Towards an Evaluation Framework for Generated Formal Knowledge

Maxime Haurel[1,2], Armelle Brun[1] and Mathieu d'Aquin[1]

[1]Université de Lorraine, CNRS, LORIA, F-54000 Nancy, France
[2]Datanello, Nancy, France

Abstract

Constructing a Knowledge Base (KB) is an expensive effort which requires expertise in a specific domain and in knowledge engineering. While existing works show that LLMs are widely used to automate knowledge graph construction, the resulting representations often lack the strict structure required for complex problem-solving. Addressing this limitation requires shifting toward more operational KBs such as those that can be expressed in Prolog. Furthermore, constructing such KBs demands advanced logical capabilities, yet the potential of Large Reasoning Models (LRMs) for this task remains unexplored. In this work, we propose a set of metrics designed to evaluate the ability of LRMs to generate formal knowledge in order to solve domain-specific problems. With the assistance of LLMs, we curate – based on a collection of LRM-generated Prolog programs – common issues in generated KBs, and derive metrics that can be used to assess such issues. The results we obtain show that the metrics we propose are covering the considered issues, meaning that they are robust enough to assess if a Prolog program is sufficiently well-formed to be integrated in a larger KB. In our experiments, no program achieves to derive an appropriate conclusion, whether it is because the program is badly formed or because the derived conclusion does not match the expected conclusion. Based on the assumption that training LRMs towards the proposed metrics could improve this state of affairs, these insights pave the way towards a bigger project that aims to automatically and incrementally construct a KB with LRMs.

Keywords

knowledge base generation, evaluation metrics, formal knowledge, prolog

1. Introduction

Knowledge Bases (KBs) are explainable and deterministic objects from which we can infer conclusions. However, constructing these objects requires the significant involvement of knowledge engineers and domain experts, making it time-consuming, expensive, and prone to errors [1]. Approaches using Large Language Models (LLMs) to construct KBs semi-automatically and automatically have emerged over the past years with the works of [2, 3] for example. Indeed, even though LLMs exhibit low-explainability, they show great utility in building a KB, acting as a knowledge engineer, a domain expert, and reducing costs. More recently, Large Reasoning Models (LRMs) have extended the scope of LLMs to better perform on reasoning-intensive tasks [4, 5].

This paper is part of a larger research endeavor in which we propose to rely on LRMs to construct a large-scale and reliable KB in a specific domain. At this stage, we have selected Prolog as the knowledge representation language on which this KB will rely (see [6] for details). Due to the large-scale dimension of the desired generated KB, we envision the approach to build it as an incremental process. More precisely, from a collection of problems (i.e. questions requiring reasoning and domain knowledge), a problem is sampled and the LRM is asked to generate a Prolog program that solves the problem. Then, the generated Prolog program is evaluated against specific metrics, and is added to the KB if the evaluation is satisfied. The process is then repeated until the training corpus has been entirely processed.

In this paper, we present an evaluation framework of generated knowledge in a general manner, that is specifically instantiated in the Prolog language. While numerous studies focused on constructing

3rd International Workshop on Evaluation of Language Models in Knowledge Engineering Co-located with the European Semantic Web Conference (ESWC 2026)

maxime.haurel@loria.fr (M. Haurel); armelle.brun@loria.fr (A. Brun); mathieu.daquin@loria.fr (M. d'Aquin)
0009-0006-8010-808X (M. Haurel); 0000-0002-9876-6906 (A. Brun); 0000-0001-7276-4702 (M. d'Aquin)

Knowledge Graphs (KGs) and ontologies automatically by using LLMs [7, 8, 9, 3, 10, 11, 12, 13], the generation of a Prolog program needs a specific, rigorous evaluation to ensure the quality of the resulting KB at the end of the incremental construction. Indeed, even if a relation exists between Prolog and the RDF/OWL languages, Prolog can embed rules which are more expressive than ontologies and, therefore, more difficult to evaluate. These rules are the core of the KB since they encode all the domain logic needed to derive conclusions from the KB. We thus take inspiration from previous studies building KGs with LLMs but keep in mind that evaluating rules is a remaining challenge that, to the best of our knowledge, other studies have not tackled. We propose several metrics that allow us to evaluate the generated Prolog program according to different dimensions: syntactic, structural, and task-related. These metrics are curated with the assistance of LLMs over a dataset of medical problems, focused on diagnosis tasks. The set of metrics we propose will be used for the larger aim of incremental construction of a KB using LRMs. This set of metrics is also already designed to support the need of fine-tuning of the LRM at a later stage of this research: Indeed, the metrics are designed to work as a reward signal used in a Reinforcement Learning (RL) algorithm such as GRPO [14].

This paper is structured as follows. We present related work in Section 2, our proposed evaluation framework in Section 3, the experiments we led in Section 4, the conclusion in Section 5, and the possible improvements in Section 6.

2. Related Work

2.1. Automatic Knowledge Base Construction

Automatic KG construction is an active research field within the broader domain of Knowledge Representation and Reasoning (KRR) [15]. Because the graph structure of entities and relations allows for efficient concept representation [16], several studies have leveraged LLMs to extract triples from unstructured text [7, 8, 9, 3]. Although our objective is to construct formal KBs rather than KGs, we draw inspiration from these LLM-assisted construction techniques. Despite their success, these approaches primarily focus on a single iteration, which is unsuitable for constructing large-scale KBs from scratch. Attempting to prompt an LLM to generate a massive KB all at once inevitably leads to context loss and severe hallucinations [17]. Consequently, constructing reliable, large-scale KBs requires an incremental approach, integrating new knowledge step-by-step.

To address this, the recent literature focuses on the iterative construction of KBs using LLMs. For instance, [11] propose an incremental, zero-shot framework that extracts entities and relations to build a KG sequentially. Their approach employs multiple modules to distill documents into semantic blocks and merge local entities into a global set. The authors extended this work with ATOM [10], which improves fact extraction and introduces temporal aspects into the KG. Similarly, the Graphiti engine [12] dynamically builds a KG from unstructured conversational data and structured business data. Another notable approach has been proposed by [13] with a zero-shot and external knowledge-agnostic framework. Although we want to build on such ideas to extend the research presented in this paper, these works do not specifically address problem-solving and do not work with rules as in our case.

Indeed, while these methods represent data as graphs, representing the complex rules derived from the knowledge used by LLMs requires more expressive formalisms than those used for knowledge graphs. Logic programming languages such as Datalog [18] and Prolog [19] are powerful tools for describing both facts and rules. Prolog has been successfully utilized for robotic planning [20], arithmetic reasoning [21], and domain-specific question answering over KBs [22]. It has also been shown that these KBs can be instantiated with factual data extracted from unstructured text [23]. Despite these advancements, to the best of our knowledge, our work is the first to focus on automatically and incrementally constructing a KB specifically tailored for problem-solving.

