Reasoning with LLMs aided by Knowledge Bases and by Context

Leonardo Riccioppo Garcez, Fabio Gagliardi Cozman

Departamento de Engenharia da Computação – Escola Politécnica da Universidade de São Paulo (USP) – São Paulo, SP - Brazil

lrgarcez@alumni.usp.br, fgcozman@usp.br

Abstract. *Recent Large Language Models can display surprising reasoning abilities through Chain-of-Thought and similar techniques. However, their difficulty with hallucinations, and the opaque nature of their internal operations, are notable drawbacks. This work proposes an approach for reasoning that employs a symbolic logic solver aided by LLMs. We mix symbolic reasoning with LLMs as presented in previous work, and we improve their reasoning skills by searching for logical statements with path context within a knowledge base.*

Resumo. *Grandes Modelos de Linguagem (GML) podem exibir habilidades de raciocínio surpreendentes através da Chain-of-Thought e técnicas similares. Entretanto, sua dificuldade com alucinações, e a natureza opaca das suas operações internas, são desvantagens importantes. Este trabalho apresenta uma proposta para o raciocínio que emprega um solver lógico auxiliado por GMLs. Nós misturamos raciocínio simbólico com GMLs como apresentado em trabalhos anteriores, e melhoramos as habilidades de raciocínio com a busca por predicados lógicos com o contexto do fluxo lógico de uma base de conhecimento.*

1. Introduction

In recent years, Large Language Models (LLMs) have changed the field of Natural Language Processing (NLP). The transformer architecture, particularly its multi-head attention mechanism (Vaswani, et al., 2017), has substantially improved performance across various NLP tasks compared to previous techniques. However, reasoning abilities are, for the most part, still rudimentary in LLMs. These models emulate reasoning through complex vector and matrix transformations, non-linear functions, and pattern recognition, resulting in outputs that are often difficult to trace and interpret.

However, some recent LLMs trained on vast amounts of natural language data have actually demonstrated impressive reasoning-like abilities (Wei, et al., 2023), (Zhou, et al., 2023). As the number of logical steps increases, their likelihood of generating incorrect or inaccurate conclusions also rises. In any case, although prompting methods exist to insert guardrails on their responses, their inherent limitations remain.

In response to these challenges, research laboratories and companies, such as OpenAI and DeepSeek, are developing LLMs with enhanced reasoning-like capabilities. Their focus is on creating models that can solve problems step-by-step, effectively "thinking" through tasks. OpenAI's GPT-4o-mini, for instance, incorporates reinforcement learning in its training process (OpenAI, 2025). Generally, these models undergo extensive pre-training on massive NLP text corpora to assimilate linguistic rules, concepts, and syntactic and semantic features. Subsequently, they undergo supervised fine-tuning to improve accuracy and reliability, while also implementing guardrails based on established policies. For instance, the DeepSeek-R1 model (DeepSeek-AI, 2025)) starts with unsupervised pre-training, but it moves, in the post-processing phase, to Reinforcement Learning through the Group Relative Policy Optimization (GRPO) algorithm. This optimization algorithm uses a sample group of outputs from the previous policy, a list of rewards, and various hyperparameters to maximize an objective function, thereby identifying the optimal new policy for the given state. The algorithm generates "thinking steps" within a section of the response, adhering to the patterns of Chain-of-Thought (CoT) prompting found in the fine-tuning datasets.

