Adding extra-parametric capabilities to a LLM

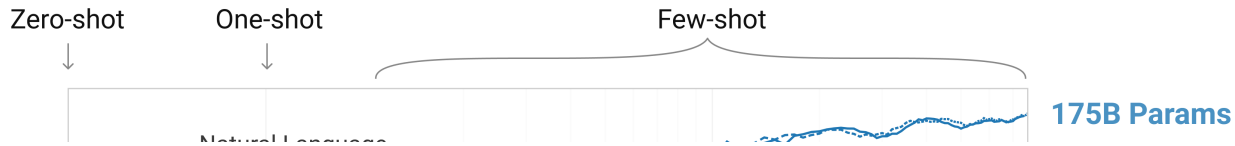
Large Language Models have demonstrated zero and few-shot ability on many tasks.

For example:

- Question Answering
- Mathematical reasoning

Moreover, some of this only emerges when the number of parameters becomes very large.

Few/One/Zero shot learning


Zero-shot
One-shot
Few-shot
175B Params

This suggests that the parameters of the LLM

- encode factual knowledge
    - book knowledge
- encode procedural knowledge
    - how to solve a math problem

Besides consuming many parameters, encoding facts and procedures in parameters have drawbacks

- Factual knowledge is current *only up to the time of training*
- When solving math problems, LLM's are known
    - to get the procedural reasoning correct
    - but make simple arithmetic errors in calculation

A recent trend has been to augment the LLM with capabilities *external* to its parameters

- Factual knowledge obtained from a live source: the Web
- Computational abilities by being able to *execute* programs produced by the LLM

We can illustrate the difference between parametric and non-parametric knowledge by considering the task of Question Answering

- Parametric Knowledge: closed book exam
    - all knowledge acquired and stored before inference time
- Non-Parametric Knowledge: open book exam

Similarly, extra parametric compute can be explained by the analogy about answering a question with a numeric answer that is the solution to an equation

- Extra parametric: using a calculator to evaluate the equation
- Non extra parametric: solving the equation by hand
    - can have the correct equation but incorrect answer due to miscalculation

We refer to

- the first case as *non-parametric knowledge*
- the second case as *extra-parametric compute*

We briefly discuss examples of both types.

Non-Parametric Knowledge: Retriever-Generator Architecture:

The *Retriever-Generator* architecture has two components

- A *Retriever* that is able to gather factual knowledge from an external (non-parametric) source
- A *Generator* that produces the answer

The process is sometimes called *Retrieval Augmented Generation (RAG)*.

Details may be found in

- this paper (https://arxiv.org/pdf/2005.11401.pdf)

There is also a nice online article (https://lilianweng.github.io/posts/2020-10-29-odqa/) describing various approaches in the context of the Question Answering task.

