LLM Timeline Generator using ReAct with Self Reflection

Nhat Nguyen

10/3/2024

Problem

Context:

State of the art LLM models today require massive compute for training and operation. Once a model is trained, it can be used, however, it cannot learn new facts about the internet after it has been trained. This point in time is often referred to as the cutoff point. The current LLM strategies employed by companies such as OpenAI and Anthropic are to gather data and retrain their models on the new data occasionally.

Problem:

Because AIs have this cutoff point in their training data, they cannot react to new data. If a user were to ask an AI model about the recent news on Bitcoin mining or recent interesting research papers, they would receive an “I don’t know” response from the LLM.

Currently, some tools exist to provide the LLM web search access, however, these models often focus on solving a singular question (example, “what is the height of the Eiffel tower?”) instead of solving more open-ended questions (example, “should I go swimming at Cocoa beach today?” which would require planning to search weather, public advisory, ratings of beach, etc).

A useful task, often time consuming and manual, is scouting research. To keep up to date on the latest information, manual searching and reading is required (if one is lucky, maybe a good emailer’s digest). For any question, it would be useful to have an LLM generate a good summary and visualization of the information. For this project, the scope will be a timeline.

Project Objectives

This project aims to create a framework for an off-the-shelf LLM to search a database (given tools to do so), pick out relevant information, and store it in a timeline format. The model must also direct itself to its own research by asking questions.

1. for an LLM to perform the ReAct architecture (reasoning, acting, observing, repeating)
2. for an LLM to observe the results and ask questions if necessary and also perform the actions in parallel
3. for an LLM to “take notes” (capture the source to the information, summarize it, store it for later recall)
4. to create a system that reads the notes and creates a displayable data format in the form of a timeline

The expected outcome should be an LLM system that takes in a user prompt and generates a timeline after doing research on the web.

Methodology

Approach:

Given a user prompt to the LLM with a pre-written system prompt, allow the LLM to explore certain parts of the web on its own. The base framework will follow ReAct protocol.

However, another two layers will be implemented called **questions** and **notes**. Also to save token usage, the model will be queried a new prompt each time. This prompt will include its notes, active questions, answered questions, last previous action & thought (concatenated if parallel), the user’s prompt, list of actions, and other important metadata (like today’s date & time).

Questions will allow the LLM to log additional nodes to explore on the web by querying other tools. When evaluating the result, the LLM is allowed to push multiple questions to its queue. After reading the results of each web article, it may fill out multiple answers to its questions and store it in its state. The model must answer all questions before it can terminate.

Questions contain a question and a brief answer to it.

Notes will be short summarizations the AI has made of the researched to allow the LLM to store its session information without consuming tokens to store the entire page. Notes will

also be stored internally in a database and can be recalled at any time by the LLM or user for better explainability).

Notes contain the pointer to the source, the quoted text, and a summary.

Given the prompt: “Give me the last 30 days of what’s happening at tesla.” The LLM may obviously choose to create the initial questions such as “what’s happening at tesla last 30 days”. This would return a page of results.

Given the results, the LLM may now branch out its initial question with new questions (such as “what is the tesla robotaxi?” and “elon musk talk” and attempt to recurrently traverse the question nodes until it is satisfied.

LLM(s) and Techniques:

LLMs:

- GPT-4o latest version
- Claude 3.5 Sonnet

Techniques:

- LangChain: used for wrappers and helpers for interacting with the LLM
- ReAct & Tools: ReAct prompting plus additional custom coded tools the LLM can use (Multi-Turn Instruction)
- Prompt Engineering: ReAct and structured output prompting strategies
- duckduckgo-search : a library for querying DuckDuckGo for search results
- Wikipedia API: for searching wikipedia
- Scholarly (https://github.com/scholarly-python-package/scholarly): used for searching google scholars for papers

Architecture/Process:

1. Initial prompting: User queries the prompt
2. Question Generation: LLM generates a list of questions regarding the prompt
3. Action: loop through all questions and let LLM Decides what action to perform (multiple actions are allowed):
    a. Ask the user some questions
    b. Search the web for questions
    c. Finish
    d. Update internal values (add/remove notes/questions)
4. Go to step 2, repeat until finish.
5. Loop through notes data type and convert it to a timeline

Data:

No training data is needed for this project. Testing data will be created manually. For testing, 10 handwritten prompts will be created along with 5 expected handwritten summaries of the current news. Careful selection will be made to acquire only recent news occurring after the model's training point.

Evaluation:

Given the output timeline of an LLM (list of sources, quotes, summaries, dates), we compare it with our ground truth (list of summaries, dates). BLEU scoring and human grading will be performed on each output. If additional time allows. The original ReAct and Act methodology will be used for scoring. This will also be done separately for GPT-4o and Claude 3.5 Sonnet.