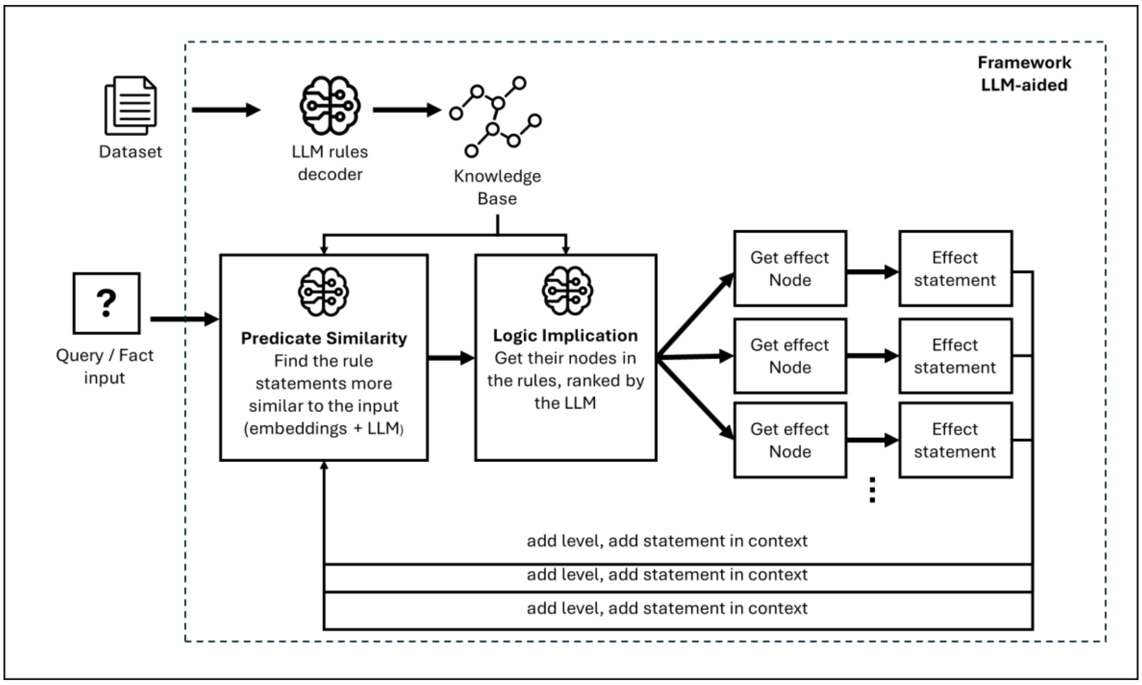
Framework LLM-aided
Dataset
LLM rules decoder
Knowledge Base
Query / Fact input
Predicate Similarity Find the rule statements more similar to the input (embeddings + LLM)
Logic Implication Get their nodes in the rules, ranked by the LLM
Get effect Node
Effect statement
Get effect Node
Effect statement
Get effect Node
Effect statement
add level, add statement in context
add level, add statement in context
add level, add statement in context

Figure 1. Framework functionality flow.

Given the complexity and analytical challenges of previous proposals, we propose a framework designed to reason through simple proofs in First Order Logic. Our model aims to construct sequences of facts so as to answer questions, by building and processing a Knowledge Base (KB) composed of logic predicates and rules. Leveraging the reasoning capabilities of LLMs, the model selects appropriate predicates and rules to follow. The framework can process a logical chain using the context and a LLM to decide each next step of the logic flow, and resorting to the KB built from the full dataset. **Figure** 1 shows the structure of our proposal.

First, we use LLM prompting to build a KB with rules and statements, using a training and a testing dataset. After building the KB, we apply logical techniques and LLMs to do reasoning whenever necessary.

We apply the approach described by (Toroghi, Guo, Pesaranghader, & Sanner, 2024), using embeddings and LLMs to rank the next statement in a logical reasoning chain, and to connect rules. However, we improve on their proposal by processing the graph's entire logic path in the LLM as context – not just the current statement. In the end, we find a similar statement and find the rules having that statement as the cause, processing in the next iteration the effect statements of those rules. This process is repeated a predefined number of times and the identification of success in the logic reasoning is finding the path with major rank that has the target effect statement. We get the target effect statement by applying the first similarity process in an apart processing.

We present key references in Section 2. The proposal is described in Section 3, and the test results using the COPA-SSE dataset are reported in Section 4. Final conclusions and future work are discussed in Section 5.

2. Related Work

Logical techniques are not a prevalent approach today for question-answering: LLMs have shifted focus towards enhancing the performance of systems utilizing Deep Neural Networks and Transformer architectures. Nonetheless, ongoing research does explore the integration of LLM features with elements of logical reasoning.

One notable approach is the Chain-of-Thought (CoT) method (Wei, et al., 2023) which sequences the LLM input problem into step-by-step solutions. CoT prompts are constructed with few-shot examples of questions and their sequential answers, enabling the LLM "to learn" from these examples and follow the abstract problem-solving pattern. Results from their test datasets demonstrate significant improvements in LLMs' reasoning capabilities. However, LLMs used for the task have required more than 100 billion parameters. For instance, errors observed with the PaLM 62B model were rectified by scaling to PaLM 540B.

