

# [DIFF] Executive summary

In the previous pipeline, I developed a question answering system based on the retrieval augmented generation (RAG) framework to answer questions using my extensive knowledge base of 1000 Markdown notes. RAG offers advantages over fine-tuning, particularly for a growing knowledge base and limited computational resources. The system combines knowledge retrieval and large language model (LLM) generation, leveraging concepts like embedding, vector databases, context windows, transformer architecture, and attention mechanisms.

In this pipeline, I assessed the RAG system's performance using a test set of 14 question-answer pairs and four metrics (semantic similarity and three LLM evaluations). Overall, the system performed strongly, achieving a high semantic similarity score of 0.89, indicating alignment between model-generated and human answers. Faithfulness scored 0.92, confirming the model's consistency in grounding responses in context without hallucination. Relevancy was perfect at 1.0, underscoring the system's ability to provide contextually relevant responses. However, there is room for improvement in the correctness metric, averaging around 0.580, suggesting occasional deviations from human reference answers. Notably, task-specific analysis revealed variations, with 'summarize' questions excelling in correctness (0.65), while 'infer' questions scored the lowest (0.38), potentially due to their larger needs of contextual information. Future enhancements will prioritize tasks crucial for knowledge base querying, specifically 'summarize' and 'extract'.

Minor updates include: review the explanation for attention mechanism (add multi-head and cross-attention), handle exceptions (avoiding OpenAI API's RateLimitError with exponential backoff), engineer prompts, add more examples, and more visual aids.



## Logistic notes

- New sections are tagged with [NEW], and sections that are updated are tagged with [DIFF]
- **I submitted this assignment 1 day later than the deadline because I have Incomplete approved for Dec 17.**

## Problem: Answer questions about my knowledge base

Over the last three years, I've been curating a personal knowledge bases of my notes in Obsidian, academic assignments in PDFs, and the content from textbooks. Considering the HC #scienceoflearning , I should have used spaced repetition to cement crucial knowledge.

Admittedly, I usually only review my notes when forgetfulness knocks. Further, navigating my evolving and extensive knowledge database is becoming more of a challenge.

In contrast, the LLMs, especially ChatGPT, have consistently exhibited agility in responding to inquiries. However, their limitations surface when confronted with intricate domains like mathematics, compelling me to seek supplementary information from alternative sources, a gap that my meticulously maintained notes aim to fill.

Recognizing the ability to understand language of LLMs (e.g. quickly extracting information and summarizing text) and the wealth and details of my own knowledge base, I want to combine their strengths. In this assignment, I am building a question answering system that can answer questions about my knowledge base. In the next assignment, I will improve upon this system.

By leveraging the language understanding prowess of LLMs and the detailed insights within my knowledge base, this question-answering system is poised to offer precise and contextually rich responses, without me going over many notes to find the answer.

## Data collection

To focus on technical details of the models, I will use a subset of my knowledge base, which is a collection of my notes (Markdown files) from the last three years. I collected the assignments and textbooks (PDFs) but will not use them in this assignment.

The dataset is loaded and counted below.

```
In [1]: from pathlib import Path

DATA_DIR = Path("./data/raw")

# Count the number of .pdf and .md files
pdf_count = sum(1 for _ in DATA_DIR.rglob("*.pdf"))
md_count = sum(1 for _ in DATA_DIR.rglob("*.md"))

print(f"Assignments & Textbooks: {pdf_count}\nNotes: {md_count}")

Assignments & Textbooks: 60
Notes: 1009
```

## Data Ingestion & Preprocessing

In this section, I perform two main steps: 1) ingest and clean markdown files, and 2) load them inside LlamaIndex.

1. Ingest and clean markdown files: I modified and modularized the code from the previous assignment, which is imported as `Markdown2Text` class. The code strips off

markdown elements and native Obsidian elements, and returns the cleaned text.

