Real-time Retrieval Argumentation for Large Language Models

Udit Raj

1INDIAN INSTITUTE OF TECHNOLOGY PATNA BIHTA -801106

May 02, 2024

Abstract

We are in the midst of a race of who can scrape most of the internet and put it on their servers in the least timeframe. Yes, that’s how all the language models are conventionally built. However, this process is slow, this process is old, this process hallucinates, and doesn’t know what is happening in the real-world. And that’s why this convectional process needs a replacement. Real-Time Retrieval Argumentation is the alternate architecture for LLMs that the new age models need to adapt. With RTRA, a model trained on just 7 billion parameters can beat models trained on hundreds of billions of parameters. This research paper marks a paradigm shift by challenging our dependency on huge computational resources to build a precise and efficient model by introducing a novel approach to train and develop language models.
Real-time Retrieval Argumentation for Large Language Models

A research paper by

Udit Raj
udit_2312res708@iitp.ac.in

INDIAN INSTITUTE OF TECHNOLOGY
PATNA BIHTA - 801106, INDIA

Date. 23rd April, 2024
Declaration

I hereby declare that this submission is my own work and that, to the best of our knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

21.03.2024
Udit Raj
Summary of the Project

We are in the midst of a race of who can scrape most of the internet and put it on their servers in the least timeframe. Yes, that’s how all the language models are conventionally built. However, this process is slow, this process is old, this process hallucinates, and doesn’t know what is happening in the real-world. And that’s why this convectional process needs a replacement.

Real-Time Retrieval Argumentation is the alternate architecture for LLMs that the new age models need to adapt. With RTRA, a model trained on just 7 billion parameters can beat models trained on hundreds of billions of parameters. This research paper marks a paradigm shift by challenging our dependency on huge computational resources to build a precise and efficient model by introducing a novel approach to train and develop language models.
Content

1. Introduction to RTRAs
2. Technical Architecture
3. Implementation
4. The Idea of Human Psychology
5. Hallucination Reduction & Citations
6. Latency Reduction & Chunking
7. Rtrafactor – Python SDK for RTRA Architecture
8. Environmental Impact
9. Fighting Plagiarism
10. Constraints & Scope of Improvement
11. Conclusion
12. References
Chapter 1

Introduction to RTRAs

Machine learning models are not new and throughout the time, we have devised multiple approaches to make our models more efficient. However, the foundation behind building a model has always been the same -

*Scraping a lot of data from the internet, uploading it on our computers and servers and letting our model retrieve that data and form a distinguished response.*

In fact, beside some lines of codes and some basic architectural changes, this has always been the foundation of every machine learning model out there.

Although this architecture helps us easily scale our models by increasing the amount of data rows it is trained on, it always puts us in a dilemma of what exactly will be the most superior machine learning models and how many rows of data will it be trained on.

This in contrast with Moore’s law\(^1\), depicts a blurry picture of the future that the race for the best model might never end.

However, this paper challenges the regular approach towards how machine learning models are trained and helps build an alternate architecture to train models on a fraction of the required data and retrieve precise responses.

This, in terms, not only litigates our dependence on expensive computational resources, but also helps us to reduce hallucinations and latency upon the development of our model. In fact, it goes one step ahead and opens new doors for a model to retrieve real-time data to formulate its responses.

Real-time Retrieval Argumentation (RTRA) is a best-in-class approach of utilizing dynamic web as a secondary dataset and training our models on only a small portion of parameters required to understand and formulate responses based on a retrieval-based mechanism.

In fact, as we see in the further sections, with the utilization of RTRA architecture, Google’s Gemma, a 7 billion parameter small model, was able to beat Google’s Gemini Pro 1.0, a 100 billion+ parameter model, on several retrieval-based performance benchmarks.

Through this paper, our goal is to enhance the performance of certain in-scope models with the RTRA architecture, and build an interface for people to build, tune and interact with these models.
Chapter 2

Technical Architecture

Present-day scenario: Dependence on a static database and constraints of the current architecture.

