPs clauses and predicates are hierarchically organized. Parameter learning for HIerarchical probabilistic Logic programs (PHIL) (Nguembang Fadja et al., 2021) is an algorithm for learning the parameters of HPLPs. Given a specific form of LPADs defining the target predicate *r* in terms of input predicates, for which the definition is given as input and is certain, and hidden predicates, defined by the rules of the program, PHIL learns the probabilities of atoms for the target predicate. Each clause in the program has a single atom in the head annotated with a probability. These programs can be divided into layers. Each layer defines a set of hidden unnamed predicates in terms of either the predicates from the layer below or the input predicates. In Example 2 from Nguembang Fadja et al. (2021) in the UWCSE domain (Kok & Domingos, 2005) we can see an example of such predicates. The first clause C_1 states that the probability of a student (A) being advised by a professor (B) is 0.3 if both A and B have worked on a project (C), and there is a relation r_1_1 between the student, the professor, and the project. The relation r_1_1 is defined in clause $C_1_1_1$, which states that r_1_1 holds with a probability of 0.2 if there is a publication P authored by both A and B and produced within project C. r_1_1 can then be renamed as "collaboretedOn". We can thus say that a student (A) is advised by a professor (B) if there is a project (C) on which both have worked, and A and B have collaborated on project C.

The first clause C_2 states that a student (A) is advised by a professor (B) with a probability of 0.6 if A is a teaching assistant (ta) for course C, which is taught by B.

Example 2 Unnamed predicate r_1_1 created by PHIL in the learning process.

```
C_1 = advised_by(A,B) : 0.3 :- student(A), professor(B),
   project(C,A), project(C,B), r_1_1(A,B,C).
C_2 = advised_by(A,B) : 0.6 :- student(A), professor(B), ta(C
   ,A), taughtby(C,B).
C_1_1_1 = r_1_1(A,B,C) : 0.2 :- publication(P,A,C),
   publication(P,B,C).
```

In the context of abductive logic programming (Kakas et al., 1992), the approach proposed by (Sato et al., 2018) aims to abduce relations by leveraging linear algebra in vector spaces. Hidden relationships between entities are learnt through matrix operations, making it useful for applications where explicit rules are hard to define. Focusing on Datalog programs with binary predicates, the authors address both non-recursive and recursive cases. For instance, in the non-recursive case, given two relations $r_1(X, Y)$ and $r_3(X, Z)$, represented as adjacency matrices R_1 and R_3 respectively, the system learns a new relation $r_2(Y, Z)$, represented by matrix R_2, that explains how r_1 and r_3 interact. This is done by solving the matrix equation $\mathrm{R}_3 = min_1(\mathrm{R}_1\mathrm{R}_2)$, where $min_1(x)$ is a nonlinear function, so that $r_3(X, Z) \Leftrightarrow \exists Y r_1(X, Y) \wedge r_2(Y, Z)$. Although r_1 and r_3 are known, r_2 is not, therefore it could be useful to find a meaningful name for easier reuse. For example, the authors have applied this method to a subset of the FB15k (Bordes et al., 2013) knowledge graph, involving person and film. By considering the relations *language(X,Z)* and *genre(X,Y)*, the following rule can be discovered:

$$language(X, Z) \Leftrightarrow genre(X, Y) \wedge r_{2_abd}(Y, Z).$$

Here, the abduced relation r_{2_abd}, which isn't originally in the graph, could be interpreted as *genre_lang*, implying that genre and language are somewhat related to each other. However, such an interpretation on the new predicate r_{2_abd} as well as the predicate name *genre_lang* have to be given by the user.

2.2 Overview of Large Language Models

Generative AI (Feuerriegel et al., 2024) is the subset of AI techniques that generate new content in the form of text, code, images, audio, and video.

One type of Generative AI are Foundation Models (FMs) (Bommasani et al., 2021), very large Machine Learning (ML) models trained on enormous quantities of generic and unlabeled data and thus capable of performing a wide range of tasks.

Among FMs, LLMs are based on the transformer architecture (Liu et al., 2024b; Vaswani et al., 2017; Alammar & Grootendorst, 2024). They are composed of multiple layers of neural networks, with an encoder and a decoder (Raffel et al., 2020; Yang et al., 2020) that process sequences of text, capturing the connections and the relationships between words. Thanks to the self-attention mechanism (Vaswani et al., 2017), the model can consider distant tokens as well. To represent words, LLMs use multi-dimensional vectors, called word embeddings, in a way that keeps related words close to each other in the embedding space.

These models often have billions of parameters and can be trained on vast amounts of data, enabling them to learn not only the grammar but also the semantics of a language. Given an input sequence, parameters are continuously adjusted during the training phase to maximize the likelihood of the next tokens.

LLMs have gained popularity in recent years (Brown et al., 2020) thanks to their ability to generate meaningful content from little input.

Various LLMs have been specifically trained and used for automatic code generation showing good performances (Brown et al., 2020; Kaddour et al., 2023), such as OpenAI's Codex (Chen et al., 2021), and its GitHub implementation, Copilot[1]. For example, Pearce et al. (2023) have tested Codex to determine if it can identify security flaws and fix them, obtaining positive results on hand-crafted datasets; however, real-world scenarios proved to be more challenging. Moradi Dakhel et al. (2023) demonstrated that Copilot can become a useful tool for software developers for implementing simple procedures, speeding up their work. Given their expertise, even if the generated code contains bugs, they should be able to identify and fix them.

2.2.1 Prompt Engineering

Prompting is crucial when working with LLMs to get the most relevant and useful answers. A carefully crafted prompt can significantly simplify the process by precisely defining the goal we aim to achieve. A basic prompt usually includes only the instructions for the task at hand, such as "summarize" or "classify". Other elements that can be included are: context; the audience; the role of the model; input data; output format; and examples of inputs and outputs (Alammar & Grootendorst, 2024). The tone of the response can also be controlled.

[1] https://copilot.github.com

Additionally, the order also matters: information in the initial or in the final part of the prompt is often given more weight than that in the middle. This behavior is referred to as primacy effect and recency effect, respectively (Alammar & Grootendorst, 2024; Liu et al. 2024a). These components are not always mandatory, but the more specific the prompt, the better the response will be. Giving an overly general prompt to the LLM may result in its inability to complete the given task.

One or more examples of the desired output can also be included in the prompt. This is referred to as in-context learning or few-shot learning (Min et al., 2022; Brown et al., 2020), in which a few examples of the task at hand and a test example are provided, as shown in Example 3 taken from Brown et al. (2020). Touvron et al. (2023), Alammar and Grootendorst (2024) proved that implementing these strategies leads to improved performances.

Example 3 Few-shot prompting example from Brown et al. (2020).

> ***Prompt:*** *A "whatpu" is a small, furry animal native to Tanzania. An example of a sentence that uses the word whatpu is:*
> *We were traveling in Africa and we saw these very cute whatpus.*
> *To do a "farduddle" means to jump up and down really fast. An example of a sentence that uses the word farduddle is:*
>
> ***Model answer:*** *One day when I was playing tag with my little sister, she got really excited and she started doing these crazy farduddles.*

Another prompting technique is Chain-of-Thought (CoT) Prompting, in which the output of one prompt is given as input of the next one, creating in this way a chain of prompts. CoT is at the base of LLMs reasoning (Wei et al., 2022; Kojima et al., 2022), as shown by OpenAI's reasoning models (OpenAI, 2025). Before generating an answer, these models learn to refine their thinking process, correct their mistakes, and explore different paths, building a chain of thought and leading to a higher-quality final result.

One of the main challenges in using LLMs is designing prompts that are both clear and effective in guiding the model toward the desired output. The quality of the prompt has a significant impact on the model's performance, and even small variations in wording or structure may lead to different results. Crafting an optimal prompt is therefore not straightforward, but is a gradual, iterative process that requires careful experimentation to determine what works best for each specific use case. Another major challenge is handling the inherent non-determinism of LLMs, as the same prompt can yield different outputs across multiple runs, making it difficult to ensure consistency and reproducibility.

3 Related Work

PI is considered a critical step, yet it is still not widely spread (Cropper & Dumančić, 2022). Choosing suitable names for newly created predicates is an even more neglected area in ILP and has not received much attention so far, with only a few attempts made in the past.

In Inoue et al. (2009) the authors adopted a meta-level abduction approach to identify previously unknown relations within a knowledge graph that represents a specific domain.

In this setting, new predicates are invented as existentially quantified variables, which act as placeholders for the missing or hidden knowledge. While the system is capable of structurally generating these new predicates, their semantic meaning (i.e., their names) is not determined automatically and must instead be manually defined or interpreted based on the domain context. This highlights a key limitation: the invented predicates are syntactically valid but lack meaningful, human-readable names without further domain-specific interpretation. In the rule abduction approach proposed in Kobayashi and Furukawa (2009), the authors built databases related to the cello playing domain to assist performers in assigning meaningful names to new abduced rules.

As the power of LLMs continues to grow, we could leverage their capabilities to enable them to reason about rules and generate meaningful names for previously unnamed predicates. While many have highlighted the impressive performances of LLMs with zero or few-shot prompting (Brown et al., 2020; Kojima et al., 2022; Wei et al., 2022), a rising number of recent works have explored their reasoning ability on more complex tasks (Li et al., 2024) involving common knowledge and planning (Valmeekam et al., 2022; Talmor et al., 2018), math (Cobbe et al., 2021; Kojima et al., 2022), and symbolic reasoning (Ishay et al., 2023; Kojima et al., 2022; Srivastava et al., 2022). In particular, LLMs are already widely employed in software engineering (Brown et al., 2020; Hou et al., 2024; Jiang et al., 2024; Kaddour et al., 2023), where they typically generate code with semantically meaningful variable names.

Li et al. (2024) evaluated state-of-the-art LLMs on an ILP benchmark and found their performance lacking when compared to a smaller neural program induction model. They evaluated LLMs on two benchmarks, one of which required the model to induce programs to deduce more complex family relationships from some basic ones. Although this setup is similar to ours, they focused on assessing the relational reasoning capabilities of LLMs using standard natural language and truth value prompting. In contrast, our goal was to determine if LLMs could identify names that are relevant to the context of the example. Han et al. (2024) used GPT-3.5 and GPT-4 for property induction, a traditional inductive reasoning task that involves generalizing a property from a few examples to new cases. In their experiments, participants–including both humans and language models–were presented with a set of objects sharing a property and asked to infer whether new objects shared the same property. Although GPT-3.5 had difficulty capturing various aspects of human behavior, GPT-4's performance was mostly close to that of humans.