2.2. Evaluation of LLM-Generated Knowledge Bases

As the scale of automatically and incrementally constructed KBs grows, ensuring their reliability requires robust evaluation metrics. The authors of [24] assess KB systems based on completeness – defined as the percentage of entailed answers successfully returned for a given query – and soundness, aggregating these into a composite measure. When building KGs, the evaluation is often done by evaluating the precision and recall of extracted triples, alongside the semantic equivalence of named entities [9, 11], or by a broad automatic validation framework [25], using LLMs to validate whether an extracted triple is correct. This is, however, not applicable in our case, where a ground truth that states the facts and rules that should be present in the generated Prolog programs is not available.

Additionally, in [7] are introduced metrics specifically for LLM-generated KGs, evaluating fact extraction performance, hallucination rate, and ontology conformance. However, the generation of KGs/KBs in formal languages introduces unique evaluation challenges, such as controlling for inconsistency and redundancy [26]. The authors of [27] highlight the necessity of conducting both qualitative and quantitative analyses to properly assess LLM-generated ontologies. To systematically evaluate these LLM-generated ontologies, [28] propose comparing LLM-generated ontologies against human-built ones, spanning structural integrity, semantic consistency, and downstream task suitability. Furthermore, automated validation remains a complex task and the authors of [29] demonstrate that using LLMs as the sole validator of a KG leads to weak performance compared to human-LLM collaboration. In the next sections, we build on these ideas to construct metrics for evaluating generated Prolog programs.

Specifically regarding the evaluation of a generated Prolog program, [21] evaluates the accuracy of the derivation of a Prolog program. However, the authors evaluate only programs whose output is an integer, while we aim to enable more general scenarios, without this limitation, in our objective of generating Prolog programs tailored to problem-solving.

Evaluating a large-scale, incrementally generated Prolog KB presents an even more significant bottleneck. Developing a large collection of custom scrapers to evaluate every syntactic and semantic aspect of generated Prolog programs is highly complex and unscalable. Consequently, the surge in LLM capabilities has introduced LLM-as-a-judge techniques. The authors of [30] demonstrate the use of LLMs as knowledge evaluation agent, mitigating architectural bias by using different models for generation and evaluation. Similarly, in [31], LLMs are utilized to evaluate generated Prolog code, applying a majority voting mechanism to ensure semantic and syntactic validity. While these LLM-as-a-judge approaches are interesting, we aim to build an evaluation framework that is more precise and that relies on non-LLM solutions to enable better interpretability and control over the evaluation.

While LLM-as-a-judge approaches are highly capable, methods such as majority voting are computationally expensive. In this paper, we propose an evaluation framework that deliberately minimizes the use of LLMs. By reducing our dependence on LLMs, our approach offers the benefit of low computational costs, deterministic explainability, and provides fine-grained control over the evaluation of the generated formal knowledge. As a result, it is also more suitable to be integrated within the process of fine-tuning models for better knowledge generation.

3. Proposed Approach

This section details our framework towards an evaluation of a Prolog program with the objective of incrementally constructing a KB automatically. Section 3.1 explains our approach to find the critical aspects to focus on when evaluating such programs and Section 3.2 lists and defines the metrics we designed to actually evaluate Prolog programs. Experiments are presented in Section 4.

3.1. Metrics Discovery

As we previously stated, evaluating knowledge represented in a Prolog program is challenging as it involves evaluating the program under many dimensions (syntactic, structural, etc.).

Prolog programs have been generated, extending the work in [6] and based on a collection of medical problems taken from a dataset that we describe in Section 4. Because we want the generated Prolog program to be domain-specific, we focus on a subset of this dataset that is targeting medical diagnosis specifically. The dataset is not labeled by default so we employed LLM-as-a-judge to identify the diagnosis-specific problems. We use Grok-4-Fast (a state-of-the-art model among the low-cost LLMs) to decide whether each problem represents a diagnostic task or not. This labeling ensures that we can confidently prompt the LRM to generate Prolog programs whose ultimate rule for querying derives a truth value for a predicate called 'diagnosis'. From this filtered set, we sampled 100 problems and generated their corresponding Prolog programs.

To provide a set of metrics that is based on a reproducible method – and not based on our personal judgment – we employ Claude-4.6-Sonnet (one of the best models on the market) to analyze each generated Prolog program. Concretely, the LLM is given the initial medical problem and the KB that has been generated. The LLM is then instructed to output what should be changed in the KB to actually solve the initial medical problem. We reference a change outputted by the LLM as an issue. This process yielded a total of 477 distinct issues across our sample.

Before being able to create metrics from this set of issues, we want to group similar issues into higher-level groups. We employ Claude-4.6-Sonnet a second time to assign one short tag to each issue. Finally, we manually curated and grouped the tags. The prompts and details of these implementations are available in the dedicated GitHub repository[1].

Based on the tags obtained from the previous process, we construct a mapping between issues and metrics. For a set of tags, we create a metric that is designed to cover the issues. The experiments assessing whether our new metrics cover the issues are described in Section 4 and the mapping issues-metrics is presented in Table 1. Note that, a significant amount of the tags returned by the LLM were domain-related (or task-related) tags. Table 1 excludes task-based metrics, although they are defined later, as this work focuses on metrics that assess the general quality of a Prolog program. Task-based metrics are domain- and problem-specific, as they depend on the correct use of domain knowledge to produce a valid solution. Therefore, since our objective is to evaluate Prolog programs independently of the task, domain-related tags are also disregarded for now.

3.2. Metrics

To evaluate the quality of a Prolog program, we create several metrics, each designed based on the groups of issues we made in Section 3.1. As a reminder, the relations between the groups of issues and the metrics is described in Table 1, and a more precise description of each metric is presented in Table 2. Each metric produces a score between 0 and 1 to make them easily aggregatable and composable, for example, as a reward function for a possible fine-tuning of the LRM.

3.2.1. Syntactic-Based Metrics

It is crucial for a Prolog program to be syntactically valid in order to be integrated in a larger KB. Because we generate the Prolog program in the reasoning traces, it is highly likely that the generated Prolog program will not always be syntactically valid.

Syntactic Correctness This metric assesses the syntactic validity of a Prolog program by loading it in the SWI-Prolog interpreter [32] and collecting the errors raised by the interpreter if there are any. The formula is the following.

$$S_{syn} = \frac{\sum_{i=1}^{k} \frac{1}{(n_i+1)}}{k}$$

where n_i is the number of errors detected by the interpreter for clause i and k is the number of clauses of the program. S_{syn} is equal to 0 if the Prolog program is empty.