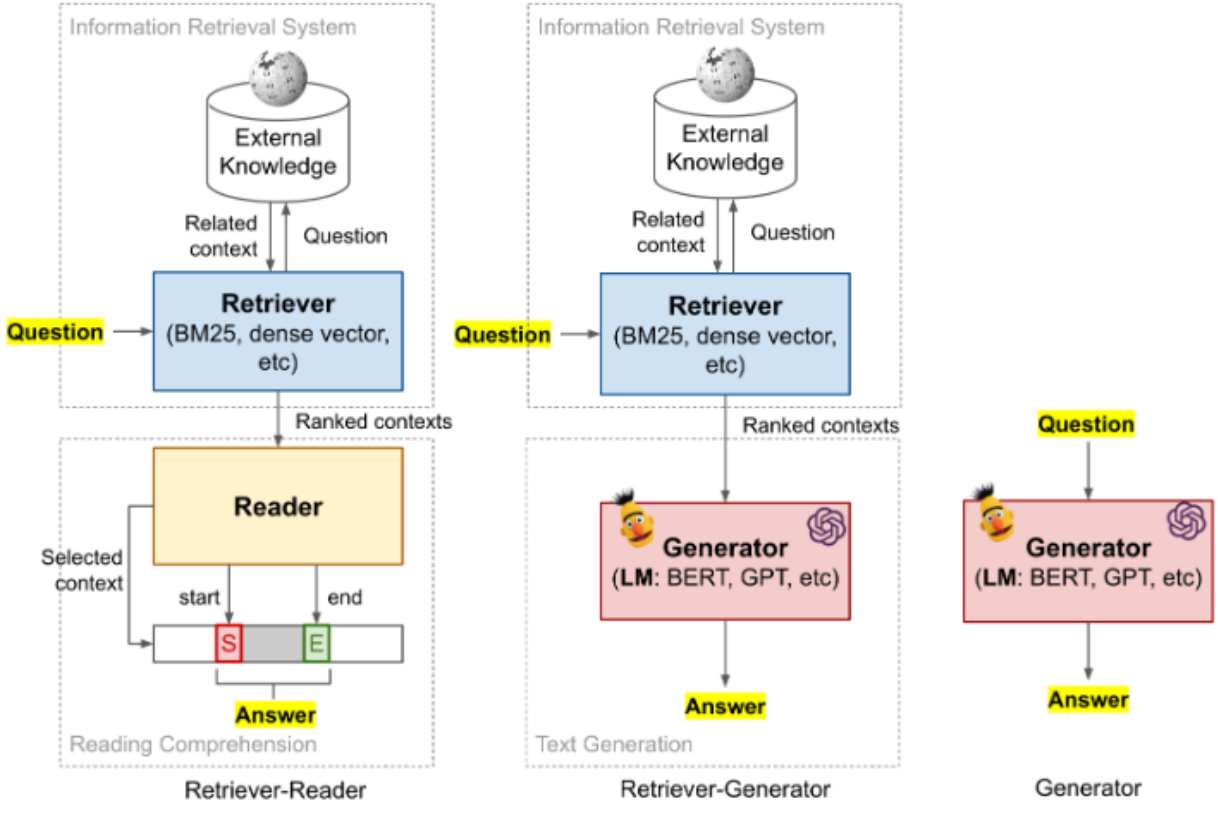


Fig. 1. Overview of three frameworks discussed in this post.

Attribution: https://lilianweng.github.io/posts/2020-10-29-odqa/ (https://lilianweng.github.io/posts/2020-10-29-odqa/)

The architecture in the far-right of the diagram is our standard LLM

- Question as input
- Answer as output

In this case: world knowledge is encoded in the parameters of the LLM.

The Retriever-Generator architecture is depicted in the middle of the diagram

- Question is the input of the Retriever
- The Retriever's output (the "Context") is the input of the Generator
    - e.g., the Top 5 facts retrieved
- The Generator (LLM) outputs the answer, given the context obtained by the Retriever

In this case: world knowledge is *non-paramteric*

The Generator only architecture computes $p(\mathbf{y}|\mathbf{x})$ directly.

The Generator component of the Retriever-Generator architecture

- is *conditioned*
- on both question $\mathbf{x}$ and context $\mathbf{z}$
- in order to produce answer $\mathbf{y}$

$$\text{Generator: } p(\mathbf{y}|\mathbf{x},\mathbf{z}) = p(\mathbf{y}|\mathbf{x},\text{Retriever}(\mathbf{x}))$$

and ultimately $p(\mathbf{y}|\mathbf{x})$

$$
\begin{aligned}
p(\mathbf{y}|\mathbf{x})_{\text{RAG Sequence}} &= \sum_{\mathbf{z}\in \text{Top } K\, p(?|\mathbf{x})} p(\mathbf{z}|\mathbf{x})_{\eta} * p(\mathbf{y}|\mathbf{x},\mathbf{z}) \\
&= \sum_{\mathbf{z}\in \text{Top } K\, p(?|\mathbf{x})} p(\mathbf{z}|\mathbf{x})_{\eta} \prod_{i=1}^{N} p(\mathbf{y}_i|\mathbf{x},\mathbf{z},\mathbf{y}_{(1:i-1)}) \qquad \text{since } 1
\end{aligned}
$$

Note

The paper (https://arxiv.org/pdf/2005.11401.pdf) contrasts

The Retriever-Reader architecture (far left of the diagram)

- is similar to the Retriever-Generator
- but uses a Reader rather than Generator to output the answer
    - The answer produced by the Reader is a sub-string of the retrieved facts
    - identified by a start/end position

The world knowledge is non-parametric (just like the Retriever-Generator)

- but the answer format is much more restricted

Retrieve-Generator: training

Both the Retriever and the Generator are parameterized.