Yang et al. (2024) investigated the ability of pretrained language models to infer natural language rules from natural language facts, obtaining promising results. Wang et al. (2023) proposed an approach to improve LLMs' inductive reasoning ability. They generate hypotheses in natural language, and then translate them into specific programs used for making predictions, breaking the task into two levels of abstraction and outperforming current baselines. In Tarau (2025) the authors explored techniques to extract and leverage the knowledge embedded in LLMs by converting it into different types of logic programs, aiming to improve reasoning capabilities and ensure that LLM outputs are consistent with their intended purposes. Additionally, Gandarela et al. (2024) aimed to determine whether LLMs can address ILP tasks, analyzing their capabilities and limitations on logic theory induction. Their results indicate that while larger LLMs can achieve competitive results compared to state-of-the-art ILP systems, long predicate relationship chains are still an obstacle for LLMs in inductive reasoning tasks. Cunnington et al. (2024) proposed NeSyGPT, a novel architecture that inte-

grates neural and symbolic reasoning, that fine-tunes a vision-language foundation model to extract symbolic features from raw data, and then learns an expressive answer set program using them. The results show that NeSyGPT outperforms various baselines in accuracy and can be used for more complex NeSy tasks. They thus demonstrated that LLMs can reduce the need for manual efforts in automating the interface between neural and symbolic components. Remaining in the NeSy field, De Giorgis et al. (2024) presented a novel approach to expand the knowledge base by integrating LLMs with structured semantic representations. It employs a state-of-the-art LLM to produce a natural language description of an input image, which is then transformed into an Abstract Meaning Representation (AMR) graph. This graph is further extended with logical design patterns and layered semantics derived from linguistic and factual knowledge bases. Kareem et al. (2025) proposed LLM2LAS, a system that learns commonsense knowledge in the form of ASP programs, from story-based question and answers in natural language, using only a few examples and combining LLMs' semantic parsing abilities with ILASP (Law et al., 2020), a tool for learning ASP programs. The authors of Ishay et al. (2023) used an LLM to generate ASP programs representing logic puzzles, given their description in natural language. Here, the authors state that the LLM successfully produced an ASP, despite some minor errors that were corrected manually, confirming the possibility of leveraging LLMs to support the creation of ASP programs. Although some findings suggest that LLMs still struggle with complex reasoning (Creswell et al., 2022; Li et al., 2024), this proves that the field yields some potential and is worth further investigation, especially considering the continuous improvements of LLMs.

4 Renaming Predicates Using LLMs

In this section, we outline our approach for assigning meaningful names to unnamed predicates in a set of logical rules.

4.1 Predicate Renaming Pipeline

After deciding the number n of LLMs to query for suggestions, the number k of suggestions they should provide, and the number j of LLM judges, the pipeline we propose consists of three main steps, as shown in Fig. 1:

1. Asking the n LLMs to find a meaningful name for each unnamed predicate, for k times to tackle possible hallucinations; for these preliminary experiments, we decided to repeat the question three times.
2. Asking the models to choose the most suitable name among their own suggested names.
3. Asking the j judges to score the different proposed names, obtaining a ranking.

4.2 Materials

To assess the ability of the LLMs to find correct names for unnamed predicates, we created different examples of logic theories, which represent the possible output of an ILP system in a declarative logic programming language. The selected case studies were intentionally designed to cover structurally heterogeneous predicate invention scenarios, including recursive defini-

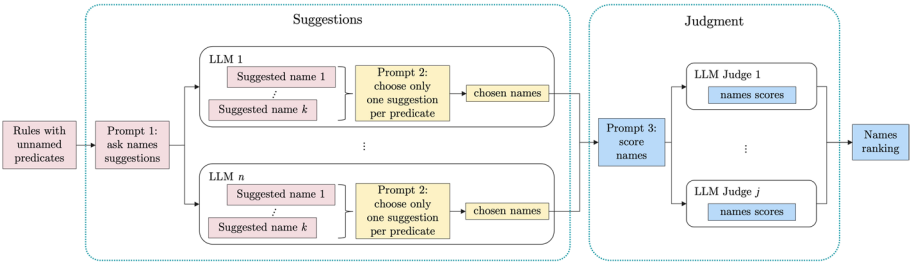


Fig. 1 Diagram illustrating key components and interactions of the approach we propose

tions, relational reasoning, arithmetic constraints, graph reachability, and real-world scientific theories. This allows us to evaluate whether LLMs can find semantically meaningful labels across different logical structures and domains, rather than on a single benchmark family.

Although constructs that are specific to Prolog were used for these experiments, this approach could be applied to scenarios where other logic programming languages are employed. For example, if rules are written in a purely declarative manner, those rules may be commonly used for both Prolog and ASP.

In each set of rules, we replaced the true names of some or all predicates with placeholders. We tested various formats (e.g., A, P, h0, and so on) for placeholders to determine whether the placeholder name influences the suggestions. Unnamed predicates can occur in the head and/or in the body of one or multiple rules. For all case studies presented in this paper, the true names of the unnamed predicates are included as comments (e.g., `# P = coauthors)` for the reader's reference. These comments were not included in the prompts provided to the LLMs.

4.2.1 Unnamed Predicates Appearing Only in the Head

The first and simplest case study, *coauthors*, consists of a single rule, as shown in Case Study 1. Here, `P` is only one unnamed predicate appearing in the head of a rule, which states that there are two researchers (A and B) and both authored a paper (C); P could thus be renamed as "coauthors".

Case Study 1 (coauthors) Rules of the coauthors case study.

```
P(A,B) :- author(A,C), author(B,C), researcher(A),
researcher(B), paper(C). # P = coauthors
```

4.2.2 Unnamed Predicates Appearing Both in the Head and in the Body

The second case study, *family*, is described in Case Study 2 and contains several rules representing various family relationships. Unnamed predicates appear both in the head and in the body of the rules, with `h2` being the only exception, as it appears only on the head on a single rule.

Case Study 2 (family) Rules of the family case study.

```
h0(X,Y) :- mother(X,Y). # h0 = parent
h0(X,Y) :- father(X,Y).
h1(X,Y) :- h0(X,Z), h0(Z,Y). # h1 = grandparent
ancestor(X,Y) :- h0(X,Y).
ancestor(X,Y) :- h0(X, Z), ancestor(Z,Y).
h2(X, Y, Z) :- ancestor(Z, X), ancestor(Z,Y). # h2 =
    common_ancestor
h3(X,Y) :- sister(X,Y). # h3 = siblings
h3(X,Y) :- sister(Y, X).
h3(X,Y) :- brother(X,Y).
h3(X,Y) :- brother(Y,X).
h4(X,Y) :- h0(PX, X), h0(PY,Y), h3(PX, PY), dif(PX,PY). # h4
    = cousins
```

The third case study, *math*, shown in Case Study 3, is a set of rules representing basic mathematical operations. Also in this case, unnamed predicates can appear both in the head and in the body of the rules.

Case Study 3 (math) Rules of the math case study.

```
A(X) :- integer(X), !. # isNumber
A(X) :- float(X).

P(X,Y) :- A(X), A(Y), X > Y. # greaterThan
Q(X,Y) :- A(X), A(Y), X >=Y. # greaterOrEqual
R(X,Y) :- A(X), A(Y), X < Y. # lessThan
S(X,Y) :- A(X), A(Y), X <=Y. # lessOrEqual
T(X,Y) :- A(X), A(Y), X = Y. # equalTo

B(X) :- A(X), 0 is mod(X,2). # isEven
C(X) :- A(X), not B(X).      # isOdd

D(X,Y) :- ( A(X), R(X,0) ->  # abs
    Y is -X
    ;
    Y is X
).

E(X,Y,Z) :- A(X), A(Y), Z is X + Y. # sum
F(X,Y,Z) :- A(X), A(Y), Z is X - Y. # difference
G(X,Y,Z) :- A(X), A(Y), Z is X * Y. # product
H(X,Y,Z) :- A(X), A(Y), dif(Y,0), Z is X / Y. # quotient

L(X,Y) :- A(X), A(Y), dif(X,0), 0 is mod(Y,X). # isDivisor

M(0,Y,Y):- A(Y), P(Y,0). # gcd
M(X,0,X):- A(X), P(X,0).
M(X,Y,Z):- A(X), A(Y), P(X,0), P(Y,0), R(X,Y), M(Y,X,Z).
M(X,Y,Z):- A(X), A(Y), P(X,0), P(Y,0), P(X,Y), T is X mod Y, M(Y,T,Z
    ).

N(X,0,0). # lcm
N(0,X,0).
N(X,Y,Z) :-
    A(X), A(Y),
    D(Y,U), D(X,K),
    M(X,Y,W),
    H(K,W,V),
    G(U,V,Z).
```

4.2.3 Unnamed Predicates Appearing Only in the Body

We also created simpler one-rule examples to investigate the ability of the LLMs to find a meaningful name for predicates appearing only in the body. These examples are taken from the previous ones and are shown in Case Study 4: the *grandparent* rule has predicate `h0` in the body, representing the `parent` relation; the *cousins* rule with predicate `h3` representing the `siblings` relation; and lastly, the *lcm* rule, representing the computation of the least common multiple using the greatest common divisor, uses predicates `G` and `H`, being `product` and `quotient`, respectively. In all these cases, the unnamed predicates appear only in the body of the rule.

Case Study 4 Rules of the *grandparent*, *cousins*, and *lcm* case studies.

```
grandparent(X,Y) :- h0(X,Z), h0(Z,Y). # h0 = parent
-----------------------------------------------------------
cousins(X,Y) :- parent(PX, X), parent(PY,Y), h3(PX, PY),
                dif(PX,PY). # h3 = siblings
-----------------------------------------------------------
least_common_multiple(X,Y,Z) :-
   G(X,Y,V), # G = product/multiply
   greatest_common_divisor(X,Y,W),
   H(V,W,Z). # H = quotient/divide
```

4.2.4 Unnamed Predicates in a Real-World Dataset

In addition, we also tried a real-world knowledge base taken from the Mutagenesis dataset (Srinivasan et al., 1996), a classic ILP benchmark dataset for Quantitative Structure-Activity Relationship (QSAR), that is, predicting the biological activity of chemicals from their physicochemical properties or molecular structure. The goal is to predict the mutagenicity of compounds given their chemical structure. We used the *ring_theory* provided with the dataset and removed the names of most of the head predicates. The set of rules is extensive, with 24 head predicates to be renamed, such as those in Case Study 5.

Case Study 5 (Mutagenesis) Examples of rules of the Mutagenesis ring theory. Here, the "@>" operator checks if the first term is greater than the second term according to Prolog's standard order of terms.

```
anthracene(Drug,[Ring1,Ring2,Ring3]) :-
   benzene(Drug,Ring1),
   benzene(Drug,Ring2),
   Ring1 @> Ring2,
   interjoin(Ring1,Ring2,Join1),
   benzene(Drug,Ring3),
   Ring1 @> Ring3,
   Ring2 @> Ring3,
   interjoin(Ring2,Ring3,Join2),
   \+ interjoin(Join1,Join2,_),
   \+ members_bonded(Drug,Join1,Join2).

no_of_nitros(Drug,No) :-
   setof(Nitro,nitro(Drug,Nitro),List),
   length(List,No).

ring6(Drug,[Atom1|List],[Atom1,Atom2,Atom4,Atom6,Atom5,Atom3
   ],
   [Type1,Type2,Type3,Type4,Type5,Type6]) :-
   bondd(Drug,Atom1,Atom2,Type1),
   memberchk(Atom2,[Atom1|List]),
   bondd(Drug,Atom1,Atom3,Type2),
   memberchk(Atom3,[Atom1|List]),
   Atom3 @> Atom2,
   bondd(Drug,Atom2,Atom4,Type3),
   Atom4 \== Atom1,
   memberchk(Atom4,[Atom1|List]),
   bondd(Drug,Atom3,Atom5,Type4),
   Atom5 \== Atom1,
   memberchk(Atom5,[Atom1|List]),
   bondd(Drug,Atom4,Atom6,Type5),
   Atom6 \== Atom2,
   memberchk(Atom6,[Atom1|List]),
   bondd(Drug,Atom5,Atom6,Type6),
   Atom6 \== Atom3.
```

Some of these rules are relatively short (1 to 3 body atoms), while others are much more complex, with more than 5 body atoms.