The work of (Zhou, et al., 2023) presents an approach like CoT, known as Least-to-Most, derived from educational psychology. This method breaks problems into sub-questions and orders them progressively, from simpler to more complex. Their work primarily used the GPT-3 code-davinci-002 model. Although the authors assert that Chain-of-Thought (CoT) prompting is distinct from Least-to-Most prompting, their methodologies appear quite similar in several instances. Furthermore, the paper by (Zhou, et al., 2023) concludes by highlighting errors associated with Least-to-Most prompting. Specifically, errors were identified in sections involving basic mathematical calculations and text comprehension. These errors were predominantly observed during the decomposition step, where sub-questions generated were often nonsensical, leading to incorrect answers. The GPT-3 code-davinci-002 model used is a relatively old version of the GPT model, and it is likely that current models like GPT-4.5 would achieve better results.

Other works aim to reduce reliance on LLMs for reasoning by coupling text-based reasoning with LLMs. One such approach is the backward chaining algorithm LAMBADA, presented by (Kazemi, Kim, Bhatia, & Xu, 2023). This algorithm proposes a method to find proof of a logic chain starting from a goal and given a set of rules and facts. The algorithm comprises four modules: Fact Check, Rule Selection, Goal Decomposition, and Sign Agreement.

- **Fact Check** verifies if there is a fact in a set of facts that entails the defined goal or its negation.

- **Rule Selection** identifies the consequent of each rule and checks which one best unifies with the goal.
- **Goal Decomposition** identifies the sub-goals needed to be proved or disproved based on the antecedent of a rule that has the selected goal as its consequent. The rules are formatted as "sub-goals entail goal" and the sub-goals are decomposed in a recurrent process of backward chaining with a maximum depth set at the beginning of the algorithm.
- **Sign Agreement verifies** if the proved or disproved equivalent goal by the rules agrees with the sign of the given goal.

The tests of LAMBADA were conducted using the ProofWriter dataset (Tafjord, Mishra, & Clark, 2021), a common dataset for reasoning tests with synthetic rules and facts in naturalistic texts. LAMBADA utilized the Open-World Assumption subset (OWA). Additionally, it employed PrOntoQA (Saparov & He, 2023), a synthetic dataset for LM-based reasoning, and ParaRules (Tafjord, Mishra, & Clark, 2021), a variation of ProofWriter where synthetically generated sentences are rewritten by crowdworkers to enhance naturalness and diversity. The model used in LAMBADA is predominantly PaLM 540B, a LLM trained with a focus on few-shot learning. All prompts explained in the article's attachments follow the few-shot reasoning approach, feeding the prompt with examples of facts and rules selection from their lists, and examples of breaking NLP rule sentences into statements to be proven in the recursive chaining process. There is considerable uncertainty when the LLM makes these interpretations, as it can hallucinate in precision reasoning tasks and when finding the best related item in a long list. The results are compared with the Chain-of-Thought (CoT) approach (Wei, et al., 2023) and Selection-Inference (SI) (Creswell, Shanahan, & Higgins, 2023). CoT uses prompts to break the logic chain (in this case, the proof) that the LLM needs to build based on the facts, rules, and goal given inside the prompt, with few-shot examples for the LLM's "learning." SI is a forward chaining reasoning approach that begins with a goal, selecting facts and rules to find the conclusion and the next goal, iterating the process until a given number of iterations as a threshold. In all the article's presented tests, LAMBADA's results surpass CoT and SI in all datasets used.

Forward chaining faces the problem of an increasing search space of statements. As the depth increases, the number of possible combinations of statement links with the rules also increases. In our work, we address these multiple combinations with probabilities for each link to find the chaining route with the highest product of probabilities. While the LAMBADA article reinforces the efficiency of backward chaining, we will implement the forward chaining approach because we believe it mirrors human reasoning more closely.

Building upon previous research, (Lee & Hwang, 2024) introduced an advanced backward chaining framework that leverages a large language model (LLM) to generate symbolic rules and a symbolic solver for reasoning tasks. This framework, termed Symba (Symbolic Backward Chaining), features a solver that governs the reasoning process, with the LLM intervening only when the solver is unable to validate a subgoal. The results of the proposed framework are compared with other natural language-based structured chaining methods such as LLM-based Chain-of-thought (Wei, et al., 2023), LLM based Least-to-most prompting (Zhou, et al., 2023) and LAMBADA (Kazemi, Kim, Bhatia, & Xu, 2023).