When the Generator is a LLM

- a pre-trained LLM may be used
- and its parameters "fine-tuned"
- not trained from scratch

But the Retriever's parameters need to be learned from scratch via training

- depending on how the Retriever obtains external knowledge.
- how to generate a "query" to the Knowledge Source

Here is a diagram

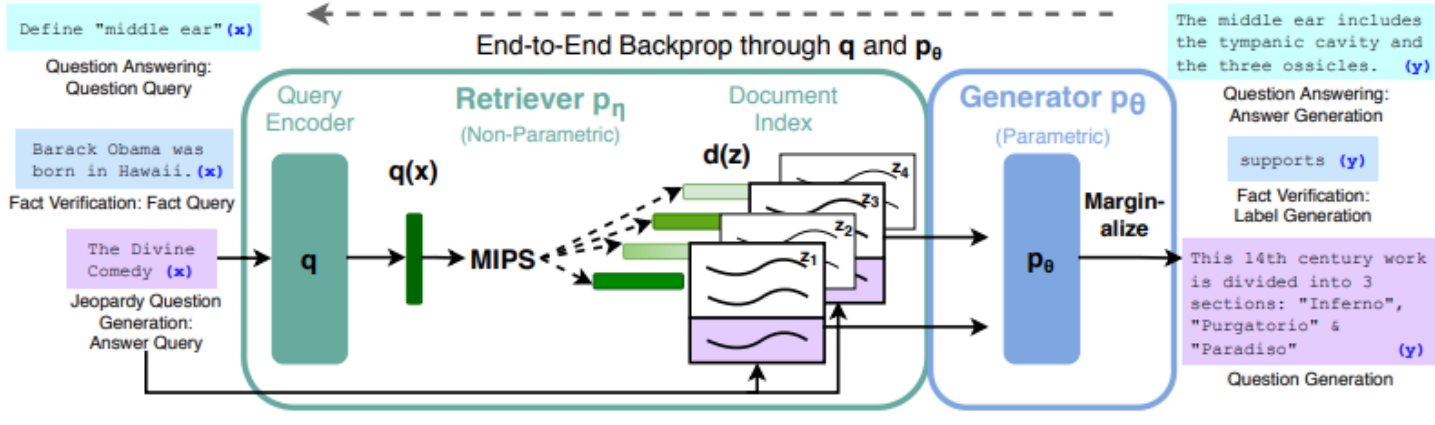


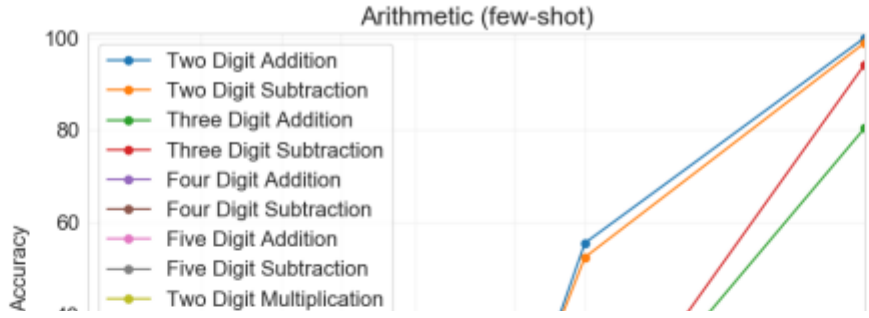
Figure 1: Overview of our approach. We combine a pre-trained retriever (*Query Encoder* + *Document Index*) with a pre-trained seq2seq model (*Generator*) and fine-tune end-to-end. For query x, we use Maximum Inner Product Search (MIPS) to find the top-K documents z_i. For final prediction y, we treat z as a latent variable and marginalize over seq2seq predictions given different documents.

Attribution: https://arxiv.org/pdf/2005.11401.pdf#page=2 (https://arxiv.org/pdf/2005.11401.pdf#page=2)

Extra Parametric Compute

The LLM has been shown to have *some* ability to perform math.

However: this seems to be one of the capabilities that "emerge" only in large models.


Arithmetic (few-shot)
Accuracy
100
80
60
Two Digit Addition
Two Digit Subtraction
Three Digit Addition
Three Digit Subtraction
Four Digit Addition
Four Digit Subtraction
Five Digit Addition
Five Digit Subtraction
Two Digit Multiplication