[1] https://github.com/MHaurel/towards-an-evaluation-framework-for-generated-formal-knowledge

Category	Metric	Issues
Syntactical	Syntactic Correctness	malformed rule structure, atom/variable confusion, general syntax error, invalid syntax annotations, invalid atom syntax, invalid numeric syntax
Structural	Compliance w.r.t. Vocabulary	malformed predicate/rule structure, wrong predicate used, atom/identifier typo, argument structure/type mismatch, wrong predicate name, inconsistent head body unification, missing entry point, argument order mismatch, wrong predicate call order, argument value mismatch, arity mismatch, missing predicate definition, predicate overloading, undefined predicate call
	Unused Terms	redundant fact, unused variable binding, redundant factor wrapping, duplicate predicate definition, redundant condition, unused argument, duplicate solutions via backtracking, unused predicate
	Unbound Variables	anonymous variable misuse, missing variable unification, redundant variable unification, incorrect variable unification, unbound variable in head, variable scoping across goals, missing output binding, unbound variable in rule, unbound variable in facts, singleton variable

Table 1
Mapping issues-metrics for Prolog-specific metrics. The metric is constructed to cover the associated issues.

Category	Metric	Detail	Notation
Syntactical	Syntactic Correctness	Ratio of errors triggered by the Prolog interpreter per clause in the program.	S_{syn}
Structural	Compliance w.r.t. Vocabulary	Compliance with respect to a defined vocabulary.	S_{vocab}
	Unused Terms	Ratio of used clauses compared to the total clauses when deriving the conclusion.	S_{used}
	Unbound Variables	Detection of unbound variables that are useless in the derivation of the conclusion.	$S_{unbound\_vars}$
Task	LRM Output Consistency	Semantic consistency between the logical conclusions derived from the Prolog program and the natural language answer produced during the same generation.	S_{con}
	Derivation Performance	Performance of the ability of the KB to derive conclusions that match the ground truth.	$S_{performance}$

Table 2
Metrics used for evaluation of a Prolog program.

3.2.2. Structural-Based Metrics

The first guard with the syntactic evaluation allows us to identify some issues but we need to go further, by evaluating the general structure of the program.

Compliance w.r.t. Vocabulary Because the final goal is to build a large-scale KB in an incremental manner, we need to obtain Prolog programs whose terms are constrained to be standardized across different programs. We defined a vocabulary which is presented in Table 3. This vocabulary defines the

Predicate (Arity)	Pos.	Semantic Type	Description
patient (2)	0	`patient_id`	The identifier of the patient.
	1	`characteristic`	A characteristic of the patient.
symptom (2)	0	`patient`	The patient identifier (the individual who has the symptom).
	1	`symptom`	A clinical symptom or sign observed in the patient (e.g., fever, cough, headache).
condition (2)	0	`patient`	The patient identifier.
	1	`medical condition`	A medical condition, disease, or syndrome (e.g., pneumonia, diabetes, hypertension).
diagnosis (1)	0	`medical diagnosis`	The final diagnostic conclusion for the patient case (e.g., stroke, myocardial_infarction).

Table 3
Reference vocabulary for the medical diagnosis Prolog program. The table defines the expected predicates, their arity, and the semantic type labels used as reference embeddings for parameter validation as well as their positions within the predicate.

names, arity and parameters' types the LRM should use in the final Prolog program generated.

Formally, the metric is defined as:

$$S_{vocab} = w_1\, S_{predicates} + w_2\, S_{arity} + w_3\, S_{type} \quad \in [0, 1]$$

Where $S_{predicates}$ is the compliance of the Prolog program to the vocabulary regarding the names of the predicates, S_{arity} is the compliance of the Prolog program to the vocabulary regarding the arity of the predicates, and S_{type} is the compliance of the Prolog program to the vocabulary regarding the type of the parameters. Note that for the moment, the weights (i.e. w_1, w_2, and w_3) are fixed to $\frac{1}{3}$.

We define a clause as compliant with respect to the vocabulary if, for each compound, the name of the predicate is defined in the vocabulary. The arity of a clause is compliant if, for each compound, the arity matches the defined arity in the vocabulary. Lastly, a parameter is compliant if its semantic cosine similarity with regard to embeddings compared to a reference parameter value is above a defined threshold. We defined a threshold of 0.27, a value obtained by maximizing the F1-score on a manually annotated validation set of 50 term pairs. For this metric, the embedding model we use is the all-MiniLM-L6-v2.

Unused Terms Even though unused clauses (rules) do not block a Prolog program to execute properly, keeping them prevents us from having an optimized KB. We then detect unused terms by running a Prolog meta-interpreter over the KB. This interpreter will, based on the query *?- diagnosis(X).*, collect the proof tree to determine which clauses are actually used in the derivation of the conclusion. The formula is the following:

$$S_{used} = \frac{|C_{used}|}{|C_{all}|} \quad \in [0, 1]$$

where $|C_{used}|$ represents the number of used clauses and $|C_{all}|$ represents the total number of clauses.

Unbound Variables In Prolog, we distinguish variables and constants. A constant is a specific, fixed atomic object (e.g. *fever*), while a variable is an unspecified object that will be instantiated when a conclusion will be derived by the KB. An unbound variable is a variable that is bound to nothing, where the head of the rule uses a variable but no premises re-use it. A Prolog rule including such unbound variables will not produce meaningful output as the knowledge is so strictly encoded that the conclusion derived by the KB will always be the same, no matter what the situation (i.e. symptoms, etc.) is. We search for unbound variables by running a custom parser over the Prolog code. Concretely, the parser

analyzes each clause and parses the unbound variables. We then construct the metric over the following formula:

$$S_{unbound\_vars} = \frac{|C_{without\_unbound}|}{|C_{all}|} \quad \in [0, 1]$$

where $|C_{without\_unbound}|$ represents the number of clauses without unbound variables and $|C_{all}|$ represents the total number of clauses.

3.2.3. Task-Based Metrics

We generate Prolog programs in the aim of representing problem-solving knowledge to finally be able to derive a conclusion symbolically. Task-based metrics represent the performance dimension of the evaluation framework. They are viewed as a state of the performance we are currently achieving in order to compare to future improvements.

Derivation Performance This metric aims to assess whether the conclusion derived by the KB matches the expected answer to the initial problem according to the dataset used. We employ LLM-as-a-judge techniques with GPT-4o to compare the conclusion derived by the KB to the actual expected answer. The formula is the following:

$$S_{performance} = \frac{1}{n} \sum_{i=0}^{n} sim(gt, c_i)$$

Where n is the number of conclusions derived by the KB, sim is the function employed to compare two strings using gpt-4o, gt is the ground truth expressed as natural language, and c_i is one of the conclusions derived by the KB.

LRM Output Consistency VS KB When generating a Prolog program in the reasoning traces, the LRM also generates an answer (*<answer></answer>*). We aim to check whether the answer generated by the LRM matches the conclusion derived by the generated KB. We compare the answer generated by the LRM to the conclusions derived by the KB using LLM-as-a-judge with GPT-4o [33] to match the research done in [34]. While it does not evaluate the performance of the LRM or the program, we view this as a measure of the internal consistency of the LRM. Because the KB can derive more than one conclusion, each conclusion is compared to the answer generated by the LRM. The formula is then the following:

$$S_{con} = \frac{1}{n} \sum_{i=0}^{n} sim(a, c_i)$$

Where n is the number of conclusions derived by the KB, sim is the function employed to compare two strings using gpt-4o, a is the answer generated by the LRM, and c_i is one of the conclusions derived by the KB.