4.2.5 Unnamed Predicates in an Actual POPPER Output

To test our approach on an actual ILP theory, we designed a new case study and used POPPER to learn a recursive definition of a reachability predicate over a graph defined by bidirectional (road/2) and unidirectional (one_way/2) connections. The goal was to induce a definition of reachable/2 using predicate invention, such that POPPER introduces an intermediate invented predicate (e.g., inv1/2) to generalize a single-step path relation. The resulting program is presented in Case Study 6. Given that this case study was specifically designed by the authors for this work, it is not present in the existing literature to the best of our knowledge. As a result, it is unlikely that the LLMs were exposed to it during training.

Case Study 6 (reachability) Rules of the reachability case study.

```
inv1(V0,V1):- road(V0,V1).
inv1(V0,V1):- one_way(V0,V1).
reachable(V0,V1):- inv1(V0,V1).
reachable(V0,V1):- reachable(V2,V1),inv1(V0,V2).
```

4.3 Prompts

We use zero-shot prompting for all the examples. Figure 2 shows *Prompt 1*, used to ask for name suggestions. We insert the corresponding rules with placeholders where `[rules]` is present. The prompt is built in sections, separated by `### section ###`: in the beginning, we provide the context; then, we give the instruction; lastly, we provide the desired output format. The prompts used for the examples differ only in the predicates listed in the instruction and output format. Since we don't want our method to be dependent on the chosen language, we instruct the LLMs to behave as a "specialist of logic programming".

Prompt 2, shown in Fig. 3, is used when asking the models to choose between their own suggestions. For each model, `[suggested names]` is replaced with the list of its own suggestions. Since models can suggest names with semantically equivalent meanings but written in different formats, such as using underscores, hyphens, or capitalized letters, we ensure that they are standardized: we remove any underscores or hyphens from the name and convert them to camel case, leaving the first letter in lowercase. After this, we remove duplicates from the list.

Finally, *Prompt 3* in Fig. 4 was used for the judgment, where we asked the model to score the suggestions for each predicate. We also provide the scoring rules in the `INSTRUCTIONS` section.

4.4 LLMs Used in This Study

For this study, we selected 7 different LLMs: OpenAI's ChatGPT based on GPT-4o (ChatGPT-4o) (OpenAI, 2024a, 2024b) and ChatGPT based on o3mini (ChatGPT-o3mini) (OpenAI, 2025); Meta's Llama 3.2 3B Instruct (Llama) (Meta, 2024a, 2024b), Google DeepMind's Gemini 1.5 Flash (Gemini) (Deepmind, 2024); FalconMamba 7B Instruct (FalconMamba) (Zuo et al., 2024) and Falcon3 7B Instruct (Falcon3) (Falcon-LLM Team, 2024) of the Technology Innovation Institute (TII); and Cohere's Command R+ 08 2024 (Command R+) (Cohere For AI, 2024).

ChatGPT is a powerful chatbot running on GPT-4o (with approximately 200 billion parameters), capable of processing multi-modal inputs and answering within a few milliseconds, mirroring the behavior of a human. Generative pre-trained Transformer (GPT) models, based on the transformers technology, are robust and versatile and are nowadays widely employed for routine tasks such as text summarization or translation. Recently, given the significant advances, the adoption of ChatGPT and LLMs in general is also spreading in more critical applications (Koubaa et al., 2023), such as generating simplified clinical reports (Jeblick et al., 2024) or supporting decision-making process (Rao et al., 2023), with high accuracy and correctness. In addition to GPT models, OpenAI provides also reasoning models, trained with reinforcement learning. Reasoning models, like o1 and the lat-

est o3mini that has 3B parameters, use a chain-of-thought approach, where intermediate reasoning steps are generated before producing the final answer. Although this increases response time, reasoning models yield higher-quality outputs and are particularly effective in tasks involving science, mathematics, and coding (OpenAI, 2025).

Google DeepMind's Gemini 1.5 Flash is a fast and versatile model that can take text, images, audio, and video as input, and produces a text output. Gemini 1.5 Flash has been adopted in an AI-based application to help users monitor and reduce their carbon emissions (Dash et al., 2024). It also obtained good results in answering multiple-choice questions regarding pediatric nephrology (Mondillo et al., 2024).

Llama 3.2 3B Instruct is a lightweight (3.21B parameters) text-only model of the Llama 3.2 family, which is a collection of multilingual, pre-trained, and instruction-tuned LLMs.

FalconMamba-7B-Instruction is a version of the base model FalconMamba 7B, fine-tuned on instruction data. The model is based on Mamba (Gu & Dao, 2023), an attention-free architecture, and has 7.27B parameters. The Falcon3 family is a collection of pre-trained and instruction-tuned, decoder-only LLMs of various sizes. Falcon3-7B-Instruct, specifically, has been pre-trained on curated high-quality and multilingual data from various sources (web, code, STEM), and according to TII it achieves state-of-art-results on different tasks, including natural language processing, common-sense reasoning, coding, and reasoning. This is due to the improvements made in the development of the base models, such as depth upscaling and knowledge distillation (Hinton, 2015).

Cohere's Command R+ 08–2024 is a multilingual model optimized for various applications, such as text summarization, reasoning, and structured data analysis, with 104 billion parameters, and trained on an extensive corpus of data in different languages. Command R+ 08–2024 has shown good performances for most of the writing tasks in Arabic tested by Magdy et al. (2024) and evaluated on *Gazelle*, the dataset they proposed, delivering clear and coherent responses.

```
### Context ###
You are a software engineer specialized in logic programming.

### Instruction ###
Given these first-order logic rules:
[rules]

Assign general and meaningful names to predicates h0, h1, h2,
    h3, and h4.
Do NOT change the body and the variables of the rules.
Give only ONE suggestion for each predicate.

### Output format ###
h0:
h1:
h2:
h3:
h4:
```

Fig. 2 *Prompt 1* for asking for one suggestion for each unnamed predicate. This prompt was used for Case Study 2. `[rules]` is replaced with the corresponding rules

```
### CONTEXT ###
You are a software engineer specialized in logic programming.
Given these first-order logic rules:
[rules]

The following names have been proposed for predicates h0, h1,
    h2, h3, and h4.
h0: [suggested names]
h1: [suggested names]
h2: [suggested names]
h3: [suggested names]
h4: [suggested names]

### INSTRUCTION ###
Choose the most suitable name for predicates h0, h1, h2, h3,
   and h4, among the proposed ones.

### OUTPUT FORMAT ###
h0: CHOSEN_NAME
h1: CHOSEN_NAME
h2: CHOSEN_NAME
h3: CHOSEN_NAME
h4: CHOSEN_NAME
```

Fig. 3 *Prompt 2* used for asking the models to choose one among their own suggestions. This prompt was used for Case Study 2. `[rules]` is replaced with the corresponding rules and the suggestions of the model being used replace `[suggested names]`

Unless otherwise specified, the models were used using their default inference parameters. For FalconMamba and Falcon3, we used `max_new_tokens=2048`, while for Llama we used `max_new_tokens=1024`. Due to the evolving nature of commercial LLM services, model behavior may vary over time even for the same reported model version. For this reason, we also report here the time period during which the experiments were conducted. The experiments involving few-shot prompting, the *reachability* case study, and the human judgment evaluation were conducted between July and August 2025, while all other experiments took place between January and February 2025. Since ChatGPT-4o and ChatGPT-o3mini were no longer available at the time of the *reachability* case study experiment, we used ChatGPT-5, the only freely accessible version at that time. This reflects a broader reproducibility challenge in LLM-based experimentation, where commercial models and APIs may evolve over time or become unavailable, potentially affecting model behavior.

4.5 Evaluation: LLM-as-a-Judge and Human Judgment

We use LLMs also for the automated evaluation of the answers, adopting the LLM-as-a-Judge strategy (Zheng et al., 2023; Alammar & Grootendorst, 2024).

We selected four models, ChatGPT-4o, ChatGPT-o3mini, Gemini, and Command R+, to act as judges. Each judge assigns a score to every proposed name for each predicate: 1 for a correct and precise name, 0.5 for a name that is imprecise or too general but still correct,

```
### CONTEXT ###
You are a software engineer specialized in logic programming.
The following logic rules have unnamed predicates in the head
    and/or in the body:
[rules]

These are the suggested names for the unnamed predicates h0,
   h1, h2, h3, and h4:
h0: [suggested names]
h1: [suggested names]
h2: [suggested names]
h3: [suggested names]
h4: [suggested names]

### INSTRUCTIONS ###
Score the proposed names for each predicate following the
   rules below:
- Assign 1 to correct and precise answers.
- Assign 0.5 to answers that are too generic, but still
   correct.
- Assign 0 to incomplete, imprecise or incorrect answers.

### OUTPUT FORMAT ###
For each predicate, list all the suggestions and their score:
h0:
h1:
h2:
h3:
h4:
```

Fig. 4 *Prompt 3* used for asking the models acting as judges to score the suggestions for each predicate. This prompt was used for Case Study 2. `[rules]` is replaced with the corresponding rules

and 0 for a wrong name. The scores are then summed to determine the final results, and the name with the highest score is selected.

Finally, to assess the ability of LLMs in both renaming and judgment, we conducted a complementary experiment involving 14 human judges on all the case studies except Mutagenesis and *reachability*. The *reachability* case study was not included in the human judgment experiment, as it was introduced after the experiment had already been conducted, while Mutagenesis requires a field expert to provide reliable feedback. The participants were students enrolled in the Master's Degree in Artificial Intelligence, Data Science and Big Data at the University of Ferrara, who had previously attended courses involving logic programming and Prolog. This background allowed them to understand the semantics of the logic rules and evaluate the appropriateness of the proposed predicate names. Each participant independently evaluated the generated predicate names using the same scoring methodology adopted in the LLM-as-a-Judge evaluation. The evaluation focused on controlled logic theories whose intended semantics could be reasonably interpreted from the rules themselves, without requiring advanced domain-specific expertise.

5 Experiments and Results

In this section, we illustrate the results of our experiments.

For this study, the desktop app was used to execute both ChatGPT running on GPT-4o and ChatGPT running on GPT-o3mini. We accessed Gemini and Command R+ via their own API. For Llama, FalconMamba, and Falcon3, experiments were performed locally via the Hugging Face Transformers framework (Wolf et al., 2020) on a cluster consisting of two machines with two AMD EPYC 9654 96-cores and four machines with two AMD EPYC 9454 48-cores. The memory limit was set to 100 GB, and the time limit was set to 17 hours and 30 minutes.

5.1 Prompting Strategy

We carried out several initial prompt design attempts to observe how LLMs responded under varying instructions. For instance, after noticing that the models often provided multiple equivalent suggestions, we decided to limit them to just one. We also added a constraint to prevent models from altering the structure of the rules, after observing such behaviors with some LLMs. Each of these attempts was executed independently, ensuring that no contextual information was shared between runs. Importantly, the models did not retain any memory of previous executions; in particular, ChatGPT was used with the memory feature explicitly disabled to prevent any persistence of prior interactions. After analyzing the models' responses and identifying recurring issues or undesired behaviors, we refined our instructions iteratively. The final version of the prompt, shown in Fig. 2, was then used to rerun all experiments from scratch. Additionally, we observed that models sometimes gave the correct response on the first try, while other times they required several attempts. To address this, and given the non-deterministic nature of the answers, we decided to repeat the question a certain number of times to address this issue. In these preliminary experiments, we decided to use three repetitions. All the models handled the task appropriately, so zero-shot prompting was sufficient in this case. Most of them followed the instructions accurately, with only a few exceptions. For example, Llama usually wrote a sentence (explaining the reasons for its choices) instead of adopting the requested format. In such cases, however, the name can be extracted from the answer using basic string manipulation. After getting three suggestions out of each model, we asked them to choose only one name among their own ones.