2. Load them inside LlamaIndex: LlamaIndex is a framework aimed at helping build applications with large language models (LLMs) by providing tools for data ingestion, structuring, and many more with various application frameworks. Loading my data as the `Document` object is the first step to use LlamaIndex later on. Each `Document` object contains the cleaned text and metadata (file name, domain).

```
In [2]: from typing import List, Dict, Optional

from llama_index.schema import Document

# Modify the Markdown2Text classes from the previous pipeline
from src.processing.strip_markdown import Markdown2Text

# Load the .md files
md_loader = Markdown2Text(
    markdown_dir=DATA_DIR,
)
md_docs: List[Document] = md_loader.load_markdown_dir(return_documents=True)

# validation checks
assert (len(md_docs) <= md_count) and (len(md_docs) > 0)
assert all([isinstance(doc, Document) for doc in md_docs])
assert all([hasattr(doc, "text") for doc in md_docs])
assert all([hasattr(doc, "metadata") for doc in md_docs])
assert all([doc.text for doc in md_docs])

# print the first document
print(f"Filled Notes: {len(md_docs)}")
print(f"Preview a document: {md_docs[0].metadata['file_name']}")
print(md_docs[0])
```

Filled Notes: 985

Preview a document: Transformer Study Guide

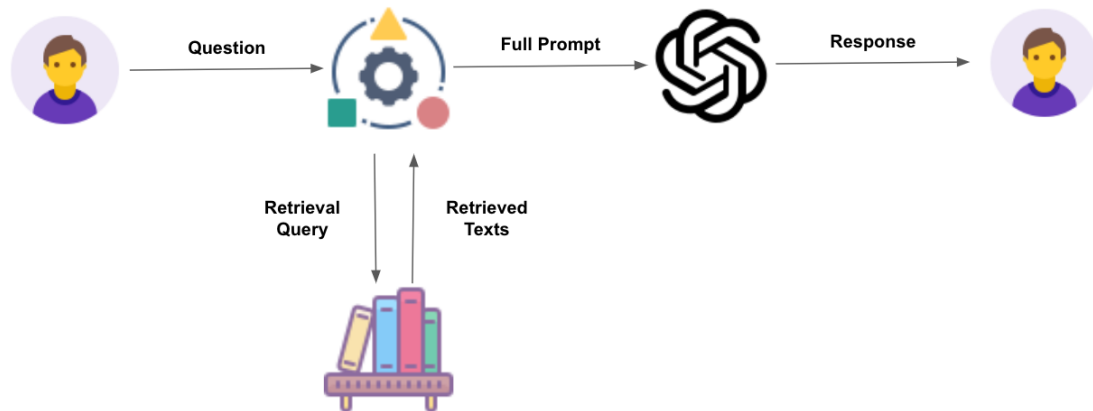
Doc ID: 070d4faa-2f27-4b5a-80ef-74bd8ce5c054

Text: #ML/NLP Generator Building Block 1: Transformer basics As we learned in class, Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) have some limitations. They are sequential models, which means they process input data one element at a time, making them slow and computationally expensive. Additionally, they struggle to handle long-...

## Approach: Retrieval-based Question Answering System

Retrieval augmented generation (RAG) is a powerful AI framework that enhances the quality of responses generated by large language models (LLMs) by grounding the model on external knowledge. This framework is shown to be particularly useful for question answering systems as it combines the strengths of a knowledge base with those of an LLM to

- produce more accurate and contextually relevant responses, especially for questions outside the LLM's trained domains
- address the limitations of static training data and enable the model to generate more reliable and up-to-date responses.



At a high level, RAG works by retrieving documents from an external dataset, which serve as the context for answering a given question. The framework consists of two main components

- Retriever: responsible for finding and fetching the most relevant information to the question, to feed to the LLM for generating the response
- Generator (transformer-based LLM): responsible for generating the response based on the original question, the retrieved information (additional context), and the instructions from prompt.

Without retriever, the process is back to the vanilla: the LLM generates the response based on the prompt from the user only.

In this section, I will attempt to explain these two components from the ground up, assuming the knowledge of Linear Algebra (vectors), and RNNs.

## RAG over Fine-tuning

Before diving into the details of RAG, it is important to understand the difference between fine-tuning and RAG.

- Fine-tuning allows the model to learn domain-specific knowledge and adjust its behavior, writing style, or responses to specific nuances, tones, or terminologies. It is effective for tasks requiring strong domain affinity. However, exclusive fine-tuning may not be practical for applications that require responses to be synced with a dynamic pool of information, as the model will need regular retraining for evolving data
- RAG is proficient in data retrieval and can provide access to dynamic external data sources. It is ideal for applications that heavily rely on external data sources, as it

continuously queries external sources, ensuring that the information remains up-to-date without frequent model retraining.

Therefore, while both of these frameworks are useful, RAG is more suitable for my application considering that my knowledge base is dynamic and constantly evolving. Further, a big constraint is that I do not have the datasets and computational resources to fine-tune a large language model.