4. Experiments

In this section, we discuss the experiments we designed to test the metrics along with the results we obtained. We assess how much the metrics we designed are able to classify the Prolog programs according to the groups of issues we described earlier in Section 3.1. We first present the dataset we use, the result of each metric on our dataset, and finally the classification performance of the metrics.

As stated previously, all the prompts and experiments are available in a dedicated GitHub repository [2].

[2] https://github.com/MHaurel/towards-an-evaluation-framework-for-generated-formal-knowledge

Input	Output	Post-processed
Given the symptoms of sudden weakness in the left arm and leg, recent long-distance travel, and the presence of swollen and tender right lower leg, what specific cardiac abnormality is most likely to be found upon further evaluation that could explain these findings?	The specific cardiac abnormality most likely to be found in this scenario is a patent foramen ovale (PFO). This condition could allow a blood clot from the venous system, such as one from a deep vein thrombosis in the leg, to bypass the lungs and pass directly into the arterial circulation. This can occur when the clot moves from the right atrium to the left atrium through the PFO. Once in the arterial system, the clot can travel to the brain, potentially causing an embolic stroke, which would explain the sudden weakness in the left arm and leg. The connection between the recent travel, which increases the risk of deep vein thrombosis, and the neurological symptoms suggests the presence of a PFO facilitating a paradoxical embolism.	patent foramen ovale

Table 4
An example of the *medical-o1-reasoning-SFT* dataset. The last column is the single value that the Prolog program should output in this example.

Dataset The dataset we use is the medical-o1-reasoning-SFT dataset [3] [34]. This dataset is a collection of medical problems, coming with the chain-of-thoughts reasoning, and the desired answer as natural language. It has been designed to train LLMs to better perform at reasoning tasks. An example is provided in Table 4.

LRM Configuration The LRM used to generate the Prolog programs is Cogito:14B[4] [4]. In an experiment extending the work presented in [6], we found that the Cogito models were able to output Prolog programs that are often syntactically correct.

4.1. Metrics relevance

In order to validate the relevance of the metrics we just presented, we test them against the fixes/issues predicted by Claude-Sonnet-4.6, as described in Section 3.1. Practically, this validation involves using the mapping tags-metrics to answer the following question: *How confidently are our metrics detecting the same issues as labeled by Claude-4.6-Sonnet?*

We employ classification measures and, because the metrics are scoring between 0 and 1, we compute f1-score, precision, and recall for a range of thresholds. Formally, we consider thresholds in $[0, 1]$ with step size $\Delta = 0.02$:

$$T = \{0, \Delta, 2\Delta, \ldots, 1\}.$$

For each threshold t_k, a binary prediction is obtained by thresholding the score:

$$\hat{y}_i(t_k) = \begin{cases} 1 & \text{if } s_i \geq t_k, \\ 0 & \text{otherwise.} \end{cases}$$

where s_i is the score obtained for the i^{th} program and t_k is the k^{th} threshold of T.

We then plot the classification scores over the threshold and we obtain the plots displayed in Figure 1. The motivation behind using these curves is to find the best threshold above which we consider the Prolog program as satisfying regarding the metric.

The ROC curves showing the performance of the metrics are showed in Figure 2. The ROC analysis shows that the proposed metrics are able to detect the associated cases, achieving AUC scores above a

[3] https://huggingface.co/datasets/FreedomIntelligence/medical-o1-reasoning-SFT
[4] https://huggingface.co/deepcogito/cogito-v1-preview-qwen-14B

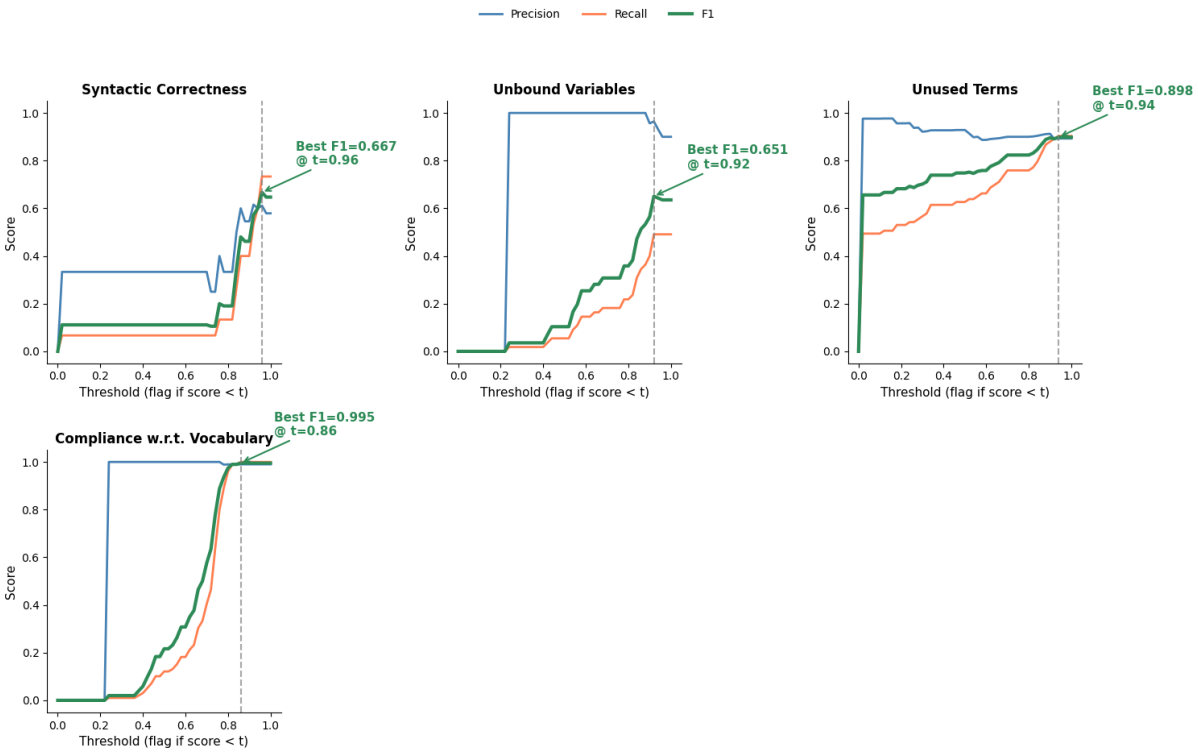

Figure 1: Classification-threshold curves for each metric compared to the ground truth we curated with the assistance of LLMs.

random guess. *Unbound Variables* is the least performing metric with an AUC score of 0.726. *Compliance w.r.t. Vocabulary* and *Syntactic Correctness* are the best performing metrics against the ground truth with near scores of respectively 0.823 and 0.816. *Unused Terms* is also performing well with an AUC score of 0.775.