Tables 1 and 2 show the final choices made by each model for each predicate of each case study; we also report for each LLM how many of the suggested names could be considered correct. Table 3 provides a summary of the number of correct name suggestions per predicate. Each model receives a score of 1 if its final suggestion matches the expected name, and 0 otherwise. In the final column, we indicate how many LLMs correctly named each predicate. This overview allows us to easily compare the performance of the models on a per-predicate basis and identify which predicates were more challenging.

5.1.1 Results for Unnamed Predicates Appearing Only in the Head

The *coauthors* case study, which was the simplest, presented no difficulties for the models. In fact, all of them suggested a correct name, although sometimes too verbose.

5.1.2 Results for Unnamed Predicates Appearing Both in the Head and in the Body

Regarding the *family* case study, the GPT models and Command R+ provided the correct name of each predicate, and Gemini failed to identify only h4. Llama was completely unable to find meaningful names for most of the predicates, with only two correct suggestions. Furthermore, one out of the three times it rewrote most of the rules, making its answer completely wrong. FalconMamba and Falcon3 also struggled with the *family* case study, as none of the answers were correct. In particular, Falcon3 provided the answer for only one predicate, likely being misled by the word "ONE" in the prompt. For this reason, its answers were not considered.

For the *math* case study, the GPT models correctly identified almost all the names. Gemini followed closely, with only two mistakes. Falcon3 and Command R+ also identified most of the correct names, while Llama just over half of them. On the other hand, FalconMamba simply rewrote the rules with different capital letters all three times, and therefore its answers were ignored. Many models couldn't find the correct name for predicates L (`isDivisor`, suggested only by Command R+), M (`gcd`), and N (`lcm`); these last two were correctly identified only by ChatGPT-4o and ChatGPT-o3mini.

5.1.3 Results for Unnamed Predicates Appearing Only in the Body

In the *lcm* case study, most models struggled to identify the correct names for predicates G and H, namely `product` and `quotient` (or equivalent terms). Only ChatGPT-4o and ChatGPT-o3mini provided correct suggestions.

Finding `siblings` for predicate h3 in the *cousins* case study was also challenging for some models, even if they suggested it in the *family* case study. Only the GPT models, Gemini and LLama provided the correct name.

For the *grandparent* case study, all the models correctly suggested `parent` for predicate h0, except for FalconMamba.

5.1.4 Results for the Mutagenesis Case Study

In Table 4 we report the results obtained on the Mutagenesis ring theory, using ChatGPT-4o and ChatGPT-o3mini. Compared to the previous case studies, Mutagenesis represents a more challenging setting, due to the larger number of predicates and the highly domain-specific nature of the underlying chemical knowledge. Some of the predicates appear to be similar to the original ones, but overall it is difficult to determine whether the suggested names are meaningful. In such a case, a domain expert should make the final decision. To improve the quality of the output, the models could be fine-tuned on the specific domain; alternatively, context (e.g., in the form of scientific papers) could be provided.

It is worth mentioning another attempt we made with Mutagenesis. For this initial experiment, we used a less constrained version of *Prompt 1*, where we simply asked for suggestions for all the predicates, without specifying the context or limiting the model to provide only one suggestion per predicate. The response of Command R+ was particularly interesting. The model not only explained the predicate to rename but in some cases it also invented a helper predicate for it. An example of such behavior is shown in Fig. 5: the model created the helper predicate `check_atom_count_recursive` for predicate HP19, which

Table 1 Final names suggestions for each predicate of all the examples, made by ChatGPT-4o, ChatGPT-o3mini, and Command R+

Predicate	ChatGPT-4o	ChatGPT-o3mini	Command R+
Coauthors[1]			
P	coauthors_with_common_paper	coauthors	co_authored_paper
correct	**1/1**	**1/1**	**1/1**
Family			
h0	parent	parent	parent
h1	grandparent	grandparent	grandparent
h2	common_ancestor	common_ancestor	common_ancestor
h3	sibling	sibling	sibling
h4	cousin	cousin	cousins
correct	**5/5**	**5/5**	**5/5**
Grandparent			
h0	parent	parent	parent
correct	**1/1**	**1/1**	**1/1**
Cousins			
h3	siblings	siblings	different_parents
correct	**1/1**	**1/1**	**0/1**
Math			
A	is_number	numeric_value/1	is_integer
B	is_even	even/1	is_even
C	is_odd	odd/1	is_odd
D	absolute_value	abs_value/2	negate_if_negative
E	add	sum/3	sum
F	subtract	difference/3	difference
G	multiply	product/3	product
H	divide	quotient/3	division
L	is_divisible	divides/2	is_divisor
M	gcd	gcd/3	modular_arithmetic
N	lcm	lcm/3	nested_calculation
P	greater_than	greater_than/2	greater_than
Q	greater_than_or_equal	greater_or_equal/2	greater_equal
R	less_than	less_than/2	less_than
S	less_than_or_equal	less_or_equal/2	less_equal
T	equal	equal_to/2	equal
correct	**15/16**	**15/16**	**12/16**
lcm			
G	compute_product	multiply	is_multiple_of
H	divide_values	divide	lcm_calculation
correct	**2/2**	**2/2**	**0/2**
reachability			
inv1	direct_connection[2]	*NA*	is_connected
correct	**1/1**	–	**1/1**

[1]In this case, several names were suggested and even if excessively verbose, they can be considered correct.

[2]Since ChatGPT-4o and ChatGPT-o3mini were no longer available at the time of the experiment, ChatGPT-5 was used instead

Table 2 Final names suggestions for each predicate of all the examples, made by FalconMamba, Falcon3, Gemini, Llama

Predicate	FalconMamba	Falcon3	Gemini	Llama
Coauthors[1]				
P	Co-authoredResearchPaper	CoAuthorResearchers	coauthored_ research_paper	authoredTogether
correct	**1/1**	**1/1**	**1/1**	**1/1**
Family				
h0	ancestor	-	parent	parent
h1	ancestor	-	grandparent	ancestor
h2	ancestor	-	commonAn-cestor	great_ancestor
h3	sister	-	sibling	sibling
h4	h3	-	halfSibling	full_sibling
correct	**0/5**	**0/5**	**4/5**	**2/5**
Grandpar-ent				
h0	rename_h0	parent	parent	parent
correct	**0/1**	**1/1**	**1/1**	**1/1**
Cousins				
h3	is_third_degree_relative	parent	sibling	sibling
correct	**0/1**	**0/1**	**1/1**	**1/1**
Math				
A	–	IsNumber	isNumber	is_integer
B	–	IsEven	isEven	is_odd
C	–	IsOdd	isOdd	is_even
D	–	IsPositiveDifference	absValue	next
E	–	Sum	sum	add
F	–	Difference	difference	subtract
G	–	Product	product	multiply
H	–	Division	quotient	divide
L	–	DivisibleBy	isDivisibleBy	modulo
M	–	CustomOperation	gcd	accumulate
N	–	ComplexOperation	complexCal-culation	accumulate_se-quence
P	–	GreaterThan	greaterThan	greater_than
Q	–	GreaterThanOrEqual	greaterTha-nOrEqual	greater_than or_equal_to
R	–	LessThan	lessThan	less_than
S	–	LessThanOrEqual	lessTha-nOrEqual	less_than_ or_equal_to
T	–	Equal	isEqual	equal_to
correct	**0/16**	**12/16**	**14/16**	**9/16**
lcm				
G	lowest_common_multiple	multiply_and_add	find_least_ common_ multiple_inter-mediate	greatest_ common_divisor
H	highest_common_divisor	divide_and_subtract	compute_lcm_	least_

Table 2 (continued)

Predicate	FalconMamba	Falcon3	Gemini	Llama
			from_gcd	common_multiple
correct	**0/2**	**0/2**	**0/2**	**0/2**
reachability				
inv1	can reach	–	directly_connected	is_connected
correct	**0/1**	**0/1**	**1/1**	**1/1**

[1]In this case, several names were suggested and even if excessively verbose, they can be considered correct

was renamed as `check_atom_count_recursive`. Regardless of the correctness of the suggested name and despite altering the structure of the rule, although preliminary, this behavior suggests that LLMs may support aspects of predicate invention.

5.1.5 Results for the Reachability Case Study

The *reachability* case study was generally unchallenging for the majority of models evaluated. Suggestions produced by GPT-5 (direct_connection), Command R+ (is_connected), Gemini (directly_connected), and LLama (is_connected) are all plausible and appropriate, as they generalize a single-step path relation. In contrast, both FalconMamba and Falcon3 exhibited notable difficulties. FalconMamba generated an output (can reach) that was syntactically invalid as a predicate, due to the inclusion of a space in the identifier, while Falcon 3 failed to produce any answer at all.

5.2 Results for the Few-Shot Prompting Experiment

We additionally conducted a few-shot prompting experiment with the LLMs that demonstrated lower performance, namely FalconMamba, Falcon3, and Llama, with the goal of improving their answers. We focused on the *math* case study, which was selected due to its potential to highlight the benefits of few-shot prompting. In this experiment, suggestions were requested for one predicate at a time, while also providing the model with the correct names of previously addressed predicates. Figure 6 shows the prompts used in the few-shot prompting experiments with answers from Falcon3. The initial prompt, *Prompt 0*, requests the name of only the first predicate (A). Then, its answers is used to replace the corresponding unnamed predicate in the following prompt, *Prompt 1*, while requesting a suggestion for the second predicate (P). This continues until the last prompt, in which all the unnamed predicates of the math case study have been renamed, except of the last one. This incremental approach aimed to support the model by narrowing its focus and reducing the complexity of the task, in contrast to requiring it to generate names for all predicates simultaneously. However, the outcomes did not meet expectations. None of the models showed improved performance in terms of correctly identifying a greater number of names; in some instances, performance even declined.

Table 5 shows the results of this experiment. With few-shot prompting Falcon3 correctly identified 10 out of 16 predicate names, a decrease from the 12 out of 16 correctly identified with zero-shot prompting. Similarly, Llama achieved 8 correct predictions with few-shot prompting, compared to 9 without. A substantial improvement was observed with FalconMamba. with zero-shot prompting, the model failed to generate a meaningful output,

Table 3 Summary of correct name suggestions for each predicate

Predicate	ChatGPT-4o	ChatGPT-o3mini	Command R+	FalconMamba	Falcon3	Gemini	Llama	Total
Coauthors								
P	1	1	1	1	1	1	1	7/7
Family								
h0	1	1	1	0	0	1	1	5/7
h1	1	1	1	0	0	1	0	4/7
h2	1	1	1	0	0	1	0	4/7
h3	1	1	1	0	0	1	1	5/7
h4	1	1	1	0	0	0	0	3/7
Grandparent								
h0	1	1	1	0	1	1	1	6/7
Cousins								
h3	1	1	0	0	0	1	1	4/7
Math								
A	1	1	0	0	1	1	0	4/7
B	1	1	1	0	1	1	0	5/7
C	1	1	1	0	1	1	0	5/7
D	1	1	0	0	0	1	0	3/7
E	1	1	1	0	1	1	1	6/7
F	1	1	1	0	1	1	1	6/7
G	1	1	1	0	1	1	1	6/7
H	1	1	1	0	1	1	1	6/7
L	0	0	1	0	0	0	0	1/7
M	1	1	0	0	0	1	0	3/7
N	1	1	0	0	0	0	0	2/7
P	1	1	1	0	1	1	1	6/7
Q	1	1	1	0	1	1	1	6/7
R	1	1	1	0	1	1	1	6/7
S	1	1	1	0	1	1	1	6/7
T	1	1	1	0	1	1	1	6/7
lcm								
G	1	1	0	0	0	0	0	2/7
H	1	1	0	0	0	0	0	2/7
reachability								
inv1	1[1]	-	1	0	0	1	1	4/6

[1]Since ChatGPT-4o and ChatGPT-o3mini were no longer available at the time of the experiment, ChatGPT-5 was used instead

For each LLM, a value of 1 indicates that its final suggestion was correct, while 0 indicates an incorrect suggestion. The *Total* column shows the number of LLMs that correctly named each predicate

simply repeating the predicates. In contrast, when few-shot prompting was applied, the model provided a suggestion for all the predicates, finding 11 correct answers out of 16. These findings, particularly in the case of FalconMamba, indicate that breaking down the prompt and using few-shot prompting can significantly enhance performance in models that otherwise struggle with zero-shot prompting.