However, RAG and fine-tuning are not mutually exclusive. Fine-tuning a RAG model can further improve the quality of responses, especially for domain-specific questions. I will explore that option in the next assignment.

## Retriever: LlamaIndex's embedding-based retriever

### Retriever Building Block 1: Embedding & Vector database

Unlike humans, machine learning models, including LLMs, do not have the ability to read and understand natural language text. Instead, they require numerical inputs to operate. To convert text into numbers, we use embedding to map smaller chunks of text (e.g. words, sentences, paragraphs) into dense, high-dimensional vectors of real numbers. Embedding is useful because it preserves the semantic relationships between words, which is crucial for the LLM to understand the context of the text.

Embedding is a mapping from a discrete space (e.g. words, sentences) to a continuous space (e.g. vectors), learned after training on a large corpus of text. The embedding space is a vector space, where each dimension represents a feature of the word. For example, the word "spider" can be represented as a vector of 2 dimensions (-1.53, 0.41) below, where each dimension represents a feature of the word "spider" (e.g. negative/positive, concrete/abstract). Notice how the embedding of the word "spider" is similar to that of the word "snake", but different from that of the word "puppy" (e.g. the words "spider" and "snake" are similar in meaning and usually more negative, while the word "puppy" is usually more positive). In general, the distance between vectors corresponds to the similarity between the corresponding text data.

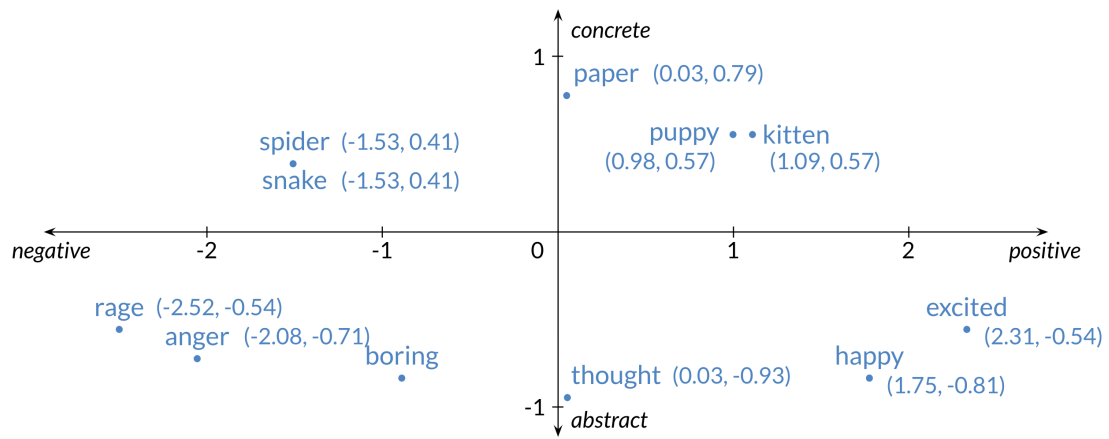


Figure 2. Embedding of the words "spider", "snake", and "puppy" in a 2-dimensional space.

Source: [Natural Language Processing Specialization Course 2](#)

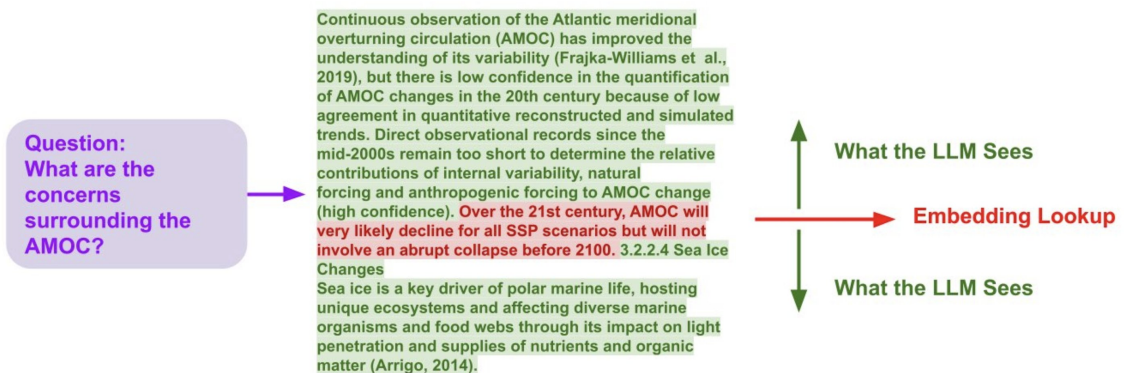
The resulting word or sentence vectors are then concatenated or averaged to produce a single vector representation of the entire text data.