We conducted an analysis to understand where the metrics fail. The *Unused Terms* metric, in particular, sometimes misclassifies programs. For instance, a constant parameter may appear only once while being dynamically instantiated through a variable, a level of detail that the current metric does not yet capture. Another limitation concerns our *Syntactic Correctness* metric. Currently, this metric evaluates the program clause-by-clause by splitting the text at every period ('.'), which is the standard end character for a Prolog clause. However, this naive parsing approach fails when a period is used elsewhere – such as within a floating-point number or a comment. In these cases, the text is incorrectly split, breaking the evaluation pattern and leading to misclassifications. It shows that, while our metrics are able to rather confidently discriminate a correct program from an incorrect one, we need to review them deeper to improve them.

Also, we observed that Claude Sonnet 4.6 (i.e. the model used to discover the metrics) assigned wrong tags to some programs, thereby creating false negatives. Because the subject discussed in this paper is a framework of evaluation to discriminate incorrect Prolog programs, we leave this aside for the moment but we keep it in mind.

4.2. Task-based results

Task-based results regarding the performance are presented in Table 5. These represent the current state of the performance of the generated program. This baseline will be used to compare future results. Table 6 shows the measure of the consistency between the conclusions of the generated program and the actual answer of the LRM.

The *LRM Output Consistency* metric scores at a mean of 0.712 for the Prolog programs that are valid (i.e. for which the diagnosis query actually returns something). However 37 problems are not valid (for

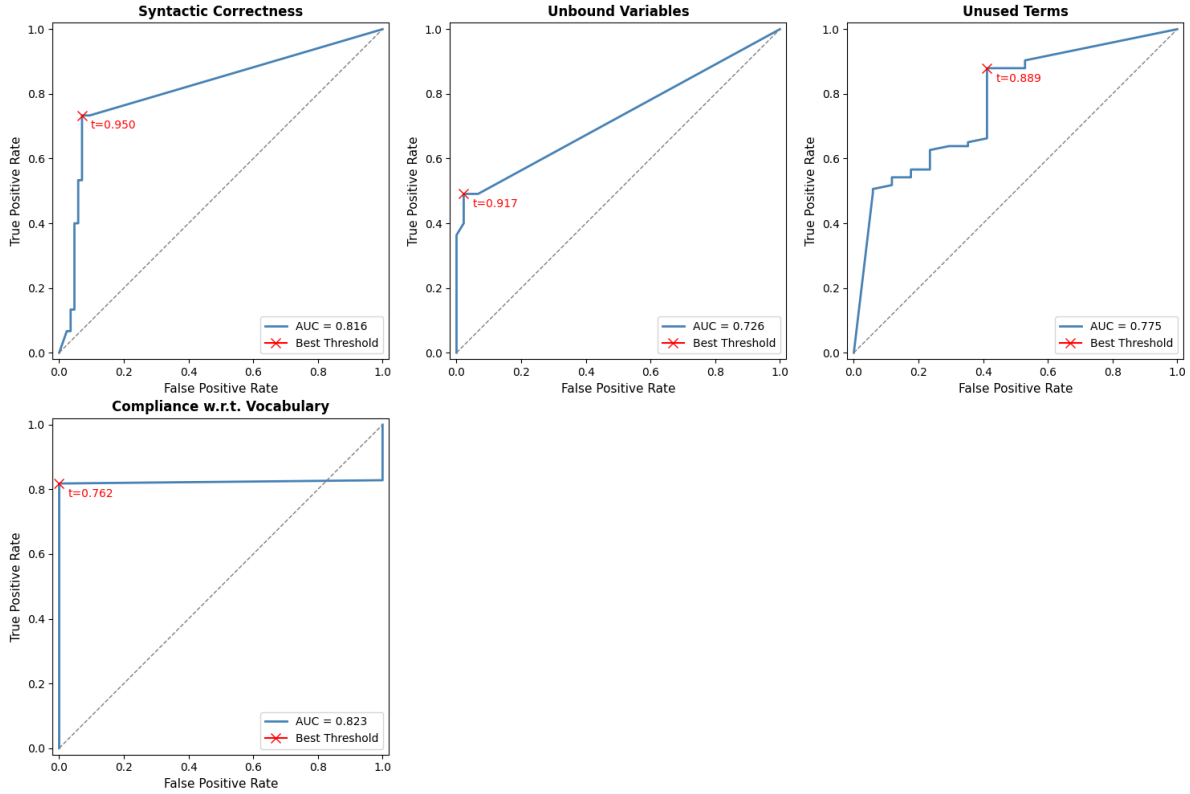

Figure 2: ROC curves for each metric, according to the threshold previously defined.

metric	N true	N false	N error	Percentage true
`KB Derivation Performance`	0	58	42	0

Table 5
Descriptive statistics for the `KB Derivation Performance` metric.

metric	N	mean	std	min	25%	50%	75%	max
`LRM Output Consistency` (only valid)	63	0.712	0.448	0	0	1	1	1
`LRM Output Consistency` (invalid=0)	100	0.448	0.495	0	0	0	1	1

Table 6
Descriptive statistics for the `LRM Output Consistency` metric. The columns min, 25%, 50%, 75% and max denote the minimum, first quartile, median, third quartile, and maximum values of the distribution, respectively.

which we assign a score of 0) and make the mean of the scores decrease to 0.448.

For the *KB Derivation Performance*, among the 42 Prolog programs classified as errors, 23 of them are inference errors (e.g. incorrect structures, unknown predicates, etc.) and the remaining 19 are Prolog programs so syntactically invalid that they could not be loaded in the SWI-Prolog interpreter for derivation. For the Prolog programs that were syntactically valid, 58 problems were deriving the wrong conclusions. No programs are deriving the appropriate conclusion. These results clearly show the need for a specialized LRM to generate knowledge.

5. Conclusion

In this paper, we presented a set of metrics to evaluate LRM-generated Prolog programs over several dimensions. We assessed their relevance by comparing their efficiency to detect problematic Prolog

programs to the ground truth that we labeled, assisted by an LLM. Three dimensions have been modeled by our metrics: *syntactical*, *structural*, and *task-related*.

As metrics are scoring a continuous value on the $[0, 1]$ interval, we determine the threshold for which the metric is scoring the best. We then calculated ROC curves to obtain the AUC score for each metric, indicating how well it is able to detect issues that were identified by a large LLM (Claude Sonnet 4.6). We observed that each metric is performing reasonably well with *Compliance w.r.t. Vocabulary* and *Syntactic Correctness* achieving scores of 0.823 and 0.816 respectively. However, our metrics are sometimes wrongly classifying programs as false positives. This happens when the metric is not designed sufficiently precisely. Further refinements on these metrics will surely improve the AUC scores we obtained in the experiment we presented. While our metrics could be improved, we learn that, based on issues detected by a state-of-the-art LLM, they are able to efficiently discriminate incorrect Prolog programs.