Table 4 Results obtained on the Mutagenesis ring theory, with ChatGPT-4o and ChatGPT-o3mini

Predicate	ChatGPT-4o	ChatGPT-o3mini
HP1 (anthracene)	fused_aromatic_ tricycles_non_overlapping	fused_aromatic_tricyclic_ system_no_bridge
HP2 (phenanthrene)	fused_aromatic_ tricycles_adjacent	fused_aromatic_tricyclic_ system_with_bridge
HP3 (ball3)	fused_aromatic_ tricycles_interconnected	fused_aromatic_tricyclic_ system_common_junction
HP4 (members_bonded)	shared_aromatic_bond	ring_junction_bridge
HP5 (no_of_benzenes)	aromatic_hexagons_count	count_aromatic_ six_membered_rings
HP6 (no_of_carbon_5_ aromatic_rings)	aromatic_pentagons_count	count_aromatic_ five_membered_rings
HP7 (no_of_carbon_6_rings)	non_benzenoid_ hexagons_count	count_nonaromatic_ six_membered_rings
HP8 (no_of_carbon_5_rings)	non_benzenoid_ pentagons_count	count_nonaromatic_ five_membered_rings
HP9 (no_of_hetero_ aromatic_6_rings)	heterocyclic_hexagons _count	count_aromatic_heterocyclic _six_membered_rings
HP10 (no_of_hetero_ aromatic_5_rings)	heterocyclic_pentagons_count _count	count_aromatic_heterocyclic five_membered_rings
HP11 (ring_size_6)	hexagonal_ring_structure	generic_six_ membered_ring
HP12 (ring_size_5)	pentagonal_ring_structure	generic_five_membered_ring
HP13 (benzene)	benzene_ring	aromatic_six_membered_ring
HP14 (carbon_5_ aromatic_ring)	cyclopentadiene_ring	aromatic_five_ membered_ring
HP15 (carbon_6_ring)	non_benzenoid_hexagon	nonaromatic_six_ membered_ring
HP16 (carbon_5_ring)	non_benzenoid_pentagon	nonaromatic_ five_membered_ring
HP17 (hetero_ aromatic_6_ring)	heterocyclic_hexagon	aromatic_heterocyclic_ six_membered_ring
HP18 (hetero_ aromatic_5_ring)	heterocyclic_pentagon	aromatic_heterocyclic_ five_membered_ring
HP19 (atoms)	atom_sequence	ring_atom_chain
HP20 (ring6)	aromatic_hexagonal_ bond_pattern	six_membered_ ring_structure
HP21 (ring5)	aromatic_pentagonal_ bond_pattern	five_membered_ ring_structure
HP22 (no_of_nitros)	nitro_group_count	count_nitro_groups
HP23 (nitro)	nitro_group_structure	nitro_group_structure
HP24 (methyl)	methyl_group_structure	methyl_group_structure

5.3 LLMs Judgment Results

In the considered case studies, reference predicate names are available because the theories were either manually constructed or derived from known target predicates. However, the goal of predicate renaming is not strict lexical matching with a single "correct" label,

```
% Rules for specific chemical patterns

% HP19: Checks if a drug contains a specific atom type with a
    certain number of occurrences.
% Args: Drug, Number of atoms, List of atoms, List of atom
   types
check_atom_count(Drug, N, Atoms, Types) :-
   atm(Drug, Atom, Type, _, _),
   Types = [Type | _],
   check_atom_count_recursive(Drug, N, [Atom], Atoms, Types,
      1).

check_atom_count_recursive(_, 1, Atoms, Atoms, _, _) :- !. %
   Base case: N is 1
check_atom_count_recursive(Drug, N, [Atom | _], Atoms, [Type
   | Types], N) :-
   N > 1,
   N1 is N - 1,
   atm(Drug, Atom, Type, _, _),
   check_atom_count_recursive(Drug, N1, _, Atoms, Types, _).
```

Fig. 5 Result obtained with Command R+ on Case Study 5 with a less constrained version of *Prompt 1*

since multiple names may adequately express the intended meaning of a predicate. For this reason, the evaluation focuses on semantic interpretability and meaning alignment, i.e., whether the generated name appropriately conveys the semantics of the predicate within the context of the logic theory.

Regarding the judgment on the *coauthors* case study, shown in Table 6, all the suggestions are plausible. LLM judges seem to prefer a longer and more detailed name, assigning the lowest score to the simplest one, i.e. `coauthors`.

In Table 7 we report the results of the judgment on the *family* case study. The correct name was chosen for each predicate. The same holds for the *grandparent* and *cousins* case studies, both shown in Table 6.

Table 8 shows the results of the judgment on the *math* case study. Here, the correct name was chosen for all the predicates, except for predicate L. While the correct name is `divisor`, it was mostly interpreted as "divisible", which is actually the opposite. Regarding predicates M and N, all the judges recognized the true names, `gcd` and `lcm` respectively, even if they didn't suggest them initially. The strategy of asking multiple models for name suggestions turned out to be effective, as it allowed the judges to identify the correct name when one model suggested it, even when the others did not. The same cannot be said for the *lcm* case study, as can be seen in Table 6. For predicate G, `findLeastCommonMultipleIntermediate` was chosen as the most fitting name, instead of `computeProduct` or `multiply`, which would have been equally correct. Similarly, for predicate H, `computeLcmFromGcd` was selected as the most appropriate name, rather than `divide` or `divideValues`.

Table 9 presents the judgment results for the reachability case study. It is important to note that ChatGPT-5 was used as a substitute for ChatGPT-4o and ChatGPT-o3mini, as these models were no longer available at the time the experiment was conducted. Three out

Prompt 0:

```
### CONTEXT ###
You are a software engineer specialized in logic programming.

### INSTRUCTION ###
Given the logic rules below, find a meaningful name for A.
Give only ONE suggestion for the predicate name.
Write only the predicate name after the predicate placeholder,
    without any additional text.
Do not rewrite the predicate body.

A(X) :- integer(X), !.
A(X) :- float(X).

Strictly follow this output format:
A: [your_suggestion]
```

Response:

```
A: is_number
```

Prompt 1:

```
### CONTEXT ###
You are a software engineer specialized in logic programming.

### INSTRUCTION ###
Given the logic rules below, find a meaningful name for P.
Give only ONE suggestion for the predicate name.
Write only the predicate name after the predicate placeholder,
    without any additional text.
Do not rewrite the predicate body.

is_number(X) :- integer(X), !.
is_number(X) :- float(X).

P(X,Y) :- is_number(X), is_number(Y), X > Y.

Strictly follow this output format:
P: [your_suggestion]
```

Response:

```
P: is_greater_number
```

Fig. 6 Prompts used in the few-shot prompting experiments with Falcon3 answers

of the four alternatives were selected as the most suitable name, each receiving the same score. This outcome is acceptable, as all three options are considered plausible. Notably, however, only ChatGPT-5 assigned a score of 0 to the proposed name `can reach`, which is syntactically incorrect. In contrast, both Gemini and Command R+ assigned it a score of 0.5, indicating a limited ability to recognize the syntactic invalidity of the suggestion.

5.4 Human Judgment Results

Although all the proposed alternatives in the *coauthors* case study are plausible, human judges did not select the most straightforward option for predicate P, that is, `coauthors`. This behavior aligns with that observed with the LLMs. They also correctly identified the appropriate names for predicates h0 and h3 in the *grandparent* and *cousins* case studies, respectively. In the *lcm* case study, human judges successfully selected the correct names

Table 5 FalconMamba, Falcon3 and Llama suggestions for the predicates of the *math* case study, in the few-shot prompting experiment

Predicate	FalconMamba	Falcon3	Llama
A : number	integer_or_float	is_number	has_integer_value
B : even	even_integer	is_even_number	parity
C : odd	odd_integer	is_odd_number	odd
D : abs	abs	negate_if_negative	divide_by_zero
E : addition	sum	add_numbers	sum
F : substraction	subtract	subtract_numbers	difference
G : multiplication	multiplication	multiply_numbers	multiply
H : division	division_by_non_zero_integer	divide_numbers	divide
L : divisor	even_integer	divisible_by	is_divisible
M : gdc	even_integer	swap_and_mod_numbers	min
N : lcm	even_integer_product	calculate_result	number
P : greater_than	greater_than_or_equal	is_greater_number	greater_integer
Q : greater_equal_than	greater_than_or_equal	is_number_or_equal	greater_than
R : less_than	less_than	is_less_number	less_than
S : less_equal_than	less_than	is_number_or_less	between
T : equal	equal_to	is_equal_number	equal
correct	**11/16**	**10/16**	**8/16**

for predicates G and H, whereas the LLMs judges failed to do so. Averaged scores assigned by human judges for these case studies are summarized in Table 10.

Table 11 presents the results of the human evaluation for the proposed names in the *family* case study. For each predicate, the correct name received the highest overall score. However, it is noteworthy that several human judges assigned non-zero scores to incorrect names. For instance, in the case of predicate h0, 4 out of 14 judges assigned a score of 1, and another 4 assigned a score of 0.5 to the incorrect name `ancestor`.

For the *math* case study, human judges consistently assigned the highest score to the correct name across all predicates, as shown in Table 12. Moreover, they successfully identified the correct name for predicate L, that is `isDivisor`, which none of the LLMs judges were able to recognize correctly.

5.5 Summary of Results

Overall, ChatGPT-4o and ChatGPT-o3mini appear to be the best-performing models, followed by Gemini and Command R+; Llama, FalconMamba, and Falcon3 performed the weakest.

