TLDR: it is a finite mixture model with a bit of handwaving.

Retrieval-Augmented Generation (RAG) is a popular technique to improve the quality of generated texts from language models. According to OpenAI (link), RAG can be quite promising if you have additional documents to provide as contexts relevant to your question asked to a language model.

There are many online tutorials on RAG, e.g. by OpenAI, LangChain and NVDIA, most of which use flowcharts/diagrams and sample code snippets. These tutorials are useful for practical purposes, but they don't reveal what is actually going on behind the scenes.

I read the original paper by Lewis et al. (2020) who first experimented with RAG. Below are my takeaways and thoughts. This note mainly covers Section 2 (Methods) of the paper.

Suppose we have a generative language model with parameters θ. A question encoded as x is asked, and the model generates an answer encoded as y (with length N) based on the following distribution.

$$p_\theta(y|x) = \prod_i^N p_\theta(y_i|x, y_{1:i-1})$$

In other words, the next token generated depends on the model, the question, and all preceding tokens.

The generated answer y might not be satisfactory if the model parameters θ are trained on some general knowledge but the question x is domain-specific (e.g. medicine, law, finance).

Now also suppose we have a collection of domain-specific passages (which could be documents or chunks of them) encoded as $\{z_j\}_{1\le j\le M}$. We could consider finetuning the model with these domain-specific passages so that the updated parameters, say θ', can incorporate domain-specific knowledge, hence the generated answer $p_{\theta'}(y|x)$ will hopefully be of higher quality.

RAG is an alternative to the potentially costly process of model fine-tuning in practice. Instead of updating the parameters θ, we search for a subset of K domain-specific passages $\{z_j\}_{1\le j\le K}$ that are most relevant to the question x. Then, we augment a potentially better answer by incorporating such relevant information.

Effectively, the passages $\{z_j\}_{1\le j\le M}$ can be viewed as latent variables. By applying properties of conditional probabilities,

$$p_{\text{RAG}}(y|x) = \sum_{j=1}^M p_\phi(y|x, z_j)p_\eta(z_j|x) \approx \sum_{j=1}^K p_\phi(y|x, z_j)p_\eta(z_j|x)$$

where $p_\eta(z_j|x)$ is called the *retriever* and $p_\phi(y|x, z_j)$ the *generator*.

Before diving into the individual terms, notice that the number of retrieved passages K can be equal to the total number of domain-specific passages M, in which case the second step of approximation is not needed. In other words, we are leveraging all available additional information for generating the answer. However, this is typically not the case in practice because M can be huge (e.g. the authors used a 2018 snapshot of Wikipedia and divided the texts into disjoint chunks of 100 words).

The *retriever* $p_\eta(z_j|x)$ serves to measure the relevance (i.e. some kind of distance) between the question x and each passage z_j. The authors based their retriever on the Dense Passage Retrieval in Karphkhin et al. (2020) such that

$$p_\eta(z_j|x) \propto \exp(d(z_j)^T q(x))$$

where $d(z_j)$ and $q(x)$ are representations of z_j and x, respectively, based on the same encoder model with parameters η. The authors note that searching for the top K most relevant passages can be done in approximately sub-linear time (see also MIPS), which is feasible for practical applications.

The *generator* $p_\phi(y|x, z_j)$ is effectively a language model with parameters ϕ that generates the answer y based on both z_j and x. Technically, it can be any language model and, ideally, one that is (somehow) fine-tuned specifically on the collection of domain-specific documents $\{z_j\}_{1\leq j\leq M}$.

For the generator, I find two hand-wavy things (one in the paper, and one in practice): sequence concatenation and model training/finetuning.

Sequence Concatenation

Most generative language models are sequence-to-sequence, that is, the model input is one single sequence and so is the model output. The authors use a simple concatenation of x and z_j to for this single sequence input, so $p_\phi(y|x, z_j)$ is basically just $p_\phi(y|\{x, z_j\})$ if we use { and } to denote concatenation.

I understand why this choice is made from a technical perspective, and maybe there will be better ways to combine x and z_j in the future. However, simple concatenation might not make sense from a layperson's perspective, because the retrieved passage z_j may not necessarily make sense to a real human (e.g. it might be a broken-up paragraph, chunked partial tables from a document, or highly irrelevant contents).

One might wonder why we need to append useless information when asking questions to the model. With a bit of understanding of finite mixture model, I would expect (or hope?) that irrelevant/useless retrievals get assigned rather low weights by the retriever $p_\eta(z_j|x)$, thus contributing very little to the combined result generated from $p_{\text{RAG}}(y|x)$.

Model Training/Finetuning

In the paper, the authors actually train both the retriever and the generator at the same time. They provide pairs of (x, y) without explicitly specifying which passages z_j are the most relevant. Consequently, the parameters η and ϕ are both updated based on the question x and y. This is very similar to training a finite mixture model because we typically don't know which latent component actually generated the observed data.

However, this goes back to the problem of model finetuning which is what we wanted to circumvent in practice in the first place. I highly doubt that commercial solutions of RAG (such as API endpoints provided by tech companies) rely on any kind of fine-tuned model as the generator, while they probably have some solutions for the retriever.

I suspect the generator is simply $p_\theta(y|\{x, z_j\})$ in practice. In other words, we concatenate the original question with whatever information is retrieved from domain-specific passages, and then feed this long message into the general language model that we started off with.

So what actually is it that makes RAG generate better answers? I think there are two factors: the additional information z_j that is searched and fed into the generator as part of the prompt, and the nature of mixture models with allows the retriever to put more weights on more relevant passages.