System Description: DeepLLM, Casting Dialog Threads into Logic Programs

Paul Tarau

University of North Texas

May 6, 2024

code at https://github.com/ptarau/recursors/
demo at https://deepllm.streamlit.app/
demo at https://deep-auto-quests.streamlit.app/

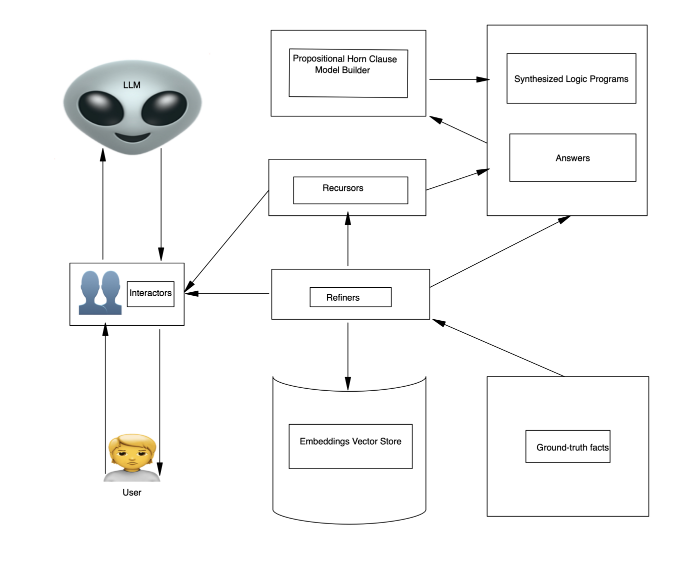
Overview

- DeepLLM is a system that automates deep step-by step reasoning in an LLM dialog thread by recursively exploring alternatives (OR-nodes) and expanding details (AND-nodes) up to a given depth
- starting from a single succinct task-specific initiator we steer the automated dialog thread to stay focussed on the task by synthesizing a prompt that summarizes the depth-first steps taken so far
- semantic similarity to ground-truth facts or oracle advice from another LLM instance is used to restrict the search space and validate the traces of justification steps returned as focussed and trustable answers
- Applications:
  - consequence predictions
  - causal explanations
  - step by step guidance to achieve a goal
  - topic-focussed exploration of scientific literature
  - supporting arguments for a thesis
  - recommendation systems

Introduction to DeepLLM

- Purpose: enhancing interactions with large language models (LLMs).
- It steers the LLMs' dialog thread by casting it into a Logic Program
- Dually: it tweaks logic programming for the LLM context
- Aims to streamline information retrieval and interaction with LLMs
- Addresses the complexity of prompt engineering
- Streamlines user engagement with LLMs across various applications

DeepLLM: System Architecture



DeepLLM: key components

- Active components (“Agents”): interactors, recursors, refiners.
  - Interactors manage input prompts and task breakdown
  - Recursors handle iterative exploration of subtasks
  - Refiners enhance clarity and relevance of LLM responses
- Resources:
  - Ground truth facts: sentences collected from online sources or local
  - Vector store: enables “semantic search” via embeddings of sentences