It is worth noting that we chose some relatively small models to test whether they could identify useful names. Llama 3.2 and Falcon3 provided several correct suggestions, though not for every predicate. While few-shot prompting did not enhance performance for Fal-

Table 6 LLM judgment results for Case Studies 1 (*coauthors*) and 4 (*grandparent*, *cousins* and *lcm*)

Case Study - *Predicate*	ChatGPT (4o)	ChatGPT (o3mini)	Gemini	Command R+	Score
coauthors - *P*:					
coauthoredResearchPaper*	1	1	1	1	1.000
coauthorsWith-CommonPaper*	1	1	1	1	1.000
coAuthoredPaper*	0.5	1	0.5	1	0.750
coAuthorResearchers*	1	0.5	0.5	1	0.750
authoredTogether*	0.5	0.5	0.5	1	0.625
coauthors*	0.5	0.5	0.5	0.5	0.500
grandparent - *h0*:					
parent*	1	1	1	1	1.000
renameH0	0	0	0	0.5	0.125
cousins - *h3*:					
siblings*	1	1	1	0	0.750
sibling*	1	1	0	0	0.500
differentParents	0	0	1	1	0.500
isThirdDegreeRelative	0.5	0	0.5	0	0.250
parent	0	0	0	0	0.000
lcm - *G*					
findLeastCommon-MultipleIntermediate	1	0.5	1	0	0.625
computeProduct*	0.5	1	0.5	0	0.500
multiply*	0.5	1	0.5	0	0.500
greatestCommonDivisor	0	0	0	1	0.250
isMultipleOf	0	0	0	0	0.000
lowestCommonMultiple	0	0	0	0	0.000
multiplyAndAdd	0	0	0	0	0.000
lcm - *H*					
computeLcmFromGcd	1	1	1	0.5	0.875
divide*	0.5	1	0.5	0.5	0.625
lcmCalculation	1	0.5	0.5	0.5	0.625
divideValues*	0.5	1	0.5	0	0.500
leastCommonMultiple	0	0	0	1	0.250
highestCommonDivisor	0	0	0	0	0.000
divideAndSubtract	0	0	0	0	0.000

Judges assign a score of 1 for correct and precise names, 0.5 for imprecise or overly general names that are still correct, and 0 for incorrect names. The final score, shown in the last column, is the average of the individual scores given by the judges. Names that can be considered correct are marked with an asterisk ("*")

con3 and Llama, it led to a substantial improvement for FalconMamba. Nevertheless, their smaller size allows them to be run locally on machines with lower processing power, making this approach accessible to users without high-performance hardware.

The evaluation strategy that we adopted is automatic, meaning that human intervention is not required except for a final assessment, which should be faster than evaluating every single suggestion for each predicate. It could be argued that an LLM could make errors not only in making suggestions but also in judging them. Even though in these experiments we asked the LLMs to score only once, for a more robust evaluation, the procedure could be

Table 7 LLM judgment results for Case Study 2 (*family*)

Predicate	ChatGPT(4o)	ChatGPT(o3mini)	Gemini	Command R+	Score
h0					
parent*	1	1	1	1	1.000
ancestor	0.5	0	0.5	0.5	0.375
h1					
grandparent*	1	1	1	1	1.000
ancestor	0.5	0.5	0.5	0.5	0.500
h2					
commonAncestor*	1	1	1	0.5	0.875
ancestor	0.5	0.5	0.5	0	0.375
greatAncestor	0.5	0	0	0.5	0.250
h3					
sibling*	1	1	1	1	1.000
sister	0.5	0	0.5	0.5	0.375
h4					
cousin*	1	1	1	0.5	0.875
cousins*	1	1	0.5	0.5	0.750
halfSibling	0	0	0.5	0.5	0.250
fullSibling	0	0	0.5	0.5	0.250
h3	0	0	0	0	0.000

Judges assign a score of 1 for correct and precise names, 0.5 for imprecise or overly general names that are still correct, and 0 for incorrect names. The final score, shown in the last column, is the average of the individual scores given by the judges. Names that can be considered correct are marked with an asterisk ("*")

repeated multiple times per judge, and the results averaged. If a tie occurs between two or more suggestions, a new evaluation could be carried out, or the final decision could be left to domain experts.

Based on the results of the human evaluation, we can conclude that, overall, LLMs were generally capable of performing both the renaming and judgment tasks effectively. Their assessments aligned with those of human judges in most cases, with discrepancies emerging primarily in the more challenging or ambiguous predicates.

6 Discussion

Having unnamed predicates in the output rules may significantly hinder their understanding and reusability, especially for lengthy theories. However, current approaches do not seem to address this issue, with invented predicates typically being left unnamed. A possible solution is to use placeholders and select names from a predefined pool (Cropper & Dumančić, 2022), but this would require knowing beforehand which predicates should be invented and their arity, which is not always feasible. While all the predicates could be renamed manually, our approach showed promising results in accomplishing this task automatically. Even though the involvement of LLMs in logic is still not much investigated, the experiments show that this method was effective in most of the tested cases, even without

Table 8 LLM judgment results for Case Study 3 (*math*)

Predicate	ChatGPT (4o)	ChatGPT (o3mini)	Gemini	Command R+	Score
A					
isNumber*	1	1	1	1	1.000
numericValue*	0.5	1	0.5	0.5	0.625
isInteger	0	0	0.5	0.5	0.250
B					
isEven*	1	1	1	1	1.000
even*	1	1	1	0.5	0.875
isOdd	0	0	0	0	0.000
C					
isOdd*	1	1	1	1	1.000
odd*	1	1	1	0.5	0.875
isEven	0	0	0	0	0.000
D					
absoluteValue*	1	1	1	1	1.000
absValue*	1	1	1	0.5	0.875
negateIfNegative	1	0.5	1	0.5	0.750
isPositiveDifference	0	0	0.5	0	0.125
next	0	0	0	0	0.000
E					
add*	1	1	1	1	1.000
sum*	1	1	1	0.5	0.875
F					
subtract*	1	1	1	1	1.000
difference*	1	0.5	1	0.5	0.750
G					
multiply*	1	1	1	1	1.000
product*	1	1	1	0.5	0.875
H					
divide*	1	1	1	1	1.000
division*	1	1	0.5	0.5	0.750
quotient*	1	0.5	1	0.5	0.750
L					
divides	1	1	0.5	0.5	0.750
divisibleBy	1	0.5	1	0.5	0.750
isDivisible	1	0	1	1	0.750
isDivisibleBy	1	0.5	1	0.5	0.750
isDivisor*	0.5	1	0	0.5	0.500
modulo	0.5	0	0.5	0.5	0.375
M					
gcd*	1	1	1	1	1.000
accumulate	0	0	0.5	0.5	0.250
modularArithmetic	0	0	0.5	0.5	0.250
customOperation	0	0	0	0.5	0.125
N					
lcm*	1	1	1	1	1.000
complexCalculation	0	0.5	0.5	0.5	0.375
complexOperation	0	0.5	0.5	0.5	0.375
nestedCalculation	0	0.5	0.5	0.5	0.375

Table 8 (continued)

Predicate	ChatGPT (4o)	ChatGPT (o3mini)	Gemini	Command R+	Score
accumulateSequence	0	0	0.5	0.5	0.250
P					
greaterThan*	1	1	1	1	1.000
Q					
greaterThanOrEqual*	1	1	1	1	1.000
greaterEqual*	1	1	1	0.5	0.875
greaterOrEqual*	1	1	1	0.5	0.875
greaterThanOrEqualTo*	1	1	1	0.5	0.875
R					
lessThan*	1	1	1	1	1.000
S					
lessThanOrEqual*	1	1	1	1	1.000
lessEqual*	1	1	1	0.5	0.875
lessOrEqual*	1	1	1	0.5	0.875
lessThanOrEqualTo*	1	1	1	0.5	0.875
T					
equal*	1	1	1	1	1.000
equalTo*	1	1	1	0.5	0.875
isEqual*	1	1	1	0.5	0.875

Judges assign a score of 1 for correct and precise names, 0.5 for imprecise or overly general names that are still correct, and 0 for incorrect names. The final score, shown in the last column, is the average of the individual scores given by the judges. Names that can be considered correct are marked with an asterisk ("*")

Table 9 LLM judgment results for Case Study 6 (*reachability*)

Predicate	ChatGPT-5[1]	Gemini	Command R+	Score
inv1				
directlyConnected*	1	1	0.5	0.833
directConnection*	1	1	0.5	0.833
isConnected*	1	0.5	1	0.833
can reach	0	0.5	0.5	0.333

[1]Since ChatGPT-4o and ChatGPT-o3mini were no longer available at the time of the experiment, ChatGPT-5 was used instead

Judges assign a score of 1 for correct and precise names, 0.5 for imprecise or overly general names that are still correct, and 0 for incorrect names. The final score, shown in the last column, is the average of the individual scores given by the judges. Names that can be considered correct are marked with an asterisk ("*")

providing specific contexts or examples. In fact, zero-shot prompting was sufficient, as the LLMs immediately understood their task. This is particularly convenient when extensive computing power to fine-tune models is not easily accessible, as models for general-purpose tasks, trained on common knowledge data and code can still perform the task (Wu et al., 2025).

Table 10 Human judgment results for Case Studies 1 (*coauthors*) and 4 (*grandparent*, *cousins*, and *lcm*)

Case Study - *Predicate*	Score
coauthors - *P*	
coauthorsWithCommonPaper	0.857
coauthors	0.643
coAuthoredPaper	0.607
coauthoredResearchPaper	0.536
coAuthorResearchers	0.464
authoredTogether	0.429
grandparent - *h0*	
parent	1.000
renameH0	0.036
cousins - *h3*	
siblings	0.964
sibling	0.857
parent	0.107
differentParents	0.071
isThirdDegreeRelative	0.036
lcm - *G*	
computeProduct	0.821
multiply	0.750
findLeastCommonMultipleIntermediate	0.321
isMultipleOf	0.214
lowestCommonMultiple	0.143
multiplyAndAdd	0.143
greatestCommonDivisor	0.036
lcm - *H*	
divide	0.643
divideValues	0.643
computeLcmFromGcd	0.500
lcmCalculation	0.429
leastCommonMultiple	0.357
divideAndSubtract	0.143
highestCommonDivisor	0.143

Judges assign a score of 1 for correct and precise names, 0.5 for imprecise or overly general names that are still correct, and 0 for incorrect names. The final score, shown in the last column, is the average of the individual scores given by the judges. Names that can be considered correct are marked with an asterisk ("*")

6.1 Limitations

There are several limitations to this early-stage approach.

Although the selected case studies include structurally diverse theories, ranging from recursive and relational programs to graph reachability and scientific domains, they still represent controlled evaluation settings in which the intended semantics of the invented predicates can be assessed with relatively high confidence. These examples were intentionally selected to enable systematic evaluation of semantic interpretability and naming quality, while reducing ambiguity in the assessment of the generated predicate names. The rules express common knowledge and typical relationships, which the models may have already seen in the LP codes they have been trained on. To partially mitigate this issue, we also included a reachability theory generated through the ILP system POPPER, which does not appear in the literature in the presented form. Nonetheless, this initial step was necessary to

Table 11 Human judgment results for Case Study 2 (*family*)

Predicate	Score
h0	
parent	1.000
ancestor	0.429
h1	
grandparent	1.000
ancestor	0.393
h2	
commonAncestor	0.929
ancestor	0.393
greatAncestor	0.250
h3	
sibling	1.000
sister	0.250
h4	
cousins	0.893
cousin	0.857
fullSibling	0.107
halfSibling	0.071
h3	0.071

Judges assign a score of 1 for correct and precise names, 0.5 for imprecise or overly general names that are still correct, and 0 for incorrect names. The final score, shown in the last column, is the average of the individual scores given by the judges. Names that can be considered correct are marked with an asterisk ("*")

assess whether LLMs could correctly process them; otherwise, their ability to grasp even more specific relationships would have seemed unlikely.

Even within these controlled experimental settings, we encountered several challenges throughout the process. One issue is that, at times, some models fail to follow the request or the output format, which complicates the automation of the entire process. For instance, in some preliminary experiments, we noticed that some models also altered the body of the rule, even if instructed to find a name for the head predicate. This is not desirable, as it could completely change the meaning of the logic theory, which is assumed to be correct. That is why we decided to add "Do NOT change the body of the rules" to the instructions. Another constraint, "Give only ONE suggestion for each predicate.", was added to the prompt to limit the number of suggestions. While one could argue that this might cause the potentially correct answer to be overlooked, we addressed this concern by repeating the request multiple times. When this behavior persisted, we decided to disregard that particular answer for that specific predicate. Even if one answer is discarded, the presence of multiple models offering suggestions reduces the impact of this issue. Models also struggled to recognize that some names, despite being written in different formats, were conceptually equivalent (e.g., *lessThan*, *less_than*, *LessThan*), even when explicitly instructed to treat them as equal. As a result, it was necessary to manually standardize these names, to ensure that they were evaluated as the same. The issue persists to some extent: singular and plural forms are considered different (e.g., *cousins* and *cousin*), as well as names that include a verb or particle (e.g., *isEqual* or *equalTo*).