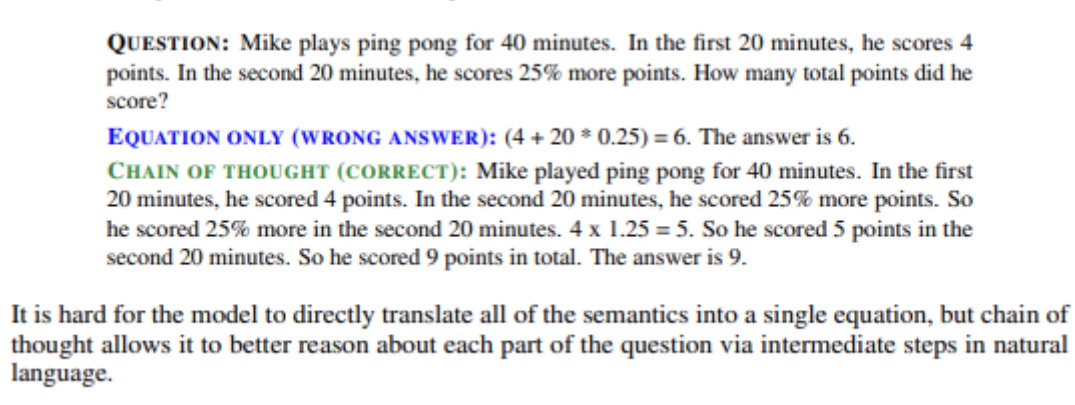
The above chart was for a simple arithmetic operation.

LLM's have also demonstrated some ability on multi-step reasoning problems.

The ability to solve multi-step problems is improved by

- Chain of Thought prompting (https://arxiv.org/pdf/2201.11903.pdf)
    - prompting the model to show the solution "step by step"
- Show your work prompting (https://arxiv.org/pdf/2112.00114.pdf)

Both these methods guide the LLM to produce the answer in small steps, rather than all at once.



QUESTION: Mike plays ping pong for 40 minutes. In the first 20 minutes, he scores 4 points. In the second 20 minutes, he scores 25% more points. How many total points did he score?

EQUATION ONLY (WRONG ANSWER): (4 + 20 * 0.25) = 6. The answer is 6.

CHAIN OF THOUGHT (CORRECT): Mike played ping pong for 40 minutes. In the first 20 minutes, he scored 4 points. In the second 20 minutes, he scored 25% more points. So he scored 25% more in the second 20 minutes. 4 x 1.25 = 5. So he scored 5 points in the second 20 minutes. So he scored 9 points in total. The answer is 9.

It is hard for the model to directly translate all of the semantics into a single equation, but chain of thought allows it to better reason about each part of the question via intermediate steps in natural language.

Attribution: https://arxiv.org/pdf/2201.11903.pdf#page=19

In multi-step math problems, LLM's

- sometimes generate the correct sequence of solution steps
- but fumbles the math (failing to carry the digit)

The CoT paper (https://arxiv.org/pdf/2201.11903.pdf#page=27) calls these "calculator errors"

- They report that 34% of examples demonstrated calculator errors
    - including those with incorrect reasoning

LLM's perform poorly on a simple mathematical task:

- output the sum of the two inputs, **plus 1** (https://github.com/google/BIG-bench/tree/main/bigbench/benchmark_tasks/modified_arithmetic#example)
- using few-shot learning

Example

The following is an example of the subtask `three_digit_addition_plus_one`.

Prompt:

```
In the following lines, the symbol -> represents a simple mathematical operation.
100 + 200 -> 301
838 + 520 -> 1359
343 + 128 -> 472
647 + 471 -> 1119
64 + 138 -> 203
498 + 592 ->
```

Answer:

```
1091
```

GPT-3 has been reported to have **zero** accuracy on this task.

Even with explicit instruction (as above) the model performs poorly on "looping"

- What is the 50^{th} number in the Fibonacci sequence

Figure 1: Comparison between Chain of Thoughts and Program of Thoughts.

On the other hand, LLM's have been shown to have the ability to generate programs.

Program of Thoughts Prompting (https://arxiv.org/pdf/2211.12588v3.pdf) is a method

- where the LLM is trained (few-shot) to produce *programs* as output
- the programs are *executed* by an external module

In other words

- the LLM tries to get the step by step process correct
- and uses an external "calculator" to avoid "doing the math"

Programs of Thought prompting is like Chain of Thought Prompting

- prompt asks for a "step by step" answer
- the exemplars encourage *descriptive variable names*
    - improves the ability to generate a correct program ?