Symba does have limitations. The precision required in creating statements using LLMs makes the framework susceptible to hallucinations and variations in the generated answers, as it relies on exact text matching of statements. Furthermore, the format of the logic rules and statements used restricts the representation of diverse problems in logic programming. Lastly, there is a limitation related to the small context used. When numerous logic rules and statements are present, it can connect a statement to a rule head that transfers the context to an undesired subgoal, as the specific context for each case is feasible in controlled test examples. Multiple contexts in the same NLP text can introduce noise into the reasoning chaining.

Another significant and recent research work integrating LLMs with logic programming is the framework LLM-TRes (Toroghi, Guo, Pesaranghader, & Sanner, 2024). Based on Theory Resolution, this framework utilizes the LLM for two tasks: i) a semantic parser that converts natural language into logic statements, similar to Symba framework; ii) an axiom repairer that uses its "knowledge" to generate a corrective logic axiom not present in the logic context database. The LLM-TRes framework employs two strategies to link statements/predicates: Resolution Priority Definition and Ordering, and Restricting Theory Resolution with Embeddings.

Resolution Priority Definition and Ordering involves using the LLM to calculate the probability of one statement entailing another, as represented by the formula:

$$\rho_{c_{res}}^{entail} = p\left(c_{target} \vdash_{LLM} c\right), \tag{1}$$

Where c_{target} is the initial clause that will result in the c_{res} clause being the biggest one tested in the *c* clause in the equation with the plausible score $\rho_{c_{res}}^{entail}$. In case the focus of this framework application is finding the response of a logic chain, to find the overall entailment proof of the chain, it is defined as the multiplication of $\rho_{c_{res}}^{entail}$ and all c_{res} parents plausible scores. Furthermore, the second priority score is defined by the max depth level of the response clause c_{res} that is used to handle the cases of ties between clauses that have equal plausible scores. This depth score, for each clause in the logic chain, is given by:

$$\rho_{c_{res}}^{l} = 1 + max\left(\rho^{l}(c')\right), \tag{2}$$

Where $\rho^{l}(c')$ is the depth level score of each parent clause before c_{res}. So, the final proof score of c_{res} will be given by a tuple ($\rho_{c_{res}}^{entail}$, $\rho_{c_{res}}^{l}$). The authors affirm that the best logic chain proof will be achieved with the smallest depth score $\rho^{l}(c)$ to avoid redundancy.

Restricting Theory Resolution with Embeddings is the process of ordering all possible clauses of the Knowledge Base before applying the *Resolution Priority Definition and Ordering.* For each selected clause, they calculate the inner product of their embedding vectors, ordering by ascending and selecting the first *b* clauses of this ordered list, where *b* is a threshold quantity of clauses.

The framework LLM-TRes is implemented in 3 kinds of tasks:

Preference Reasoning: Recommendation considering context in natural language. They used the dataset Recipe-MPR with 500 user queries to select a recipe.

Deductive Reasoning: They use ProntoQA (Saparov & He, 2023) as presented in previous cited studies. It contains 500 queries in natural language on Knowledge Bases composed by axioms and facts, in this case, the dataset is used for common sense reasoning.

Causal Commonsense Reasoning: Here they use COPA-SSE (Brassard, Heinzerling, Kavumba, & Inui, 2022), a semi structured Knowledge Base about event causes and effects. The NL query of a cause or an effect has an explanation in NL sequence of rules and facts with an assigned quality score representing the level of explanation of the NL sentences.

The framework is compared with results of LAMBADA (Kazemi, Kim, Bhatia, & Xu, 2023) with GPT-3.5 Turbo as the LLM (this model is the one used also in the decoding of natural language rules to clausal form), zero-shot Chain-of-Thought (Jin, Liu, & Tan, 2024) and for executing the entailment calculations it is used the model BART finetuned with MNLI (Williams, Nangia, & Bowman, 2018). The tests with zero-shot Chain-of-Thought were made using prompts in GPT-3.5 Turbo, Llama3 8B, Mistral 7B, and Gemma 7B. They also made tests with pure BART-large entailment scores between facts and query as a baseline to show that the difference of performance was not provided only by the LLM used.

LLM-TRes has considerable performance in all dataset tests. Even CoT with GPT-3.5 being better than the framework in Recipe-MPR, it is outperformed by Mistral-7B in ProntoQA dataset and has approximate results in COPA-SSE dataset.