Lastly, this approach was unsuccessful when tested on predicates from a real-world dataset, Mutagenesis. This case study represents a particularly challenging setting, as nearly all predicate names in the background knowledge were masked, substantially reducing the semantic cues available to the models. Moreover, the dataset described a specialized domain that requires advanced notions of chemistry and biology. It should be noted that nearly all of the predicates' true names were masked, which significantly hindered the models' ability

Table 12 Human judgment results for Case Study 3 (*math*)

Predicate	Score
A	
isNumber	0.929
numericValue	0.679
isInteger	0.250
B	
isEven	0.929
even	0.750
isOdd	0.071
C	
isOdd	0.929
odd	0.750
isEven	0.071
D	
absoluteValue	0.964
absValue	0.964
negateIfNegative	0.250
isPositiveDifference	0.000
next	0.000
E	
add	0.714
sum	0.964
F	
subtract	0.857
difference	0.893
G	
multiply	0.786
product	0.964
H	
divide	0.750
quotient	0.679
division	0.821
L	
isDivisible	0.464
divides	0.643
isDivisor	0.750
divisibleBy	0.393
isDivisibleBy	0.500
modulo	0.107
M	
gcd	0.893
modularArithmetic	0.357
customOperation	0.286
accumulate	0.107
N	
lcm	0.893
nestedCalculation	0.321
complexOperation	0.286
complexCalculation	0.286

Table 12 (continued)

Predicate	Score
accumulateSequence	0.107
P	
greaterThan	1.000
Q	
greaterThanOrEqual	0.821
greaterOrEqual	0.821
greaterEqual	0.500
greaterThanOrEqualTo	0.857
R	
lessThan	0.929
S	
lessThanOrEqual	0.821
lessOrEqual	0.821
lessEqual	0.500
lessThanOrEqualTo	0.857
T	
equal	0.750
equalTo	0.857
isEqual	0.893

Judges assign a score of 1 for correct and precise names, 0.5 for imprecise or overly general names that are still correct, and 0 for incorrect names. The final score, shown in the last column, is the average of the individual scores given by the judges. Names that can be considered correct are marked with an asterisk ("*")

to identify them and grasp the relationship between them. These results suggest that purely context-based predicate renaming may become insufficient in domains requiring specialized scientific knowledge. In such scenarios, domain-specific adaptation, additional contextual information, or integration with external knowledge sources may be necessary. Nevertheless, our findings hint that domain-specific fine-tuning might be necessary for real-world applications (Rezayi et al., 2022), provided that enough computational resources are available. Alternatively, big models could be replaced by smaller or quantized fine-tuned versions, which are more computationally efficient and have been shown to not significantly decrease the quality of the results (Liu et al., 2024b; Kim et al., 2023; Dettmers et al., 2023).

7 Conclusion

In this paper, we have proposed a straightforward pipeline that leverages LLMs to find suggestions for unnamed predicates in logic theories, since their presence limits the interpretability. Based on the results obtained from some hand-crafted sets of rules, LLMs appear to be a promising solution to this problem. To the best of our knowledge, this is the first study to address this issue. Rather than providing an exhaustive benchmark across ILP domains, this work aims to systematically investigate whether LLMs can recover interpretable semantic labels for invented predicates across structurally heterogeneous logic theories. The limitations of this study are primarily related to the examples we used, and investigating more complex scenarios is an interesting future work. We believe that the suitability of the proposed names is primarily determined

by the context and semantics of the program and thus developed an approach that is independent of the language. Nevertheless, differences in language-specific conventions may play a role, and investigating this aspect could offer further insights into the generalization ability of LLM-based renaming approaches. Another topic worth exploring is the use of fine-tuned LLMs for domain-specific naming suggestions, as well as the development of quantized versions that can run locally on machines with lower processing power. Future research could also consider the potential of LLMs for directly performing PI and their application to other approaches such as approaches such as Learning From Interpretation Transitions (LFIT) (Ribeiro et al., 2021) to reduce the number of rules.

Author Contributions All authors contributed to the study conception and design. E. G. performed material preparation, data collection and analysis. E.G. wrote the first draft of the manuscript. All authors commented and revised the draft manuscript. All authors read and approved the final manuscript.

Funding Open access funding provided by Università degli Studi di Ferrara within the CRUI-CARE Agreement. This work was supported in part by the NII international internship program and the NII MoU Grant. Elisabetta Gentili contributed to this publication while attending the PhD programme in Engineering Science at the University of Ferrara, Cycle XXXVIII, with the support of a scholarship financed by the Ministerial Decree no. 351 of 9th April 2022, based on the NRRP - funded by the European Union - NextGenerationEU - Mission 4 "Education and Research", Component 1 "Enhancement of the offer of educational services: from nurseries to universities" - Investment 4.1 "Extension of the number of research doctorates and innovative doctorates for public administration and cultural heritage". This work has been partially supported by Spoke 1 "FutureHPC & BigData" of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 - Next Generation EU (NGEU) and by Partenariato Esteso PE00000013 - "FAIR - Future Artificial Intelligence Research" - Spoke 8 "Pervasive AI", funded by MUR through PNRR - M4C2 - Investimento 1.3 (Decreto Direttoriale MUR n. 341 of 15th March 2022) under the Next Generation EU (NGEU). Elisabetta Gentili and Fabrizio Riguzzi are members of the Gruppo Nazionale Calcolo Scientifico – Istituto Nazionale di Alta Matematica (GNCS-INdAM). Katsumi Inoue has been supported in part by JST CREST Grant Number JPMJCR22D3 and JSPS KAKENHI Grant Number JP25K03190, Japan. Tony Ribeiro has been funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or ERCEA. Neither the European Union nor the granting authority can be held responsible for them.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no competing interests.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of databases*. Addison-Wesley.

Alammar, J., & Grootendorst, M. (2024). Hands-On Large Language Models. O'Reilly, Sebastopol, CA, USA, https://www.oreilly.com/library/view/hands-on-large-language/9781098150952/.

Basu, K., Murugesan, K., & Atzeni, M. (2021). A hybrid neuro-symbolic approach for text-based games using inductive logic programming. Combining learning and reasoning: programming languages, formalisms, and representations

Bommasani, R., Hudson, D. A., & Adeli, E. (2021). On the opportunities and risks of foundation models. arXiv:2108.07258

Bordes, A., Usunier, N., & Garcia-Duran, A. (2013). Translating embeddings for modeling multi-relational data. Advances in Neural Information Processing Systems 26.

Brown, T., Mann, B., & Ryder, N. (2020). Language models are few-shot learners. In: H. Larochelle, M. Ranzato, & R. Hadsell (Eds.), *Advances in neural information processing systems* (Vol. 33, pp. 1877–1901). Curran Associates, Inc.

Chen, M., Tworek, J., & Jun, H. (2021). Evaluating large language models trained on code. arXiv:2107.03374

Cobbe, K., Kosaraju, V., & Bavarian, M. (2021). Training verifiers to solve math word problems. arXiv:2110.14168

Cohere For AI (2024) c4ai-command-r-plus-08-2024. https://doi.org/10.57967/hf/3135, https://huggingface.co/CohereForAI/c4ai-command-r-plus-08-2024

Colmerauer, A., & Roussel, P. (1996). The birth of Prolog, Association for Computing Machinery, New York, NY, USA, pp. 331–367. https://doi.org/10.1145/234286.1057820.

Creswell, A., Shanahan, M., & Higgins, I. (2022). Selection-inference: Exploiting large language models for interpretable logical reasoning. arXiv:2205.09712

Cropper, A., & Dumančić, S. (2022). Inductive logic programming at 30: A new introduction. *Journal of Artificial Intelligence Research, 74*, 765–850.

Cropper, A., & Morel, R. (2021a). Learning programs by learning from failures. *Machine Learning, 110*(4), 801–856.

Cropper, A., & Morel, R. (2021b). Predicate invention by learning from failures. arXiv:2104.14426

Cunnington, D., Law, M., Lobo, J., & Russo, A. (2024). The role of foundation models in neuro symbolic learning and reasoning. In: T. R. Besold, A. d'Avila Garcez, & E. Jimenez-Ruiz (Eds.), *Neural-symbolic learning and reasoning* (pp. 84–100). Springer.

Dash, A., Mathur, B., & Patil, P. (2024). Enhancing carbon emission tracking with gemini: An ai solution for sustainable development. https://doi.org/10.13140/RG.2.2.26165.90089

De Giorgis, S., Gangemi, A., & Russo, A. (2024). Neurosymbolic graph enrichment for grounded world models. arXiv:2411.12671

DeepMind, G. (2024). Gemini 1.5 Flash. https://ai.google.dev/gemini-api/docs/models/gemini?hl=it#gemini-1.5-flash.

Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms. In: A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, S. Levine (Eds.), *Advances in Neural Information Processing Systems* (Vol. 36, pp. 10088–10115). Curran Associates Inc.

Fadja, A. N., Lamma, E., & Riguzzi, F. (2017). Deep probabilistic logic programming. *Plp@ Ilp* (pp. 3–14)

Falcon-LLM Team. (2024). The Falcon 3 Family of Open Models. https://huggingface.co/blog/falcon3.

Feuerriegel, S., Hartmann, J., Janiesch, C., & Zschech, P. (2024). Generative ai. *Business & Information Systems Engineering, 66*(1), 111–126.

Gandarela, J. P., Carvalho, D. S., & Freitas, A. (2024). Inductive learning of logical theories with LLMs: A complexity-graded analysis. arXiv:2408.16779

Gelfond, M., & Lifschitz, V. (1988). The stable model semantics for logic programming. In: ICLP/SLP, https://api.semanticscholar.org/CorpusID:261517573.

Gu, A., & Dao, T. (2023). Mamba: Linear-time sequence modeling with selective state spaces. arXiv:2312.00752

Gulwani, S. (2011). Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices, 46*(1), 317–330.

Han, S.J., Ransom, K.J., Perfors, A. & Kemp, C. (2024). Inductive reasoning in humans and large language models. *Cognitive Systems Research, 83*, Article 101155. https://doi.org/10.1016/j.cogsys.2023.101155

Hinton, G. (2015). Distilling the knowledge in a neural network. arXiv:1503.02531

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., & Wang, H. (2024). Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology, 33*(8), 1–79.

Inoue, K., Doncescu, A., & Nabeshima, H. (2013). Completing causal networks by meta-level abduction. *Machine Learning, 91*(2), 239–277.

Inoue, K., Furukawa, K., Kobayashi, I., & Nabeshima, H. (2010). Discovering rules by meta-level abduction. In: Inductive logic programming: 19th international conference, ILP 2009, Leuven, Belgium, July 02-04, 2009. Revised Papers 19, Springer, pp. 49–64.

Inoue, K., Ribeiro, T., & Sakama, C. (2014). Learning from interpretation transition. *Machine Learning, 94*, 51–79.