The above diagram (d1.0) represents a general architecture of a language model which utilizes fine-tuning or embedding functionality. This is currently a state-of-the-art model as embedding vectors and fine-tuning gives it connectivity with external knowledge that in-fact the model was not trained on. However, if we take a closer look at the architecture, we will notice that all the three foundational databases that are in action in this model, are static.

This means the model has absolute zero interaction with the external world.

At the first sight, this might not appear as a significant problem as embeddings are working externally and there must be a way to substitute the vector database with a real-time data connectivity.
However, vector databases work by storing and indexing high-dimensional vectors through processes like indexing, query processing, representation, etc. And as soon as we integrate a real-time data source, the vector architecture starts to fall apart.

This happens due to several limitations of a vector-based database such as indexing, latency, durability, ingestion, concurrency, etc.

To better understand, think of a vector database as a distinct arrangement of constellation stars in our universe. Each star holds a unique position and has a fixed location in the three-dimensional space. This, in turn, is why vector databases fail to interact with dynamic and real-time datasets. Because they are structured as distinct vector-locations at distinct positions. And then assume that each of these stars represents some data stored in a vector database.

Now, if we run a query, we want to reach the star to get the data it holds. And as we know the location and position of that star, it is very easy for us to reach there and get the data.

However, suppose that every second, the space sees different stars with different data and different locations.

Then, practically you will need a lot of computation and resources to reach the star and collect the data.

---

**How a vector-based database works?**

![Vector Database Diagram](Diagram.png)

---

as a secondary database for language models.
Through this paper, we tend to introduce an alternate language model architecture that utilizes real-time web as a dynamic secondary database.

Furthermore, we tend to discuss the retrieval architecture argumentation and summarization of multiple responses that is generated throughout different stages in the workflow.

The diagram d3.0 represents the workflow of models utilizing the *Real Time Retrieval Argumentation* architecture.

Unlike the general architecture, RTRA focuses on completing 03 major processes to generate a response -

1. Retrieval
2. Response Generation
3. Argumentation

Let us discuss the architecture in detail in the next section, i.e., methodology.
RTRA Architecture : Methodology

To discuss our architecture in detail, we will divide our methodology into three different stages -

1. Core Architecture
2. Underlying Processes
3. Algorithms & Implementation

Core Architecture
Whenever a new query is initiated in a model equipped with RTRA architecture, the query is sent to two different locations - the model & the search section.

The search section performs a web-based search and finds links to relevant sources which, in turn, might hold the answer we are searching. Then, this result is sent to the scraper section of the library which reads through all the data available on those links.

Here, the preprocessing model triggers and a primary summarization takes place. This process is important as most of the models have a very limited context window and might not be able to read all the scraped data. We will read more about this in detail in further sections.

Further, the preprocessed data is sent to the model through its context. The model combines this context along with the initial query to generate one response.

Now, due to multiple processes running behind the scenes, the answer might not always be in the most human-readable format. There’s where a secondary summarization model is triggered to facilitate completion, formatting and summarization of the model’s response.

Underlying Processes
As we talked about in the last section, there are three major underlying processes at work in the RTRA architecture - Retrieval, Response Generation, and Argumentation.

Retrieval. In the initial stage, the system retrieves information relevant to the user's query. It initiates a web search, submitting the user's query to a search engine. The relevant search results returned by the engine contain URLs of web pages deemed relevant to the query.

The system then accesses each of these web pages and extracts textual content using web scraping techniques. This process involves retrieving the HTML content of the web pages and parsing it to extract the desired text, which will be used for further processing.

Response Generation. With the extracted textual content at hand, the system generates an initial response to the user's query. It utilizes the underlying model to generate this response.

The extracted text and the user's query are input into the model, which leverages pre-trained language models to generate a response tailored to the query.
If the initial response is lengthy or overly detailed, the system may opt to further summarize it into smaller, more manageable chunks. Each chunk undergoes summarization using the same underlying model, resulting in concise summaries of the information.