In the context of RAG, embedding is used to convert both the source data and the user queries into numerical representations that can be compared and matched by the retrieval model. The retrieval model then returns the most relevant documents to the user query. Usually to conserve more information, the sentence embedding is used.

And the vector database is a collection of sentence vectors from the source data.

## Retriever Building Block 2: Context Window in Dense Passage Retrieval

Embedding, which is typically used for smaller text chunks, may not provide sufficient context for the LLM. Instead of embedding just one sentence at a time to the vector base, we can embed a window of text around sentence (e.g. three sentences including the main one).



Although there might be only one relevant sentence, the context window (three sentences) provides more information to the LLM. Source: [Jerry Liu](#)

## Retriever Building Block 3: Similarity search

As mentioned above, the retrieval model returns the most relevant documents to the user query. To do so, it compares the user query (a given embedding vector) with the vector database and searches for the most similar documents (embedding vectors). The similarity between two vectors  $\vec{u}$  and  $\vec{v}$  can be measured by dot product

$$\vec{u} \cdot \vec{v} = \vec{u}^\top \times \vec{v}$$

or cosine similarity (more commonly used). Cosine similarity is the dot product of the vectors over the product of vector magnitudes.

$$\text{sim}(\vec{u}, \vec{v}) = \cos(\theta) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|}$$

It has values between -1 and 1, where 1 means the vectors are identical (best), 0 means the vectors are orthogonal, and -1 means the vectors are opposite (worst).

## Retrieval Advanced 1: Embedding-based Reranking

While retrieved documents are usually relevant to the user query, they may not be ordered by the most relevant. To improve the performance, we introduce reranking that is trained to rank the documents based on their relevance to the question, ensuring that the most relevant information is presented to the LLM for response generation. The main steps are as follows:

1. Train the reranking model on a dataset of question-document pairs (where the question represents the user query and the document is the retrieved document) and learn to assess the relevance of documents to specific queries. This training enables the model to discern and prioritize the most relevant documents for a given query.
2. Once the initial retrieval phase is complete, the reranking model scores the retrieved documents based on their relevance to the user query. Subsequently, it reorders the documents, promoting the most relevant ones to higher positions in the ranking.
3. Reintegrate the reranking model into the retrieval model, which now returns the most relevant documents to the user query, ordered by relevance.

## Retrieval last note: LLM-based reranking

Above, we have discussed the retrieval model that uses embedding to convert text into dense vectors and then compares the user query with the vector database to find the most similar documents. However, we can also use LLMs for reranking to return documents that are more relevant (by human standard) than embedding-based method, with the tradeoff being much higher latency and cost.





## Generator: GPT-3.5

### Generator Building Block 1: Transformer basics

As we learned in class, Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) have some limitations. They are sequential models, which means they process input data one element at a time, making them slow and computationally expensive. Additionally, they struggle to handle long-range dependencies (i.e. capture the relationship between words/phrases/sentences far apart), which are common in natural language processing tasks.

Unlike RNNs and LSTMs, the Transformer model ([Vaswani, et al., 2017](#)) uses self-attention to process input data in parallel, making it faster and more efficient. Self-attention allows the model to focus on different parts of the input sequence and capture long-range dependencies more effectively. The Transformer model also uses positional encoding to preserve the order of the input sequence.

At a high-level, Transformers uses the encoder-decoder architecture, where the encoder processes the input sequence and the decoder generates the output sequence. Later simplified Transformer was shown to achieve great performance in language modeling tasks, like in encoder-only BERT or decoder-only GPT.

