Retrieval Augmented Generation – The Process of Improving Output Using External Resources

Sebastian Newberry Endri Islami Parthiv Gajula

April 28, 2025

Abstract

RAG stands for Retrieval Augmented Generation, and involves a complex process which starts with retrieving relevant documents. These documents can be broken up into many different forms and used in many different ways depending on the task and desired result that the user expects. Overall, there are two major categories that retrieval tasks can be broken up into. The first is to find and return part of a document that has an exact answer to a user input query, the second is to find a document that answers a user query, and generate a response by passing that document into an LLM as context. Both of these problems become increasingly difficult as the number of documents we have to parse through increases. In order to solve these problems as they become more complex, neural networks are used with retrievers to find and parse relevant documents. Embeddings of documents in varying sizes are stored in a vector database. This allows retriever algorithms to compute similarity scores between query and document embeddings and extract documents that are relevant. In algorithms like REALM, a BERT reader is used to re-rank selected documents for a specific query. Once documents have been ranked, they can be fed into a GPT model to generate text based on the context that is passed from a retriever.

1 Evaluation Method

In this section, we explain the data, metrics, and the baseline learning algorithms used to examine the performance of in-context learning of Transformers.

To create the training and test prompts $P = (x_1, f(x_1), \ldots, x_k, f(x_k + 1))$ these two steps are followed: First, sample a random function f from the class over D_F. Second, generate a set of random inputs $x_1, \ldots, x_{k+1}$ drawn independently from D_x and evaluate f on these inputs.

To have a better idea of the performance of the model, we compare the Transformer performance with three well-known learning algorithms:

- **Least Square Estimator:** In the Least Square method, which is the optimal estimator for this problem, we minimize the sum of squared residuals:

$$\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

  where $\hat{y}_i$ is the predicted value. The closed-form solution is:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

- **N-Nearest Neighbors:** For a given x, we average the y_i values of the N nearest neighbors of x_{query} using Euclidean distance:

$$\text{Distance}(p, p') = \sqrt{\sum_{i=1}^{n} (p_i - p'_i)^2}$$

- **Averaging:** Estimate v by averaging the values $y_i x_i$.

The evaluation metric is Squared Error (SE), defined as:

$$\text{SE} = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

2 Experimental Results

We now dive into the implementation and evaluation of the model.

2.1 Training a Transformer

To train a Transformer, we follow the general structure: given functions f from distribution D_F and inputs from D_x, we sample prompts

$$P = (x_1, f(x_1), \ldots, x_k, f(x_k), x_{k+1})$$

and train a model to anticipate $f(x_{k+1})$.

Method	**Accuracy**
Baseline	85.2%
Our Model	**89.7%**

Table 1: Accuracy comparison

This paragraph will wrap around the table. You can continue writing text here and it will naturally flow around the right-aligned table. This is useful when you want a compact layout, such as in research posters or narrow-column papers.