**Argumentation.** In this stage, the system assembles the summarized chunks of text and associated citations into a final response.

It organizes the summarized content and citations in a coherent manner, structuring them to present a clear and informative response to the user's query. The final response aims to provide users with a comprehensive understanding of the topic while also offering references to the original sources of information.

This presentation of the information in a human-like manner enhances the user's experience and fosters trust in the accuracy and credibility of the response.

**Algorithms & Implementation**

We built a **one-shot response retrieval argumentation** that utilizes the underlying processes in a certain order, combined with a novel prompt to classify and summarize multiple responses generated through the architecture into a single response.
Chapter 3

Implementation

While building the RTRA architecture, our goal was to deploy small instances of our work globally to let waitlisted users get access to what we are building.

We started off with the development of a real-time chatbot which we tested with over 100 users. Later, we scaled that agent to build Eternity Search, a real-time semantic search engine.

For a hackathon at NIT-Patna, we built a real-time fine-tuned model, backed by RTRA architecture and based on Mistral 7B open source. There we met many people and Phd students who got interested in our project and later converted to join us as individual contributors to this technology.

More than 300 users participated in testing our Eternal Search and BH1 (a real-time bug detection model that we built over RTRA) and we received over 50+ distinct constructive feedback on how to improve the underlying tech.

All this led us to understand that different developers might be building or utilizing different language models for different use cases and building just one model will not solve their problem.

That is when we started building Rtrafactor - our own Python library to integrate the RTRA architecture with any models that are available on the HuggingFace’s Inference API. The single Python library tends to make language models 30x more precise than the non RTRA-implemented predecessors.

One of our key focuses was to let anyone achieve singular model efficiency with the utilization of the RTRA architecture. With our contributions in open-source, we tend to achieve this. In the first four days of making it public, Rtrafactor crossed 800+ downloads mark on PyPi.org (1,500+ downloads including mirrors) with seven public releases of our library, pushing us to the 0.7 version while in beta.

The library is currently being rapidly upgraded with new bug fixes, efficiency updates and feature addons.
Chapter 4

Inspiration from Human Psychology

Overview

When we started working on this paper, we just wanted to build a model that can replicate the human mind. To be precise, we wanted to create a human clone. However, as soon as we dived deep into this field, we became familiar with multiple constraints of language models. The few most significant of which were the real-time retrieval process, latency and hallucinations.

Think about it as your sibling asking you which watch to buy. Here, you have 2 ways - you either go and download every content on the internet available about watches offline and spend the next 2 years just to learn about it. Then, after you have all the needed knowledge, you can tell your sibling that okay, the Breguet Grande Complication 2019 will be the best. However, this is not the most ideal way because neither of you have the time and patience to wait for 2 hours until your training completes. Additionally, many exceptionally good watches might appear in these two years which you will have no idea of.

That's why, you, me or any human will go with the second way - researching about it. Here, you might go and search about the best watches on Google, maybe on Amazon, or maybe you will watch videos about watches on YouTube. The best part while going this way is that based upon the time constraint, you can conclude your research in 1 minute, 1 hour or even 1 day.

And this is what stuck in our head - we found that there's no way that what's happening in this moment can be synchronized with a model's training dataset in real-time. However, if we tweak how we perceive the general architecture of language models, we can change the retrieval flow of LLMs.

Our focus shifted from modifying the training dataset to building a secondary dataset that can be triggered after the query is initiated.

For Rtra to work ideally, the underlying model has to be good at not text completion but summarization. This is far less complicated and can be done with a very light database.

Just like how we do not exactly need to know about watches to suggest someone a watch, the RTRA architecture also makes sure that your model doesn't need a huge global training dataset to provide precise responses.
Chapter 5

Hallucination Reduction & Citations

Overview

The RTRA architecture highly reduces language model hallucinations through its response argumentation mechanism and real-time web connectivity. The v0.8 of our Python SDK also introduces citations which means that every source of the model's response can be easily verified and cited.