A first example: AGI as seen by GPT4

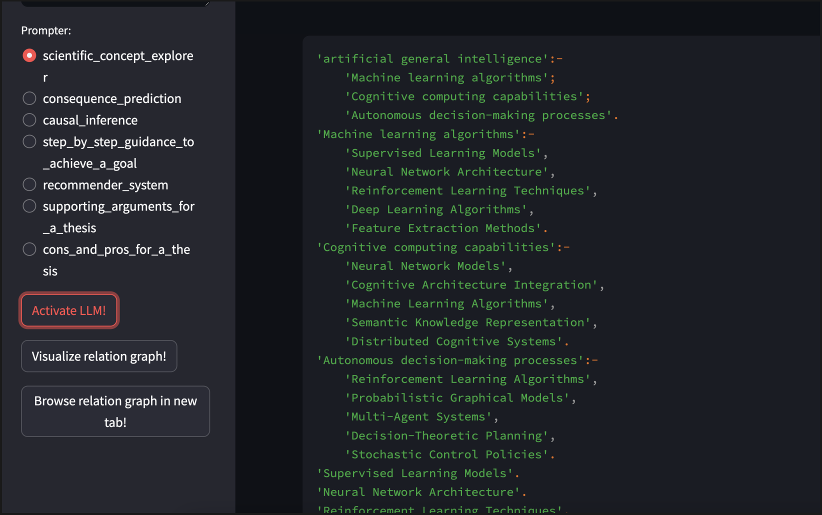


Figure: A Logic Program: AGI as seen by GPT4

Graph of Relations defining AGI

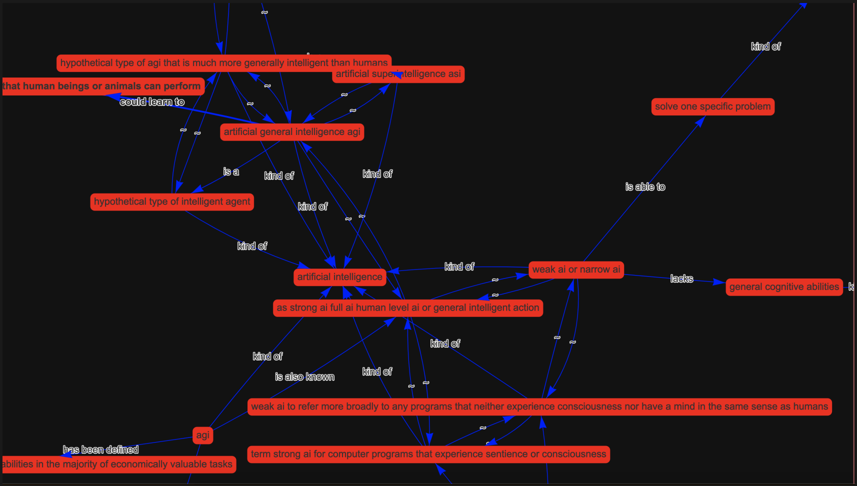


Figure: Relations defining AGI

Logic Programming : Propositional Horn Clauses in Prolog

```
pet :- dog.
pet :- cat.
pet :- snake.

dog :- barks, walks, bites.

cat :- purrs, walks, hisses.

snake:- hisses, slither, bites.

walks :- true.
purrs :- true.
bites :- true.
hisses :- true.

slither :- false.
bites :- false.
barks :- false.
```

```
?- pet.
true % pet -> walks, purrs, hisses -> cat
```

DeepLLM: Logic Programming, but done differently!

- SLD-resolution's clause selection via unification is replaced by LLM-driven dynamic clause head creation with an option of focusing by proximity of embeddings to ground truth facts
- as dialog units are sentences, the underlying logic is propositional
- client-side management (via the API) of the LLM's memory is based on the equivalent of a *goal stack* and a *goal trace* recording our steps on the current search path
- instead of variable bindings, answers are traces of justification steps clearly explaining where they are derived from
- when their depth-limit is reached, the items on the goal stack are interpreted as "abducibles", statements that can be hypothetically assumed and then checked against "integrity constraints"

Keeping the ground truth in mind

- our depth-bounded refinement steps support compilation of the dialog threads to a Horn Clause program to be explored with logic programming solvers
- modular, task specific, customizable prompt engineering primitives are aggregated together for “AND-step” and “OR-step” prompts
- normalized semantic similarity measures of embeddings can be made available when generating probabilistic logic programs
- sentences in authoritative documents or collections seen as "ground-truth facts" can be used to select abducibles via semantic similarity or advice of an LLM-based oracle

Overall, our approach exploits synergies between structured prompt engineering, logic-guided recursion over LLM queries and semantic search in an embeddings vector store.

Interactors

- the LLM API: same client-server interface for local and outside LLM
- prompt templates example:
  - AND-OR prompt patterns for causal reasoning

```
causal_prompter = dict(
    name='causal',
    and_p="""We need causal explanations in this context: "$context"
      Generate 3-5 explanations of 2-4 words each for the causes of "$g".
      Itemize your answer, one reason for "$g" per line.
      No explanations needed, just the 2-4 words noun phrase,
      nothing else.""",
    or_p="""We need causal explanations in this context: "$context"
      Generate 2-3 alternative explanations citing facts that might
      cause "$g".
      Itemize your answer, one noun phrase per line.
      No explanations needed, just the noun phrase, nothing else.
      """
)
```

Recursors

- OR-prompter expanding the head h in a series of alternatives $a_1, \ldots, a_n$:
  - $h \; :- \; a_1.$
  - $h \; :- \; a_2.$
  - $\ldots$
  - $h \; :- \; a_m.$
- more concisely expressed with a disjunctive body as:
  - $h \; :- \; a_1; \; a_2 \; ; \; \ldots \; ; \; a_m.$
- the result of an AND-prompter:
  - $h \; :- \; b_1, \; b_2, \; \ldots, \; b_n.$
- when a depth limit is reached, the remaining unexplored goals are considered as *abducibles*