Regarding the task-based metrics, they provide an evaluation of how efficient the LRM is to generate Prolog programs that actually solve initial problems. While using those metrics shows that the generated Prolog programs are never deriving the appropriate conclusion, we will use these scores as a baseline to compare to future improvements. We assume that these improvements will notably be achieved through implementing in-context learning, fine-tuning through RL, and possibly Retrieval-Augmented Generation (RAG).

In this paper, we showed that, in the scope of automatically and incrementally building a KB with an LRM, these metrics represent a strong foundation for evaluating a single Prolog program over multiple dimensions.

6. Future Works

As future work, we need to extend this evaluation, passing from single programs to incremental KB construction. This involves testing the continuous classification performance of the large KB when integrating new problems to it, measuring the potential improved performance when the error raised by a metric is passed back to the LRM to trigger a new generation.

Moreover, based on the results we obtained, we might need to improve the capabilities of the LRMs used to generate the knowledge expressed as Prolog programs. A first possibility is to use in-context learning to enrich the context of the LRM in its prompt by giving one (one-shot) or more (few-shot) examples. We feel confident that fine-tuning through RL will be needed to improve the LRM. The design of our metrics – each returning a score belonging to $[0, 1]$ – already embraces this view, allowing us to create a reward score based on a composite metric.

Additionally, since we are constructing domain-specific KBs, we will assess the performance of LRMs to generate a full KB with enhanced context through techniques such as RAG [35].

Finally, in some domains, the required knowledge cannot be encoded with strict/crisps definitions through rules. Some problems might indeed require modeling knowledge with probabilistic estimations. We consider using ProbLog, an extension of Prolog that allows to specify a confidence scores associated to facts, while all facts described in pure Prolog being considered as strictly true.

Acknowledgments

This research is funded by an ANRT grant through the CIFRE system with Datanello as the industrial collaborator. The authors acknowledge Dr. Yacine Abboud and Dr. Benjamin Gras, directors at Datanello.

Declaration on Generative AI

During the preparation of this work, the author(s) used Gemini-3-Pro in order to: Grammar, spelling check, and style enhancement. After using these tool(s)/service(s), the author(s) reviewed and edited

the content as needed and take(s) full responsibility for the publication's content.

References

[1] M. Suwa, A. C. Scott, E. H. Shortliffe, An approach to verifying completeness and consistency in a rule-based expert system, Ai Magazine 3 (1982) 16–16.

[2] V. K. Kommineni, B. König-Ries, S. Samuel, From human experts to machines: An LLM supported approach to ontology and knowledge graph construction, 2024. doi:10.48550/arXiv.2403.08345. arXiv:2403.08345.

[3] H. Chen, X. Shen, Q. Lv, J. Wang, X. Ni, J. Ye, SAC-KG: Exploiting Large Language Models as Skilled Automatic Constructors for Domain Knowledge Graphs, 2024. doi:10.48550/arXiv.2410.02811. arXiv:2410.02811.

[4] DeepCogito, 2025. URL: https://www.deepcogito.com/research/cogito-v1-preview.

[5] DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, Z. F. Wu, Z. Gou, Z. Shao, Z. Li, Z. Gao, A. Liu, B. Xue, B. Wang, B. Wu, B. Feng, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Ding, H. Xin, H. Gao, H. Qu, H. Li, J. Guo, J. Li, J. Wang, J. Chen, J. Yuan, J. Qiu, J. Li, J. L. Cai, J. Ni, J. Liang, J. Chen, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Zhao, L. Wang, L. Zhang, L. Xu, L. Xia, M. Zhang, M. Zhang, M. Tang, M. Li, M. Wang, M. Li, N. Tian, P. Huang, P. Zhang, Q. Wang, Q. Chen, Q. Du, R. Ge, R. Zhang, R. Pan, R. Wang, R. J. Chen, R. L. Jin, R. Chen, S. Lu, S. Zhou, S. Chen, S. Ye, S. Wang, S. Yu, S. Zhou, S. Pan, S. S. Li, S. Zhou, S. Wu, S. Ye, T. Yun, T. Pei, T. Sun, T. Wang, W. Zeng, W. Zhao, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, W. L. Xiao, W. An, X. Liu, X. Wang, X. Chen, X. Nie, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yang, X. Li, X. Su, X. Lin, X. Q. Li, X. Jin, X. Shen, X. Chen, X. Sun, X. Wang, X. Song, X. Zhou, X. Wang, X. Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. Zhang, Y. Xu, Y. Li, Y. Zhao, Y. Sun, Y. Wang, Y. Yu, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Ou, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Xiong, Y. Luo, Y. You, Y. Liu, Y. Zhou, Y. X. Zhu, Y. Xu, Y. Huang, Y. Li, Y. Zheng, Y. Zhu, Y. Ma, Y. Tang, Y. Zha, Y. Yan, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Xie, Z. Zhang, Z. Hao, Z. Ma, Z. Yan, Z. Wu, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Pan, Z. Huang, Z. Xu, Z. Zhang, Z. Zhang, DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning, 2025. doi:10.48550/arXiv.2501.12948. arXiv:2501.12948.

[6] M. Haurel, Extracting problem-solving knowledge from LLMs with reasoning abilities, ISWC 2025 Companion Volume: Joint Proceedings of Industry, Doctoral Consortium, Posters and Demos of the 24th International Semantic Web Conference (ISWC-C 2025) (2025).

[7] N. Mihindukulasooriya, S. Tiwari, C. F. Enguix, K. Lata, Text2KGBench: A Benchmark for Ontology-Driven Knowledge Graph Generation from Text, 2023. doi:10.48550/arXiv.2308.02357. arXiv:2308.02357.

[8] Z. Cauter, N. Yakovets, Ontology-guided Knowledge Graph Construction from Maintenance Short Texts, in: Proceedings of the 1st Workshop on Knowledge Graphs and Large Language Models (KaLLM 2024), Association for Computational Linguistics, Bangkok, Thailand, 2024, pp. 75–84. doi:10.18653/v1/2024.kallm-1.8.

[9] J. Sun, S. Qian, Z. Han, W. Li, Z. Qian, D. Yang, J. Cao, G. Xue, LKD-KGC: Domain-Specific KG Construction via LLM-driven Knowledge Dependency Parsing, 2025. doi:10.48550/arXiv.2505.24163. arXiv:2505.24163.

[10] Y. Lairgi, L. Moncla, K. Benabdeslem, R. Cazabet, P. Cléau, ATOM: AdapTive and OptiMized dynamic temporal knowledge graph construction using LLMs, 2026. doi:10.48550/arXiv.2510.22590. arXiv:2510.22590.