Ishay, A., Yang, Z., & Lee, J. (2023). Leveraging large language models to generate answer set programs. arXiv:2307.07699

Jeblick, K., Schachtner, B., Dexl, J., Mittermeier, A., Stüber, A. T., Topalis, J., Weber, T., Wesp, P., Sabel, B. O., Ricke, J., & Ingrisch, M. (2024). Chatgpt makes medicine easy to swallow: An exploratory case study on simplified radiology reports. *European Radiology, 34*(5), 2817–2825.

Jiang, J., Wang, F., & Shen, J. (2024). A survey on large language models for code generation. arXiv:2406.00515

Kaddour, J., Harris, J., & Mozes, M. (2023). Challenges and applications of large language models. arXiv:2307.10169

Kakas, A. C., Kowalski, R. A., & Toni, F. (1992). Abductive logic programming. *Journal of Logic and Computation, 2*(6), 719–770.

Kareem, I., Gallagher, K., Borroto, M., Ricca, F., & Russo, A. (2025). Using learning from answer sets for robust question answering with llm. In C. Dodaro, G. Gupta, & M. V. Martinez (Eds.), *Logic programming and nonmonotonic reasoning* (pp. 112–125). Springer.

Kim, J., Lee, J. H., Kim, S., Park, J., Yoo, K. M., Kwon, S. J., & Lee, D. (2023). Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization. In A. Oh, T. Naumann, & A. Globerson (Eds.), *Advances in neural information processing systems* (Vol. 36, pp. 36187–36207). Curran Associates Inc.

Kobayashi, I., & Furukawa, K. (2009). Hypothesis selection using domain theory in rule abductive support for skills. SIG-SKL (Skill Science). Japanese Society for Artificial Intelligence. in Japanese.

Kojima, T., Gu, S. S., & Reid, M. (2022). Large language models are zero-shot reasoners. In: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Advances in Neural Information Processing Systems*, vol 35. Curran Associates, Inc., Red Hook, NY, USA, pp. 22199–22213, https://proceedings.neurips.cc/paper_files/paper/2022/file/8bb0d291acd4acf06ef112099c16f326-Paper-Conference.pdf

Kok, S., & Domingos, P. (2005). Learning the structure of markov logic networks. *Proceedings of the 22nd international conference on Machine learning* (pp. 441–448).

Koubaa, A., Boulila, W., & Ghouti, L. (2023). Exploring ChatGPT capabilities and limitations: A survey. *IEEE Access*. https://doi.org/10.1109/access.2023.3326474

Kowalski, R. A. (1988). The early years of logic programming. *Communications of the ACM, 31*, 38–43.

Kramer, S. (1995). Rapport technique OFAI-TR-95-32, Austrian Research Institute for Artificial Intelligence. Vienna. Predicate invention: A comprehensive view.

Law, M., Russo, A., & Broda, K. (2014). Inductive learning of answer set programs. In: Logics in Artificial Intelligence: 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings 14, Springer, pp. 311–325.

Law, M., Russo, A., & Broda, K. (2020). The ilasp system for inductive learning of answer set programs. arXiv:2005.00904

Li, Z., Cao, Y., & Xu, X. (2024). Llms for relational reasoning: How far are we? *Proceedings of the 1st International Workshop on Large Language Models for Code* (pp. 119–126).

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2024a). Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics, 12*, 157–173.

Liu, Y., He, H., & Han, T. (2024b). Understanding llms: A comprehensive overview from training to inference. arXiv:2401.02038

Magdy, S. M., Alwajih, F., Kwon, S. Y., Abdel-Salam, R., & Abdul-Mageed, M. (2024). Gazelle: An instruction dataset for Arabic writing assistance. https://arxiv.org/abs/2410.18163

Meta. (2024a). Llama 3.2 Model Card. https://github.com/meta-llama/llama-models/blob/main/models/llama3_2/MODEL_CARD.md.

Meta. (2024b). Llama 3.2: Revolutionizing edge AI and vision with open, customizable models. https://github.com/meta-llama/llama-models/blob/main/models/llama3_2/MODEL_CARD.md.

Min,S., Lyu, X., Holtzman, A., Artetxe, M., Lewis, M., Hajishirzi, H., & Zettlemoyer, L.(2022). Rethinking the Role of Demonstrations: What makes In-context Learning Work? In: EMNLP.

Mondillo, G., Frattolillo, V., & Colosimo, S. (2024). Basal knowledge in the field of pediatric nephrology and its enhancement following specific training of chatgpt-4 "omni" and gemini 1.5 flash. *Pediatric Nephrology,* 1–7.

Mooney, R. J. (1996). Inductive logic programming for natural language processing. *International conference on inductive logic programming* (pp. 1–22). Springer.

Moradi Dakhel, A., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M. C., & Jiang, Z. M. J. (2023). GitHub Copilot AI pair programmer: Asset or liability? *Journal of Systems and Software, 203*, Article 111734. https://doi.org/10.1016/j.jss.2023.111734

Muggleton, S. (1991). Inductive logic programming. *New Generation Computing, 8*, 295–318.

Muggleton, S., & Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. *Machine Learning Proceedings 1988* (pp. 339–352). Elsevier.

Muggleton, S., De Raedt, L., Poole, D., Bratko, I., Flach, P., Inoue, K., & Srinivasan, A. (2012). Ilp turns 20: Biography and future challenges. *Machine Learning, 86*, 3–23.

Muggleton, S. H., Lin, D., & Tamaddoni-Nezhad, A. (2015). Meta-interpretive learning of higher-order dyadic Datalog: Predicate invention revisited. *Machine Learning, 100*(1), 49–73.

Nguembang Fadja, A., Riguzzi, F., & Lamma, E. (2021). Learning hierarchical probabilistic logic programs. *Machine Learning, 110*(7), 1637–1693.

OpenAI. (2024a). GPT-4o System Card. https://openai.com/index/gpt-4o-system-card.

OpenAI. (2024b). Hello GPT-4o. https://openai.com/index/hello-gpt-4o.

OpenAI. (2025). OpenAI o3-mini System Card. https://openai.com/index/o3-mini-system-card.

Pearce, H., Tan, B., Ahmad, B., Karri, R., & Dolan-Gavitt, B. (2023). Examining zero-shot vulnerability repair with large language models. In: 2023 IEEE Symposium on Security and Privacy (SP), pp. 2339–2356, https://doi.org/10.1109/SP46215.2023.10179324

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research, 21*(140), 1–67.

Rao, A., Pang, M., Kim, J., Kamineni, M., Lie, W., Prasad, A. K., Landman, A., Dreyer, K., & Succi, M. D. (2023). Assessing the utility of ChatGPT throughout the entire clinical workflow: Development and usability study. *Journal of Medical Internet Research, 25*, Article e48659.

Rezayi, S., Liu, Z., & Wu, Z. (2022). Agribert: Knowledge-infused agricultural language models for matching food and nutrition. *IJCAI,* 5150–5156.

Ribeiro, T., Folschette, M., Magnin, M., & Inoue, K. (2021). Learning any memory-less discrete semantics for dynamical systems represented by logic programs. *Machine Learning*. https://doi.org/10.1007/s10994-021-06105-4

Riguzzi, F. (2023). *Foundations of probabilistic logic programming: Languages, semantics, inference and learning* (2nd ed.). River Publishers.

Sato, T., Inoue, K., & Sakama, C. (2018). Abducing relations in continuous spaces. *IJCAI* (pp. 1956–1962).

Sha, J., Shindo, H., & Kersting, K. (2024). Neuro-symbolic predicate invention: Learning relational concepts from visual scenes. *Neurosymbolic Artificial Intelligence,* 1–26.

Srinivasan, A., King, R. D., & Muggleton, S. H. (1997). Carcinogenesis predictions using ilp. *International Conference on Inductive Logic Programming* (pp. 273–287). Springer.

Srinivasan, A., Muggleton, S. H., Sternberg, M. J., & King, R. D. (1996). Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence, 85*(1–2), 277–299.

Srivastava, A., Rastogi, A., & Rao, A. (2022). Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. arXiv:2206.04615

Stahl, I. (1993). Predicate invention in ilp-an overview. *European conference on machine learning* (pp. 311–322). Springer.

Stahl, I. (1995). The appropriateness of predicate invention as bias shift operation in ILP. *Machine Learning, 20*, 95–117.

Talmor, A., Herzig, J., & Lourie, N. (2018). Commonsenseqa: A question answering challenge targeting commonsense knowledge. arXiv:1811.00937

Tarau, P. (2025). On llm-generated logic programs and their inference execution methods. arXiv:2502.09209

Touvron, H., Lavril, T., & Izacard, G. (2023). Llama: Open and efficient foundation language models. arXiv:2302.13971

Valmeekam, K., Olmo, A., & Sreedharan, S. (2022). Large language models still can't plan (a benchmark for llms on planning and reasoning about change). NeurIPS 2022 Foundation Models for Decision Making Workshop.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. U., & Polosukhin, I. (2017). Attention is all you need. In: Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., & Garnett, R. (Eds.), Advances in Neural Information Processing Systems, vol 30. Curran Associates, Inc., Red Hook, NY, USA, https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Vennekens, J., Verbaeten, S., & Bruynooghe, M. (2004). Logic programs with annotated disjunctions. In: Logic Programming: 20th International Conference, ICLP 2004, Saint-Malo, France, September 6-10, 2004. Proceedings 20, Springer, pp. 431–445.

Wang, R., Zelikman, E., & Poesia, G. (2023). Hypothesis search: Inductive reasoning with language models. arXiv:2309.05660

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems, 35*, 24824–24837.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., & Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In: *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*. Association for Computational Linguistics, Online, pp. 38–45, https://www.aclweb.org/anthology/2020.emnlp-demos.6

Wu, Z., Zhang, L., Cao, C., Yu, X., Liu, Z., Zhao, L., Li, Y., Dai, H., Ma, C., Li, G., Liu, W., Li, Q., Shen, D., Li, X., Zhu, D., & Liu, T. (2025). Exploring the trade-offs: Unified large language models vs local fine-tuned models for highly-specific radiology NLI task. *IEEE Transactions on Big Data, 11*(3), 1027–1041. https://doi.org/10.1109/TBDATA.2025.3536928

Yang, Z., Dong, L., Du, X., Cheng, H., Cambria, E., Liu, X., Gao, J., & Wei, F. (2020). On the sub-layer functionalities of transformer decoder. In: Cohn T, He Y, Liu Y (Eds.), Findings of the Association for Computational Linguistics: EMNLP 2020. Association for Computational Linguistics, Online, pp. 4799–4811. https://aclanthology.org/2020.findings-emnlp.432/

Yang, Y., Wang, L., Shi, S., Tadepalli, P., Lee, S., & Tu, Z. (2024). On the sub-layer function-alities of transformer decoder. In: Cohn, T., He, Y., Liu, Y (Eds.), Language models as inductive reasoners. In: Graham Y, Purver M (Eds.), Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, St. Julian's, Malta, pp 209–225, https://aclanthology.org/2024.eacl-long.13/

Zhang, Z., Yilmaz, L., & Liu, B. (2024). A critical review of inductive logic programming techniques for explainable AI. *IEEE Transactions on Neural Networks and Learning Systems, 35*(8), 10220–10236. https://doi.org/10.1109/TNNLS.2023.3246980

Zheng, L., Chiang, W., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li,D., Xing, E.P., Zhang, H., Gonzalez, J.E., Stoica, I.(2023). Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems, 36*, 46595–46623. https://proceedings.neurips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html

Zuo, J., Velikanov, M., Rhaiem, D.E., Chahed, I., Belkada, Y., Kunsch, G., Hacid, H.: Falcon Mamba: The First Competitive Attention-free 7B Language Model (2024). https://arxiv.org/abs/2410.05355

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.