Refiners

- The Embeddings Vector Store
  - Filtering abducibles with semantic distance to ground-truth facts
- Refining decisions with LLM-based oracles
  - The LLM-based True/False Decider
  - The LLM-based Rater

The Model builder: fast Propositional Horn Clause Satisfiability solvers

- loops in the generated clauses would create problems with Prolog's depth-first execution model
- also, satisfiability of propositional Horn clause logic is in P (actually, there's a linear graph-based algorithm by Dowling and Galier)
- implement a simple solver, propagating truth from facts to rules!

Implementing the The Model builder

- given a Horn Clause $h : -b_1, b_2, ..., b_n$, when all b_i are known to be true (i.e., in the model), h is also added to the model
- if integrity constraints (Horn clauses of the form $false : -b_1, b_2, ..., b_n$) have also been generated by the oracle agents monitoring our refiners, in the advent that all $b_1, b_2, ..., b_n$ end up in the model, $b_1, b_2, ..., b_n$ implying *false* signals a contradiction, thus unsatisfiability of the Horn formula associated to the generated program
- options to handle it:
  - return the logically correct answer: no solutions
  - stop as soon as a proof of the original goal emerges, assuming that contradictions far away are not affecting the result

An alternative (not in the paper, but implemented): a torch-based GPU-friendly solver

- `https://github.com/ptarau/recursors/tree/main/tenslogic`
- based on Sakama, Inoue and Sato's tensor fixpoint computation
- to take advantage of GPUs, we use torch
- while complexity is not linear anymore, GPU-acceleration makes it practical for mid-size LLM-generated programs
- future work:
  - use of sparse tensors for reduced complexity
  - adaptation to work with vector embeddings and soft unification

Toward Trustable Generative AI

- trustable AI: requirement for wide adoption of today's LLMs in medical, educational, defense and several other business applications
- likely subject of upcoming government regulations
- $\Rightarrow$ DeepLLM provides a principled approach toward trustable generative AI:
  - oracles enforcing consistency and factuality
  - focussed reasoning steps enforced by casting AND/OR steps into a propositional Horn clause program

How DeepLLM enhances a user's LLM Interactions?

- Reduces complexity of prompt engineering
- Automates decomposition of tasks into simpler sub-tasks
- Refines prompts for more accurate user intent capture
- Generates dynamic responses for a wide range of queries
- Improves user experience with concise and relevant information

Future Work

- Granularity Refiners
  - higher granularity: one can work with sentences/statements instead of noun phrases
  - lower granularity,: SVO triplets
- Question generators
  - asking LLMs to generate follow-up questions, recursively
- Diversifiers and Harmonizers
  - to restrict unwanted "hallucinatory" generation twists *diversifiers* and *harmonizers* can be expressed as additional integrity constraints
  - diversifier: no two OR-nodes are too close semantically
  - harmonizer: no two AND-nodes are too far semantically
  - implemented with semantic distances in the embeddings vector store
- Extended Implementation Techniques
  - fixpoint computation with torch tensors (actually already working!)
  - extend the power of the underlying logic language: ASP, probabilistic LP, Symmetric LP - based on Dual Horn clauses - to be out soon!

Conclusion

- automation of deep step-by step reasoning in LLM dialog threads while staying focussed on the task at hand
- we have made LLMs function as de facto logic programming engines that mimic SLD-resolution as they invent propositional Horn programs
- instead of trying to parse sentences into logic formulas, we have accommodated our logic engine to fit the natural language reasoning patterns LLMs have been trained on
- semantic similarity to ground-truth facts and oracle advice from another LLM instance has been used to restrict the search space and validate the traces of justification steps returned as answers
- focussed, controllable output, enabling:
  - deep investigations into details of specific scientific domains
  - expert-level causal reasoning or consequence predictions
- a method to extract hallucination-free focussed knowledge as a logic program that encapsulates trustable AI in a clearly expressed and easily verifiable framework

Questions?

Thank you for listening!

Email: paul.tarau@unt.edu

Links, ready to try out!

DeepLLM code at `https://github.com/ptarau/recursors/`
demo at `https://deepllm.streamlit.app/`
demo at `https://deep-auto-quests.streamlit.app/`