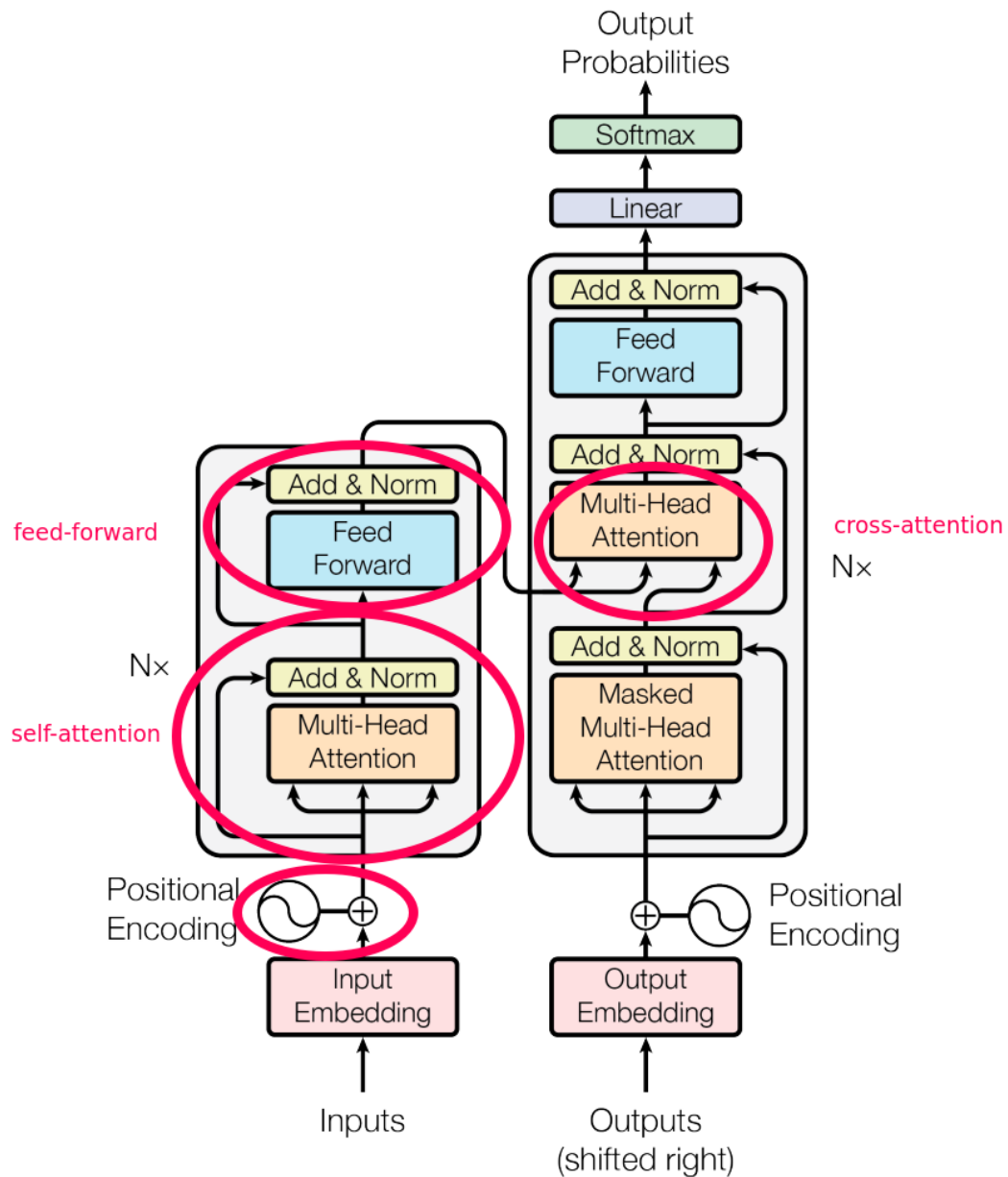


Figure 1: The Transformer - model architecture.

Transformer model architecture. Source: (Vaswani et al., 2017)

## Generator Building Block 2: Encoder-Decoder Architecture Deep Dive

### Input Embedding & Output Embedding

Each input sequence (e.g. sentences) is first converted into a sequence of vectors using an embedding layer. The embedding layer maps each word in the input sequence to a vector in the embedding space. The output of the embedding layer is then passed to the encoder.

The Input Embedding  $\mathbf{X} \in \mathbb{R}^{L \times d}$  has each element mapped into an embedding vector of shape  $d$ , same as the model size, and has another dimension of segment length of input

sequence  $L$ .

The Output Embedding  $\mathbf{Y} \in \mathbb{R}^{L \times d}$  is obtained from the encoder component.

### Positional Encoding (Sinusoidal)

The Transformer does not inherently capture the sequential order of input tokens like recurrent neural network, as it processes them in parallel. To incorporate positional information, positional encodings are added to the input embeddings. These positional encodings provide the model with information about the relative or absolute position of tokens in the input sequence.