For a traditional model, it is impossible to generate citations due to the underlying architecture. The models are built to generate a response out of billions of offline content files whereas, any model integrated with RTRA architecture works primarily with a dynamic online database that can be easily retrieved and cited for reference purpose.

Impact

For the world of research and academic studies, this architecture sets benchmarks as the genuinity of the model’s response highly increases and research questions can be sourced from well-established institutions to further make inventions, discoveries, studies and research more efficient.

Architecture

Web search, crawling and content scraping, and response argumentation leads to the foundations of reduced hallucinations in the RTRA architecture. Whereas the citation model along with the web search method of our architecture, together makes the references and citations possible.

Let us better understand the underlying process in RTRA architecture which powers the citation references and reduces hallucinations within our model. See diagram D4.0 for better understanding:
Chapter 6

Latency Reduction & Chunking

Once the RTRA architecture is triggered to respond to a query, multiple underlying processes are initiated. Due to this, a system overload happens and a sudden latency spike occurs. To tackle this, we introduced a chunking method.

Chunking is basically a process to break a huge singular data into multiple parts either during retrieval or response generation. Let us find how RTRA utilizes chunking:

1. **Chunking for Summarization.** The scraped text is divided into smaller chunks to facilitate summarization. The chunking process involves breaking down the accumulated text into segments of a fixed size. Each chunk represents a portion of the text that is small enough to be processed effectively by the summarization model.

2. **Iterative Processing of Chunks.** After chunking the text, the system iterates through each chunk individually to generate a summary. This iterative approach allows the system to focus on summarizing smaller portions of the text at a time, improving the efficiency and accuracy of the summarization process.

3. **Summarization of Each Chunk.** For each chunk of text, a summarization prompt is constructed based on the content of that chunk. The prompt typically instructs the summarization model to generate a concise summary of the provided text. The chunk-specific prompts are sent to the model, which generates summaries for each chunk independently.

4. **Combining Summaries.** Once summaries are generated for each chunk, they are combined to form a comprehensive summary of the entire input text. The combined summary provides a condensed overview of the original text, capturing the key information across all chunks.

5. **Handling Citations for Chunks.** Alongside summarization, the system also collects citations for each chunk of text. Citations include the URLs of the web pages from which the text was extracted, providing users with references to the original sources of information. These citations are associated with their respective chunks and included in the final output along with the summaries by the Citation model that is triggered as the final process of the RTRA architecture.

By employing chunking techniques, the system effectively breaks down large amounts of text into manageable segments for summarization. This approach enhances the efficiency and effectiveness of the summarization process, allowing the system to generate concise summaries while maintaining the integrity of the original information.
Chapter 7

Rtrafactor - Python SDK for RTRA Architecture

Overview

Although RTRA is a straight-forward model architecture, users might get lost in development complexities. That is why we developed Rtrafactor - an open-source Python SDK to integrate RTRA architecture to your pre-existing models.

Currently, models deployed on HuggingFace Inference API can be easily integrated with RTRA through Rtrafactor. Support for custom models is still not available but will be accessible in future updates.

Rtrafactor best works with summarization & instruct models and has been tested rigorously with the Mistral-7B-Instruct-v0.2.

Prerequisites & Limitations

Although the RTRA architecture is built to adapt to any model without complexities, the current versions of the Rtrafactor library has certain limitations:

1. **Text classification models**: Rtrafactor is currently tested only on text-based models.
2. **Instruct Models**: Rtrafactor was able to work with general models too, but performed best with instruct models.
3. **Summarization Models**: If you are building a custom model to work with Rtrafactor, we recommend building a small summarization model which has the main objective to summarize and process data.
4. **Wide Context Window (Optional)**: This one is optional but you can make sure that the base model has a little wider context window.