[11] Y. Lairgi, L. Moncla, R. Cazabet, K. Benabdeslem, P. Cléau, iText2KG: Incremental Knowledge Graphs Construction Using Large Language Models, 2024. doi:10.48550/arXiv.2409.03284. arXiv:2409.03284.

[12] P. Rasmussen, P. Paliychuk, T. Beauvais, J. Ryan, D. Chalef, Zep: A Temporal Knowledge Graph Architecture for Agent Memory, 2025. doi:10.48550/arXiv.2501.13956. arXiv:2501.13956.
[13] S. Carta, A. Giuliani, L. Piano, A. S. Podda, L. Pompianu, S. G. Tiddia, Iterative Zero-Shot LLM Prompting for Knowledge Graph Construction, 2023. doi:10.48550/arXiv.2307.01128. arXiv:2307.01128.
[14] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. K. Li, Y. Wu, D. Guo, DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models, 2024. doi:10.48550/arXiv.2402.03300. arXiv:2402.03300.
[15] R. Brachman, H. Levesque, Knowledge Representation and Reasoning, Elsevier, 2004.
[16] A. Hogan, E. Blomqvist, M. Cochez, C. D'amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A.-C. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, A. Zimmermann, Knowledge Graphs, ACM Computing Surveys 54 (2022) 1–37. doi:10.1145/3447772.
[17] Y. Zhang, Y. Li, L. Cui, D. Cai, L. Liu, T. Fu, X. Huang, E. Zhao, Y. Zhang, Y. Chen, L. Wang, A. T. Luu, W. Bi, F. Shi, S. Shi, Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models, Computational Linguistics 51 (2025) 1373–1418. doi:10.1162/COLI.a.16.
[18] S. Ceri, G. Gottlob, L. Tanca, What you always wanted to know about Datalog (and never dared to ask), IEEE Transactions on Knowledge and Data Engineering 1 (1989) 146–166. doi:10.1109/69.43410.
[19] A. Colmerauer, P. Roussel, The birth of Prolog, in: T. J. Bergin, R. G. Gibson (Eds.), History of Programming Languages—II, ACM, New York, NY, USA, 1996, pp. 331–367. doi:10.1145/234286.1057820.
[20] K. Acharya, A. Velasquez, H. H. Song, A Survey on Symbolic Knowledge Distillation of Large Language Models, IEEE Transactions on Artificial Intelligence 5 (2024) 5928–5948. doi:10.1109/TAI.2024.3428519. arXiv:2408.10210.
[21] X. Yang, B. Chen, Y.-C. Tam, Arithmetic Reasoning with LLM: Prolog Generation & Permutation, 2024. doi:10.48550/arXiv.2405.17893. arXiv:2405.17893.
[22] N. Madani, R. K. Srihari, K. Joseph, Domain Specific Question Answering Over Knowledge Graphs Using Logical Programming and Large Language Models, 2023. doi:10.48550/arXiv.2303.02206. arXiv:2303.02206.
[23] C. Saetia, J. Phruetthiset, T. Chalothorn, M. Lertsutthiwong, S. Taerungruang, P. Buabthong, Financial Product Ontology Population with Large Language Models, in: Proceedings of TextGraphs-17: Graph-based Methods for Natural Language Processing, Association for Computational Linguistics, Bangkok, Thailand, 2024, pp. 53–60. doi:10.18653/v1/2024.textgraphs-1.4.
[24] Y. Guo, Z. Pan, J. Heflin, An Evaluation of Knowledge Base Systems for Large OWL Datasets, Proceedings of the 3rd International Semantic Web Conference (ISWC) (2004).
[25] J. Boylan, S. Mangla, D. Thorn, D. G. Ghalandari, P. Ghaffari, C. Hokamp, KGValidator: A Framework for Automatic Validation of Knowledge Graph Construction, 2024. doi:10.48550/arXiv.2404.15923. arXiv:2404.15923.
[26] T. J. Murray, M. R. Tanniru, Control of inconsistency and redundancy in PROLOG-type knowledge bases, Expert Systems with Applications 2 (1991) 321–331. doi:10.1016/0957-4174(91)90038-G.
[27] J. Plu, O. M. Escobar, E. Trouillez, A. Gapin, R. Troncy, A Comprehensive Benchmark for Evaluating LLM-Generated Ontologies, Proceedings of the ISWC 2024 Special Session on Harmonising Generative AI and Semantic Web Technologies (2024).
[28] M. Llugiqi, F. J. Ekaputra, M. Sabou, From Experts to LLMs: Evaluating the Quality of Automatically Generated Ontologies, 2nd International Workshop on Evaluation of Language Models in Knowledge Engineering Co-located with the Extended Semantic Web Conference (ESWC 2025) (2025).
[29] S. Tsaneva, D. Dessì, F. Osborne, M. Sabou, Knowledge graph validation by integrating LLMs and human-in-the-loop, Information Processing & Management 62 (2025) 104145. doi:10.1016/j.ipm.2025.104145.

[30] G. Hannah, J. de Berardinis, T. R. Payne, V. Tamma, A. Mitchell, E. Piercy, E. Johnson, A. Ng, H. Rostron, B. Konev, Large Language Models as Knowledge Evaluation Agents, 2nd International Workshop on Evaluation of Language Models in Knowledge Engineering Co-located with the Extended Semantic Web Conference (ESWC 2025) (2025).

[31] Z. Di, C. Zhang, H. Lv, L. Cui, L. Liu, LoRP: LLM-based Logical Reasoning via Prolog, Knowledge-Based Systems 327 (2025) 114140. doi:10.1016/j.knosys.2025.114140.

[32] J. Wielemaker, T. Schrijvers, M. Triska, T. Lager, SWI-Prolog, Theory and Practice of Logic Programming 12 (2012) 67–96. doi:10.1017/S1471068411000494.