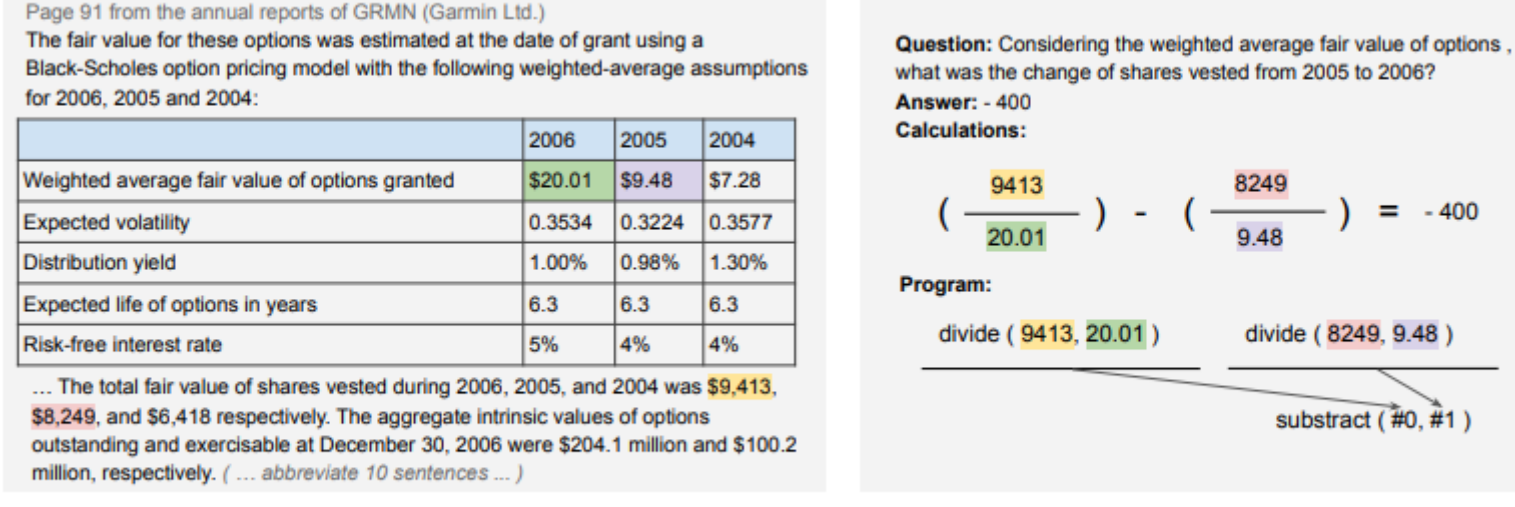
Some recent advances on solving multi-step quantitative reasoning problems can be found in

- Minerva: Solving Quantitative Reasoning Problems with Language Models (https://ai.googleblog.com/2022/06/minerva-solving-quantitative-reasoning.html)

FinQA: Financial Question Answering

The [FinQA dataset (https://arxiv.org/pdf/2109.00122.pdf)](https://arxiv.org/pdf/2109.00122.pdf) was created to test the ability of a model

- to perform Question Answering in the domain of Finance
- demonstrating the reasoning behind the answer
    - by outputting a program to calculate the ansser



Page 91 from the annual reports of GRMN (Garmin Ltd.)

The fair value for these options was estimated at the date of grant using a Black-Scholes option pricing model with the following weighted-average assumptions for 2006, 2005 and 2004:

	2006	2005	2004
Weighted average fair value of options granted	$20.01	$9.48	$7.28
Expected volatility	0.3534	0.3224	0.3577
Distribution yield	1.00%	0.98%	1.30%
Expected life of options in years	6.3	6.3	6.3
Risk-free interest rate	5%	4%	4%

... The total fair value of shares vested during 2006, 2005, and 2004 was $9,413, $8,249, and $6,418 respectively. The aggregate intrinsic values of options outstanding and exercisable at December 30, 2006 were $204.1 million and $100.2 million, respectively. *(... abbreviate 10 sentences ...)*

Question: Considering the weighted average fair value of options , what was the change of shares vested from 2005 to 2006?

Answer: - 400

Calculations:

$$\left(\frac{9413}{20.01}\right) - \left(\frac{8249}{9.48}\right) = -400$$

Program:

divide (9413, 20.01), divide (8249, 9.48), substract (#0, #1)

Figure 1: An example from FINQA: The system needs to learn how to calculate the number of shares, then select relevant numbers from both the table and the text to generate the reasoning program to get the answer.

The authors demonstrate a Retriever-Generator model for the task.

- Retriever: External Knowledge Source to store Financial Reports on companies
- Generator: outputs a "calculator program"

Here we see both forms of extra-parametric capabilities integrated with a LLM.

The authors of the FinQA dataset have also created the ConvFinQA dataset (https://arxiv.org/pdf/2210.03849.pdf)

- *conversational* question answering
- a follow-up question can reference the answer to a previous question

Conclusion

There are some obvious benefits to adding Extra Parametric capabilities to a LLM

- The LLM can be smaller
    - knowledge (factual and procedural) stored outside of parameters
- New "skills" can be added via exemplars demonstrating calls to a new library
    - Derivative pricing library

In [2]:
```
print("Done")
```

```
Done
```