Methodology

To integrate Rtrafactor with your language model, go to our official GitHub repo for instructions. Meanwhile let us discuss the different components of the library.

Installation

```
pip install rtrafactor
```

Install Rtrafactor into your project using pip. The project is hosted on PyPi with an Apache 2.0 license.
**Package Validation**

`pip show rtrafactor`

Ensure the authenticity of the installed package and make sure that the latest version of Rtrafractor is installed.

**Usage**

Rtrafractor is available as a L2LM (Language-to-Language Model) architecture, currently integrated with HuggingFace's Inference API. Follow these steps to use Rtrafractor:

**Import RTRACConnector:**

```python
from rtrafactor import RTRACConnector
```

**Instantiate RTRACConnector:**

```python
connector = RTRACConnector(huggingface_model, huggingface_api_token)
```

**Query for Answers:**

```python
query = "Your question here?"
one_shot_answer = connector.compare_answers(query)
print(one_shot_answer)
```

**Example**

Here are some example queries you can try with Rtrafractor:

1. Why is Delhi's CM in jail?
2. Who is Dr. Kuldip Singh Patel?
3. Who is Udit Akhouri?

**Legacy Projects**

If you are willing to get hands-on products that laid the foundation of Rtrafractor, feel free to [click here](#) and visit Eternal Chat.

This was the most primitive and basic project that laid the foundation of Rtrafractor and the RTRA Architecture.
Chapter 8
Environmental Impact

Problem
Our research empowers sustainable computing and is oriented to place a positive environmental impact with our findings. With this paper, our focus is to reduce our dependence on huge computational resources for training a model.

We cannot ignore the importance and value additional that powerful LLMs can bring to human civilization but we can also not ignore the negative impact that running huge servers for training those models adds on to our environment as most of the electricity consumed in operating these servers comes from non-renewable energy sources and together, these servers increases the enthalpy of our universe.

This leads to various unnecessary environmental changes from melting glaciers to unpredicted rainfall.

We can see that the rate of YoY temperature increase took a sudden acceleration after the 1960s. Even if global changes happen, we cannot disregard the amount of heat dispersed in the surrounding from transistor and server related development, research and operation:

Solution
Although our research can not completely eliminate the usage of servers and computational resources for training a language model, it can help reuse pre-existing databases and create a sustainable model training workflow.

Currently, duplicate data is available on different servers. Some articles uploaded by an author on his website are uploaded at multiple server locations by different developers for their models.

If developers follow the RTRA architecture, the same articles published by the original author can be sourced by different models by different developers without using additional computing and increasing the overall global enthalpy.
Chapter 9
Fighting Plagiarism & Rights of Authors

Overview

Generative AI models have been a big challenge for authors and creators around the world. Responses of any question that a LLM generates, comes from unauthorized access of works from creators who put hours of their time to create that content.

This, not just challenges the genuinity of a work curated in the current age, but also leaves us questioning whether our original work will get the recognition it deserves or not.

During our research, we found one interesting article at Forbes that explained it very well. It mentions two major challenges that generative AI brings for authors and creators -

Potential plagiarism of training data, &
Potential copyright infringement.

It goes on to explain these topics further and also suggests that some technological advancements be made to provide proper citations and credits to the original creators.

However, as we look further into the underlying architecture that these models follow, we will understand why providing citations to every response is not possible.

Architecture, destined to doom

Language models function on encoder-decoder structure with self-attention mechanisms to capture long-range dependencies in the input data.

However, this architecture does not inherently keep track of where each piece of information comes from - it just learns statistical patterns from the data.

So while these models can generate fluent and coherent text by recombining patterns from their training, they don't have an explicit grounding of where each fact or piece of knowledge originates.

This is not necessarily a flaw in the fundamental architecture, but rather a limitation stemming from the self-supervised training approach.