[33] OpenAI, A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. J. Ostrow, A. Welihinda, A. Hayes, A. Radford, A. Mądry, A. Baker-Whitcomb, A. Beutel, A. Borzunov, A. Carney, A. Chow, A. Kirillov, A. Nichol, A. Paino, A. Renzin, A. T. Passos, A. Kirillov, A. Christakis, A. Conneau, A. Kamali, A. Jabri, A. Moyer, A. Tam, A. Crookes, A. Tootoochian, A. Tootoonchian, A. Kumar, A. Vallone, A. Karpathy, A. Braunstein, A. Cann, A. Codispoti, A. Galu, A. Kondrich, A. Tulloch, A. Mishchenko, A. Baek, A. Jiang, A. Pelisse, A. Woodford, A. Gosalia, A. Dhar, A. Pantuliano, A. Nayak, A. Oliver, B. Zoph, B. Ghorbani, B. Leimberger, B. Rossen, B. Sokolowsky, B. Wang, B. Zweig, B. Hoover, B. Samic, B. McGrew, B. Spero, B. Giertler, B. Cheng, B. Lightcap, B. Walkin, B. Quinn, B. Guarraci, B. Hsu, B. Kellogg, B. Eastman, C. Lugaresi, C. Wainwright, C. Bassin, C. Hudson, C. Chu, C. Nelson, C. Li, C. J. Shern, C. Conger, C. Barette, C. Voss, C. Ding, C. Lu, C. Zhang, C. Beaumont, C. Hallacy, C. Koch, C. Gibson, C. Kim, C. Choi, C. McLeavey, C. Hesse, C. Fischer, C. Winter, C. Czarnecki, C. Jarvis, C. Wei, C. Koumouzelis, D. Sherburn, D. Kappler, D. Levin, D. Levy, D. Carr, D. Farhi, D. Mely, D. Robinson, D. Sasaki, D. Jin, D. Valladares, D. Tsipras, D. Li, D. P. Nguyen, D. Findlay, E. Oiwoh, E. Wong, E. Asdar, E. Proehl, E. Yang, E. Antonow, E. Kramer, E. Peterson, E. Sigler, E. Wallace, E. Brevdo, E. Mays, F. Khorasani, F. P. Such, F. Raso, F. Zhang, F. von Lohmann, F. Sulit, G. Goh, G. Oden, G. Salmon, G. Starace, G. Brockman, H. Salman, H. Bao, H. Hu, H. Wong, H. Wang, H. Schmidt, H. Whitney, H. Jun, H. Kirchner, H. P. d. O. Pinto, H. Ren, H. Chang, H. W. Chung, I. Kivlichan, I. O'Connell, I. O'Connell, I. Osband, I. Silber, I. Sohl, I. Okuyucu, I. Lan, I. Kostrikov, I. Sutskever, I. Kanitscheider, I. Gulrajani, J. Coxon, J. Menick, J. Pachocki, J. Aung, J. Betker, J. Crooks, J. Lennon, J. Kiros, J. Leike, J. Park, J. Kwon, J. Phang, J. Teplitz, J. Wei, J. Wolfe, J. Chen, J. Harris, J. Varavva, J. G. Lee, J. Shieh, J. Lin, J. Yu, J. Weng, J. Tang, J. Yu, J. Jang, J. Q. Candela, J. Beutler, J. Landers, J. Parish, J. Heidecke, J. Schulman, J. Lachman, J. McKay, J. Uesato, J. Ward, J. W. Kim, J. Huizinga, J. Sitkin, J. Kraaijeveld, J. Gross, J. Kaplan, J. Snyder, J. Achiam, J. Jiao, J. Lee, J. Zhuang, J. Harriman, K. Fricke, K. Hayashi, K. Singhal, K. Shi, K. Karthik, K. Wood, K. Rimbach, K. Hsu, K. Nguyen, K. Gu-Lemberg, K. Button, K. Liu, K. Howe, K. Muthukumar, K. Luther, L. Ahmad, L. Kai, L. Itow, L. Workman, L. Pathak, L. Chen, L. Jing, L. Guy, L. Fedus, L. Zhou, L. Mamitsuka, L. Weng, L. McCallum, L. Held, L. Ouyang, L. Feuvrier, L. Zhang, L. Kondraciuk, L. Kaiser, L. Hewitt, L. Metz, L. Doshi, M. Aflak, M. Simens, M. Boyd, M. Thompson, M. Dukhan, M. Chen, M. Gray, M. Hudnall, M. Zhang, M. Aljubeh, M. Litwin, M. Zeng, M. Johnson, M. Shetty, M. Gupta, M. Shah, M. Yatbaz, M. J. Yang, M. Zhong, M. Glaese, M. Chen, M. Janner, M. Lampe, M. Petrov, M. Wu, M. Wang, M. Fradin, M. Pokrass, M. Castro, M. O. T. de Castro, M. Pavlov, M. Brundage, M. Wang, M. Khan, M. Murati, M. Bavarian, M. Lin, M. Yesildal, N. Soto, N. Gimelshein, N. Cone, N. Staudacher, N. Summers, N. LaFontaine, N. Chowdhury, N. Ryder, N. Stathas, N. Turley, N. Tezak, N. Felix, N. Kudige, N. Keskar, N. Deutsch, N. Bundick, N. Puckett, O. Nachum, O. Okelola, O. Boiko, O. Murk, O. Jaffe, O. Watkins, O. Godement, O. Campbell-Moore, P. Chao, P. McMillan, P. Belov, P. Su, P. Bak, P. Bakkum, P. Deng, P. Dolan, P. Hoeschele, P. Welinder, P. Tillet, P. Pronin, P. Tillet, P. Dhariwal, Q. Yuan, R. Dias, R. Lim, R. Arora, R. Troll, R. Lin, R. G. Lopes, R. Puri, R. Miyara, R. Leike, R. Gaubert, R. Zamani, R. Wang, R. Donnelly, R. Honsby, R. Smith, R. Sahai, R. Ramchandani, R. Huet, R. Carmichael, R. Zellers, R. Chen, R. Chen, R. Nigmatullin, R. Cheu, S. Jain, S. Altman, S. Schoenholz, S. Toizer, S. Miserendino, S. Agarwal, S. Culver, S. Ethersmith, S. Gray, S. Grove, S. Metzger, S. Hermani, S. Jain, S. Zhao, S. Wu, S. Jomoto, S. Wu, Shuaiqi, Xia, S. Phene, S. Papay, S. Narayanan, S. Coffey, S. Lee, S. Hall, S. Balaji, T. Broda, T. Stramer, T. Xu, T. Gogineni, T. Christianson, T. Sanders, T. Patwardhan, T. Cunninghman, T. Degry, T. Dimson, T. Raoux, T. Shadwell, T. Zheng, T. Underwood, T. Markov, T. Sherbakov,

T. Rubin, T. Stasi, T. Kaftan, T. Heywood, T. Peterson, T. Walters, T. Eloundou, V. Qi, V. Moeller, V. Monaco, V. Kuo, V. Fomenko, W. Chang, W. Zheng, W. Zhou, W. Manassra, W. Sheu, W. Zaremba, Y. Patil, Y. Qian, Y. Kim, Y. Cheng, Y. Zhang, Y. He, Y. Zhang, Y. Jin, Y. Dai, Y. Malkov, GPT-4o System Card, 2024. doi:10.48550/arXiv.2410.21276. arXiv:2410.21276.

[34] J. Chen, Z. Cai, K. Ji, X. Wang, W. Liu, R. Wang, J. Hou, B. Wang, Huatuogpt-o1, towards medical complex reasoning with llms, 2024. URL: https://arxiv.org/abs/2412.18925. arXiv:2412.18925.

[35] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, H. Wang, Retrieval-Augmented Generation for Large Language Models: A Survey, 2024. doi:10.48550/arXiv.2312.10997. arXiv:2312.10997.