It remains unclear whether the knowledge base assumed for each dataset item is confined to the statements and rules within the item's context or if it encompasses a global list. The examples provided in the referenced article did not address the ordering of statements to limit the number of tests via embedding vector inner products. Consequently, we will assume that each knowledge base is restricted to the context of individual dataset items. Another open question is the extent to which the prompt engineering presented to convert the natural language sentences in the dataset into logic rules was really successful and did not require manual corrections.

3. A Proposal: LLMs for Logical Reasoning using Knowledge Bases

As previously noted, our goal is to develop a framework that processes potential proofs employing forward chaining, aiming to simulate simple human reasoning using its knowledge base. We wish to execute this process in the most automatic way possible, letting the system learn the rules and to find the correct logic path. Another change from previous proposals is the use of LLM to rank probabilities of the statements in the Predicate Similarity step and in Logic Similarity step. The rank probability of the next predicate given by the LLM should be better defined if the LLM receives the context of the current logic path as input.

So, this work proposes a framework that builds a Knowledge Base (KB) using LLM from the whole dataset and executes reasoning tasks using the KB, the context logic path and a LLM to execute the reasoning forward chain. We divided the proposal into a few blocks as follows:

1. Dataset Processing: using COPA-SSE to process part of the test dataset;
2. Decoder: using LLM prompt engineering to convert sentences in sequence of rules;
3. Knowledge Graph: List of predicates and rules formed by those predicates;

4. Solver with LLM: process reasoning through the rules graph starting from the query statement and trying to find the correct target statement for each case. Each association comes with a probability of the path in the logic chain.

We now explain in more detail each one of these tasks. We start with the prompt (in tests applied to GPT4o mini) that we use to extract rules and predicates from text to create a Knowledge Base:

Convert the following text to First Order Logic clausal sentences using entailment like "predicate B (x) :- predicate A (x)". if any predicate has more than one word, use "_" and not spaces. Also do not join the words, use "_" to separate them. Return as response only a list of rules in ASP language separed by ;. Follow the examples below:

1) Text: The child disobeyed her parents. Her parents punished her. Her parents hugged her. Disobeying parents causes punishment.

Rules and predicates: disobeyed(child, parents); hugged(parents,child); punish(parents, X):-disobeying(child, parents)

2) Text: The man felt cold while reading the newspaper. He chuckled. He sipped coffee. Sipping coffee is motivated by feeling cold. The man desires to be warm.

Rules and predicates: feeling_cold(man); reading_newspaper(man); chuckled(man); motivates_sipping_coffee(man):-feeling_cold(man); get_warmed(man):-sipping_coffee(man); desire_to_be_warm(man):-feeling_cold(man)

3) …

Text: [Sentences]

After the results of all LLM requests are gathered we have an algorithm that manages all the predicates and rules from the result in two json files: one with all enumerated predicates generated and other enumerating all rules generated using those predicates. All predicates are considered true.

Creating the Knowledge Graph for the framework is a process aided by LLMs. As it is very difficult to craft a large KB with all the connection predicates in the rules exactly equal, we propose to use LLMs to create the rules with the predicates.

The first approach we pursued was using a LLM to break the given sentences in NLP to rules and predicates of FOL. The LLM used was GPT 4o-mini, with the prompt shown above, executing one LLM call by sentence case in COPA-SSE dev and test datasets. It is important to notice that we have considered just the row with the highest "mean-rating" for each question. In the table below we show some examples of the generated input sentences and the rules.

Table 1. Examples of getting rules from sentences.

Explain Sentence	LLM Output Rules	e-id
The bowling ball knocked over the bowling pins. Ball causes inertia. Inertia causes energy transfer. Energy transfer causes pins knocked down. So the man rolled the bowling ball down the alley.	inertia(Ball):-Ball; energy_transfer(Inertia):-inertia(Inertia); pins_knocked_down(Energy_transfer):-energy_transfer(Energy_transfer); rolled_down_alley(man,bowling_ball):-man,bowling_ball	q411_e6

An awful song came on the radio. Awful song causes the desire to cover ears. So I covered my ears.	desire_to_cover_ears(X):-awful_song(X); covered_ears(Y):-desire_to_cover_ears(Y)	q1042_e1
The boaters caught a nice current. The boat is capable of drifting. The nice current causes the boat to be drifted. So their boat drifted to shore.	boat_drift(x):-boat(x),capable_drifting(x); capable_drifting(x):-boat(x); nice_current_causes_drift(x):-nice_current,boat(x); drifted_to_shore(x):-nice_current_causes_drift(x)	q1063_e2

We now describe the Logic Solver, composed of the two principal functions of the recursive part: Predicate Similarity and Logic Implication.