RTRA as the solution

Unlike the pre-existing transformer architecture which relies on running self-attention mechanism over the training data to retrieve response and doesn't keeps a track of where information comes from, RTRA utilizes a chunk-based dynamic learning architecture which means that every time a query is run, the response is generated in small chunks which in-turn, backs the answer and is retrieved from specific sources on the web.
**Explanation of the solution**

Rather than relying on a vast training dataset, RTRA relies on a relatively smaller, more relevant dataset to respond to a query.

All the dataset is well marked and the sources are clearly cited. That’s why, while responding with a final answer, clear citations are available and credits are given to the original authors of the content.

Below are some test runs that our model performed and provided the much-needed citations and credits to the original authors of the content:

1. **Who is Udit Akhouri**:

   We asked our model to respond to this query asking a question about ‘Udit’ and it responded with an answer similar to one provided by any generative AI model.

   However, to notice, it also provides multiple citations of different articles all over the internet that RTRA used to retrieve content about the subject to form the final answer.

   It shows not 1 or 2 but all the sources it utilized as the secondary dynamic dataset.

2. **Are Sam Altman and Microsoft making some supercomputer?**

   The RTRA model was easily able to answer this query in record time along with multiple citations and references it used to generate the response.
Chapter 10

Constraints & Scope of Improvement

We are still carrying out successive researches over RTRA as even though this architecture challenges the currently adapted method of LLM development, there is still a lot of scope for improvement for this technology.

1. **Optimisation and latency reduction**: The architecture is not yet fully optimized and methods like chunking may be devised further to enhance the overall performance of the model. Secondly, too many processes might end up increasing the latency of the system and there is a scope to end some non-necessary processes and decrease latency.

2. **Switching from series scraping to parallel scraping**: The Rtrafactor, specifically, carries out a series-scraping method to scrape content from the internet. If switched to parallel scraping methods, architecture performance can be optimized.

3. **Independency from HuggingFace**: For Rtrafactor, our SDK is entirely dependent on HuggingFace’s inference API. Although it does not impact the RTRA architecture, the Rtrafactor library might not work upon the unavailability of the model or HuggingFace at all.

   On 22nd April, 2024, we recorded this incident where HuggingFace was down and Rtrafactor could also not operate.

   This needs to be fixed; dependency must be eliminated by an API-based calling method in future papers.

4. **A more human model as the base model**: Currently, the base model is utilized majorly for the summarization process. However, the impact and possibility of it is untapped. If we can use a model tuned over human understanding and behavior, a much greater efficiency can be seen in the quality of answers retrieved.

   Overall, the paper marks a breakthrough by introducing a less computation-dependent LLM architecture. However, there is still a lot of scope for improvement and making this architecture better.
Chapter 11

Conclusion

In this paper, we introduced a novel Real-Time Retrieval Argumentation Architecture for Language Models (LLMs). Recognizing the growing demand for dynamic and interactive information retrieval systems, our architecture integrates LLM with real-time argumentation frameworks to enhance the efficiency and effectiveness of information retrieval processes.

While writing the paper, our objective was to lower our dependence on big computational resources to build an efficient language model. This, in turn, laid the foundation for so many discoveries to reduce hallucinations, integrate real-time retrieval of information, let original authors get the credits and citations they deserved, etc.

Not just that, we also focused on pushing a positive impact over the environment through our architecture by lowering the global heat dissipation.

However, like any novel approach, our architecture also presents challenges and opportunities for future research which we elaborated in the constraints & scope of improvement section. Further investigation is needed to optimize the performance of LLMs in real-time retrieval scenarios, explore additional argumentation strategies to enhance argument quality, and evaluate the scalability and robustness of the proposed architecture in large-scale applications.

In conclusion, our Real-Time Retrieval Argumentation Architecture for LLMs represents a promising step towards the development of more intelligent and interactive information retrieval systems by lowering the overall dependency on computational resources.

By leveraging the strengths of LLMs and argumentation frameworks, we believe that our approach holds great potential to revolutionize the way information is retrieved, processed, and presented in real-time, ultimately benefiting both users and applications in various domains.
References