Related Work

Literature Review:

3 Papers were reviewed.

When Search Engine Services meet Large Language Models: Visions and Challenges (https://arxiv.org/abs/2407.00128)

This paper provides an overhead view of the current landscape of search engine based LLM solutions. Its area of focus is using search engines to improve LLMs (known as Search4LLM) and its opposite where LLMs improve the search engine (known as LLM4Search). The area of interest is section 3.3 detailing using LLMs content alignment detailing a potential architecture for online searching. Another interesting area is 4.2.2 noting that LLMs perform well when indexing pages or extracting semantic labeling. Overall, the paper gives a good understanding of where to start when implementing a search engine based LLM.

ReAct: Synergizing Reasoning and Acting in Language Models (https://arxiv.org/abs/2210.03629)

The Search Engine paper above builds upon this paper. ReAct is the prompting strategy proposed. The model is allowed to perform an external action, observe the result, reason with it, then repeat until it is satisfied. It performs better than pure chain of thought (CoT) reasoning. This paper will be the initial foundation for this project. In terms of decisions, the results deemed that CoT combined with ReAct leads to better results. Specifically,

CoT-SC (self-consistency baseline). Due to the small scope of this project, CoT will be used in place of CoT-SC. The rest of the ReAct architecture will be used.

From Local to Global: A Graph RAG Approach to Query-Focused Summarization (https://arxiv.org/abs/2404.16130)

This graph RAG paper presents the idea that paragraphs are semantically correlated with other paragraphs. As a result, each RAG paragraph can be treated as a node, and each note connection be treated as a semantic correlation. This leads to the development of a knowledge graph which exhibits useful properties. The most important property is the ability to query the graph at different hierarchical levels, which allows LLMs to retrieve the high-level ideas first, then refine it. These refined ideas can be used to traverse the nodes to find other similar nodes, eventually leading to a subgraph of related information. This lowers the chance of a bad retrieval and also provides the information source/link.

Positioning:

This project builds upon the ideas of all three papers. Here we use the same ReAct-like system but allow the model to perform multiple actions. The main goal of this is to reduce the number of processed tokens and improve the response time. Breaking the actions into multiple pieces allows tasks to be executed in parallel. In terms of labelling and organization, this project draws ideas from Graph RAG's concept of recalling information through a relationship graph. In this project's implementation, we will treat it similarly to a Graph RAG search with an unknown initial RAG state.

The project will therefore make these contributions:

- Multi-action ReAct
- Graph RAG runtime indexing: building the graph rag during prompting
- Timeline extraction: combining all features to extract a timeline

Timeline

Phase 1: (month 1)

- LLM Setup: setup LLM APIs in LangChain
- Implement DuckDuckGo searching
- Implement note taking (only summaries, quote)
- Implement question generation
- Implement finish

Phase 2: (month 2)

- Finish implementing note taking (date, source link)
- Implement timeline generation
- Implement parallel querying
- Implement the remaining tools (wikipedia, scholarly)

Milestones:

Milestone 1: The model can generate questions and then requests actions to be performed (the actions won't be implemented yet) (10/19/24)

Milestone 2: The model can perform the search and finish it requests for and take notes (11/2/24)

Milestone 3: The model can perform all tool actions (wiki search, scholarly) + the entire methodology (11/16/24)

Milestone 4: Code submission, documentation, evaluations are done (11/30/24)

Challenges and Risks

Data, computational resources, and model complexity don't affect this project due to the use of an off-the-shelf LLM. Time constraint will be an issue if additional work is needed to sanitize the LLM due to bad output format. Another potential risk is the trouble of setting up additional libraries and writing the action interpreter and learning to properly use those libraries.

Mitigations in place will be to cut milestone 3 and cross evaluation (Act vs ReAct vs Timeline).

Resources Needed

Hardware/Software:

No specialized hardware or software needed. Only APIs credits are needed for this project.

Data Requirements:

External APIs required:

- LLM model access for OpenAI and Anthropic
- Duck-duck-go search

- Wikipedia
- Google scholarly (unofficially done through scholarly python lib on github)

Expected Deliverables

Code:

Main.py - wrapper script to execute the prompt query

TimelineLLM.py - class object to perform the querying and tracking, questions, answers, and notes. It also produces the timeline data object.

Interpreter.py - class that attempts to sanitize the input from the LLM.

Actions.py - class that contains all the possible actions that the LLM can do.

Final Report:

Methodology includes the same methodology as this proposal unless an interesting discovery is made that pivots the project.

Experimental results will outline Act vs ReAct vs Timeline, compared with 2 LLMs (GPT & Claude) via score comparison between each timeline entry. Timeline outputs will be scored using a custom metric combining: F1 for event date matching (was the date good?), ROUGE score for summary matching, and cosine similarity via sentence embedding.

Analysis will include graphs and sample output.

Additional Artifacts:

A demo is currently planned. I would like to screenshare my project and type in example queries for my peers. If time allows and results are interesting, a presentation will also be included. Visualizations in the form of trees will be included if time allows.