Given a KB, we need to navigate through the graph to connect the rules. As mentioned in the session presenting the LLM-Res and SymBa, it is more common to find a sequence of rules with the predicates that link them not totally equal. If we get a large quantity of rules generated from sentences and texts, it is very difficult to craft all the relations between the predicates with the rules, having identical predicates in different rules. So, to solve that logic chain we try to create a more automatic and statistical model to simulate simple reasoning using the logic rule chain.

To build the Logic Rule Chain, we need to state some definitions:

Logic Rule: is a true entailment, between one or more predicates in the way (predA, predB,…) entails (pred1, pred2, ….) wrote in ASP language as:

$$pred1, pred2, \ldots :- predA, predB, \ldots \tag{3}$$

Predicate: is a logic true condition or affirmation, It is composed by a sequence of words joined by underlines with input parameters that can be generic variables (Predicate_A(X)) or other predicates (Predicate_B(Y, Predicate_C(Z))).

We create an algorithm that uses LLM and embedding vectors to relate predicates between rules. For each predicate we do the following steps in a recursive way: Predicate Similarity and Logic Implication.

In the end, start the process again for each final predicate listed. In both steps we need to evaluate the predicates in the knowledge list considering the rule nodes that can have them. We called the process **Context Proximity Evaluation**.

Predicate Similarity

The process of Predicate Similarity gets as input a predicate and finds the most similar predicates in the full list of the knowledge base. The predicates are firstly ordered by the cosine distance between its embedding vector and the input embedding vector. We get the 500 more similar predicates as a selected buffer. So, this selected list is used as label input in the BART MNLI model with the input predicate as sentence input in the model. This model returns ordered values from 0 to 1 for each one of the 500 labels. Those score values are assumed as the probability of the connection between the input node and the new one, by the link of both predicates. We get the first 5 predicates and search nodes from rules that contain them. We list the possible nodes and calculate the

BART MNLI probability again, but this time we consider all the predicates of the selected nodes and the context of the logic flow as inputs. **Figure** 2 shows the link between nodes.

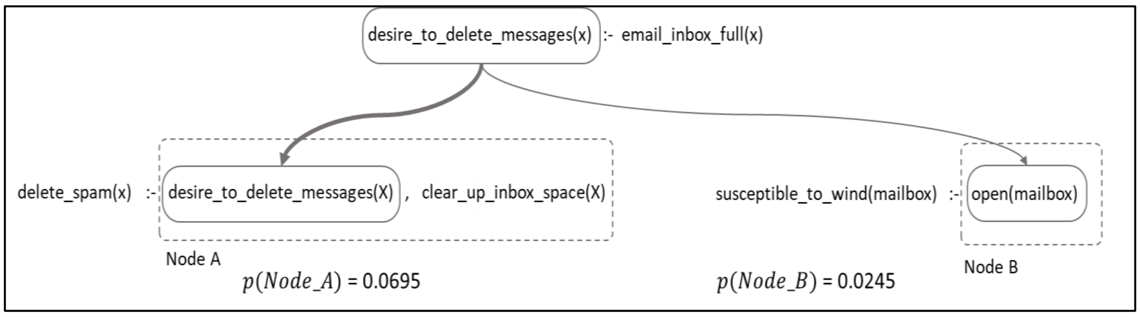
desire_to_delete_messages(x) :- email_inbox_full(x)
delete_spam(x) :- desire_to_delete_messages(X) , clear_up_inbox_space(X)
susceptible_to_wind(mailbox) :- open(mailbox)
Node A
$p(Node\_A)$ = 0.0695
$p(Node\_B)$ = 0.0245
Node B

Figure 2. Example of probabilities in connection nodes given the context

In the case above there are two contexts that the chaining can proceed, one is with the physical mailbox, and another is with the virtual e-mail that has the bigger probability of connection, given by the similarity of the predicate and by the context alignment of other statements inside the same node.

After ordering them, we get the first 5 nodes. In this case the BART MNLI probability is calculated for each node alone added the context. The score for each node will return as the score for this new level of reasoning.

Logical Implication

This process seems like the Similarity Process, however, in this one we do not make the search for similar predicates. So, given the predicate, we search the rules that have it inside the cause node. After listing them, apply the Context Proximity Evaluation to define the score for each node. We order the nodes by the score and get the first 5 ones.

As our final step, we present a commented recursive procedure that looks for the next rule to apply:

```
function process_foward_chain_recursive(predicate, query, parent_node, context)
   if parent_node.level + 1 is bigger than the LEVEL_THRESHOLD   return null
   # -- Predicate Similarity --
   if predicate text is equal query text: # find the predicate more similar to the NLP query
       Nodes_with_found_predicate, score_of_node = FIRST_finding_similar_predicates(predicate) # here we get the one predicate with best score given by embeddings inner product and by LLM selection
       Append the nodes in Nodes_with_found_predicate as parent_node's child
   else
      Nodes_with_found_predicate, nodes_scores = find_similar_predicates(predicate, context)
      Append the nodes in Nodes_with_found_predicate as parent_node's children
   # -- Logic Implication --
   for each node in Nodes_with_found_predicate
      for each p in node predicates
         Nodes_implication, scores_implication = process_logic_implication(predicate, node, context) # here in this method we search all the possible effect predicates in all rules and get the implication score by letting the LLM indicate the best logic implication based in the context
```

```
    Append nodes in Nodes_implication as  node's children
# -- Process the same function recursively --
for each node in Nodes_with_found_predicate
  for each nod in node.children
    for each p in nod.predicates
      update_context() # insert the logic path to achieve predicate p in the context: node->nod
      process_foward_chain_recursive(p, query, node, context)
```

4. Results

Forward chain reasoning linking and ranking logic rules is known to be an exponential problem, as the number of levels ahead increases. Thus, we limited our tests to 5 levels of depth. Also, to facilitate the analysis, we used only the 100 first cases of COPA-SSE "effect" type test cases. The analysis of the results must be done manually because there are details like the statement representing the target and the query, the rules decoded by the LLMs that are difficult to get by automatic methods. Our framework achieved 45 cases with the correct target and 55 cases with errors, however, most errors were due to lack of correct rules from the LLM decoding process, as **Table** 1 shows the availability of correct rules in the test set. One interesting point is that even without the correct rules getting decoded by the LLM, our framework found the correct target in 4 cases because it had the query and the target statements in the Knowledge Base, hence the Embeddings + LLM indicated their implication as a probable one.

Table 1. Availability of correct rules and finding correct target cases.

	Available	Not Available	Total
Cases with Target Found	41	4	45
Cases without Target Found	7	48	55

One error example is the q514_e4 case. The rules created do not have a strong link. The created rules are "desire_to_delete_messages(man) :- full_of_spam(email_inbox,man)" and "clearing_inbox_space(man) :- deleted_spam(man)". Connecting the "desire_to_delete_messages" to the "clearing_inbox_space" is not so obvious to a LLM when having a pool of other statements. Another problem is the second rule is inverted to a good interpretation of the framework. Some other examples of lack of link between statements are cases like q540_e2, q557_e1, q580_e3.

We can infer that as the LLM converts the sentences to more linkable rules and the framework tends to work in the right way to find the correct target effect given the input query. To reinforce this inference, we selected 6 cases with different depth levels which the framework failed to find the correct forward chain (q514_e4, q519_e5, q557_e1, q569_e1, q577_e3, q643_e3) and manually corrected the rules and the predicates. After re-running the framework for those cases, we found the correct targets. Another important consideration is the difficulty in defining the correctness of the query or the target statement since some words with the similar meaning are selected. The case of q512_e2 is an example: given the query statement “hit head man”, we automatically found the target statement as “got_concussion(he)” but the framework got the statement “got_concussion(man)”. The result is right, but in an automatic analysis we can not know all the possible synonyms that can appear using this Knowledge Base. Another example following this problem is case q533_e6 where the expected target statement is “gain_compliment(man)” but the framework found “gaining_compliment(man)”. Because of the “ing” suffix in “gaining”, the automatic result check algorithm did not find the correct statement, so it is necessary for a human to do visual analysis for each case in order to obtain the real result considering the synonyms.

Another observed problem in results is the repeated occurrence of an almost correct logic chain when the LLM creates the rules, but the last statement created is “desire of doing something” and not “did something”, which breaks the correctness of the expected logic chain. The cases q514_e4 with “desire_to_delete_messages(man)” instead of "deleted_spam(man)", q562_e2 with “desire_to_fire(accountant)” instead of “fired(accountant)”, q611_e3 with “desire_to_call_back(I,girlfriend)” instead of “called_back(I,girlfriend)” are examples of those. This occurs because some explanation sentences in the dataset do not have the explicit action “did something” but have an explanation of why “this thing is done”: by the “desire to do the action”. Those cases penalize a lot the results of this test even having correct cases in the prompt few-shot list.

5. Conclusion

We have proposed a novel framework to execute forward chaining reasoning aided by LLMs that simulate simple elements of first order logic. We observed that the GPT model does not decode the sentences in rules following the few-shot examples correctly in almost half of cases. This is thus difficult for the framework to find the correct path since the link between the rules are not strong enough. Having well crafted logic rules facilitate processing in the solver, improving accuracy of the reasoning.

The nature of the vector embeddings asks for other ways of mathematical modelling to execute the forward chaining – not only inner products to compare the similarity between those vectors. If we consider only vector embeddings to define whether an embedding vector is similar to another, reasoning can go astray. For instance, during the process of the first predicate similarity in q577_e3 case, the mean vector of “intensified(crowd)” has two interesting inner products. The one with just “crowd” has an inner product of 0.72 while the “intense(crowd)” mean vector gets only 0.646, because the embedding vector “intensified” is considerably different from “intense”. However, if we consider the meaning of this logic flow, the more similar vector to “intensified(crowd)” is “intense(crowd)”, not just “crowd”. Therefore, adding the LLMs to aid the logic forward chaining linkage between the rule predicates improves reasoning.

In future work, we plan to evaluate our framework in all the reasoning datasets and benchmarks in papers mentioned in Section 2. Also, we plan to implement a variant of the procedure “node_context_proximity_bart” using Mistral or Llama and to get the product of all probabilities of answer selected tokens. It is also necessary to develop an algorithm to improve the correctness case check, so as to reduce human evaluation effort.

6. Acknowledgements

F.G.C. was partially supported by CNPq grant 305753/2022-3. The authors also thank support by CAPES - Finance Code 001. We thank support by the Center for Artificial Intelligence (C4AI-USP), with support by the São Paulo Research Foundation (FAPESP grant #2019/07665-4) and by the IBM Corporation

7. References

Brassard, A., Heinzerling, B., Kavumba, P., & Inui, K. (2022). COPA-SSE: Semi-structured Explanations for Commonsense Reasoning. *Proceedings of the 13th Conference on Language Resources and Evaluation*.

Creswell, A., Shanahan, M., & Higgins, I. (2023). Selection-Inference: Exploiting Large Language Models for Interpretable Logical Reasoning.

DeepSeek-AI. (2025). DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning.

Jin, F., Liu, Y., & Tan, Y. (2024). Zero-Shot Chain-of-Thought Reasoning Guided by Evolutionary Algorithms in Large Language Models.

Kazemi, M., Kim, N., Bhatia, D., & Xu, X. (2023). LAMBADA: Backward Chaining for Automated Reasoning in Natural Language.

Lee, J., & Hwang, W. (2024). SymBa: Symbolic Backward Chaining for Structured Natural Language Reasoning.

OpenAI. (2025, April 04). *GPT 4o-mini Release*. Retrieved from Portal Open AI: https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/

Saparov, A., & He, H. (2023). LANGUAGE MODELS ARE GREEDY REASONERS - A SYSTEMATIC FORMAL ANALYSIS OF CHAIN-OF-THOUGHT.

Tafjord, O., Mishra, B., & Clark, P. (2021). ProofWriter: Generating Implications, Proofs, and Abductive Statements over Natural Language.

Toroghi, A., Guo, W., Pesaranghader, A., & Sanner, S. (2024). Verifiable, Debuggable, and Repairable Commonsense Logical Reasoning via LLM-based Theory Resolution. *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, (pp. 6634–6652).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., . . . Polosukhin, I. (2017). Attention Is All You Need.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., . . . Zhou, D. (2023). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.

Williams, A., Nangia, N., & Bowman, S. (2018). A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference.

Zhou, D., Scharli, N., Hou, L., Wei, J., Scales, N., Wang, X., . . . Chi, E. (2023). LEAST-TO-MOST PROMPTING ENABLES COMPLEX REASONING IN LARGE LANGUAGE MODELS.