For a given position  $\text{pos} = 1, \dots, L$  and dimension  $i = 1, \dots, d$ , the positional encoding  $PE(\text{pos}, i)$  is computed using sine and cosine functions:

$$\begin{aligned} PE(\text{pos}, 2i) &= \sin\left(\frac{\text{pos}}{10000^{2i/d}}\right) \\ PE(\text{pos}, 2i + 1) &= \cos\left(\frac{\text{pos}}{10000^{2i/d}}\right) \end{aligned}$$

- $d$  = the dimensionality of the model
- $L$  = the input sequence length

This results in a positional encoding matrix  $\mathbf{P} \in \mathbb{R}^{L \times d}$ , where the  $i$  row  $\mathbf{p}_i$  is the positional encoding for input element  $\mathbf{x}_i$ .

### Multi-head Attention (Encoder)

The encoder consists of multiple layers, each of which consists of a multi-head self-attention layer and a feedforward neural network. The multi-head self-attention layer allows the model to focus on different parts of the input sequence for each head. The attention mechanism calculates attention scores for each position based on its relation to other positions. It uses  $h$  attention heads to capture diverse information.

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \dots, \text{head}_h) \cdot W_O \\ \text{head}_i &= \text{Attention}(Q \cdot W_i^Q, K \cdot W_i^K, V \cdot W_i^V) \end{aligned}$$

- $Q$  = the current position-word vector in the input sequence
- $K$  = all the position-word vectors in the input sequence
- $V$  = all the position-word vectors in the input sequence
- $W_{Qi}, W_{Ki}, W_{Vi}, W_O$  are learned linear projection for the  $i^{\text{th}}$  query, key, value, and output transformations, respectively.

I will elaborate on the attention mechanism in the next section.

### Add & Norm

This is a residual connection followed by layer normalization. The residual connection allows the model to learn identity functions, which can help flow the information throughout the deep neural networks. Layer normalization normalizes the output of the previous layer by subtracting the mean and dividing by the standard deviation.

For an input  $X$  and a sub-layer output  $\text{SubLayer}(X)$ , it is computed as follows:

$$\text{AddNorm}(X, \text{SubLayer}(X)) = \text{LayerNorm}(X + \text{SubLayer}(X))$$

## Feedforward Neural Network

The feed-forward layer consists of two linear transformations with a ReLU activation in between. It introduces non-linearity by applying a feedforward neural network to each position independently

For an input  $X$ , this layer is computed as follows: 
$$\begin{aligned} \end{aligned}$$

$$\begin{aligned} \text{FFN}(X) &= \text{ReLU}(X \cdot W_{\{1\}} + b_{\{1\}}) \cdot W_{\{2\}} + b_{\{2\}} \\ &= \max(0, X \cdot W_{\{1\}} + b_{\{1\}}) \cdot W_{\{2\}} + b_{\{2\}} \end{aligned}$$

Here,  $W_1, W_2, b_1, b_2$  are learnable parameters

## Masked Multi-head Attention (Decoder)

This is similar to multi-head attention in the encoder, except that it uses a mask to prevent the decoder from looking at future positions in the sequence during training. It makes the decoder predict the next positions by previous positions learned from encoder only.

## Multi-head Attention (Decoder)

The decoder uses multi-head attention, similar to the encoder, to focus on relevant parts within the decoder input (previous positions generated by the decoder). However, it also attends to the encoder's output to obtain information about the input sequence.

## Linear

This is a linear transformation layer that projects the output of the decoder to the output vocabulary size

$$\text{Linear}(X) = X \cdot W + b$$

## Softmax

This softmax activation layer that converts the output of the linear layer into a probability distribution over the output vocabulary.

$$\text{Softmax}(X) = \frac{\exp(X)}{\sum_{i=1}^V \exp(X_i)}$$

## [DIFF] Generator Building Block 3: Attention mechanism

([Raschka, 2023](#)) ([Weng, 2023](#))

**Attention** is a mechanism in neural network that a model can learn to make predictions by selectively paying attention to a specific parts of the input. The amount of attention is quantified by learned weights and thus the output is usually formed as a weighted average.

### Self-attention

**Self-attention** is a type of attention mechanism where the model makes prediction for one part of a data sample using other parts of the observation about the same sample.

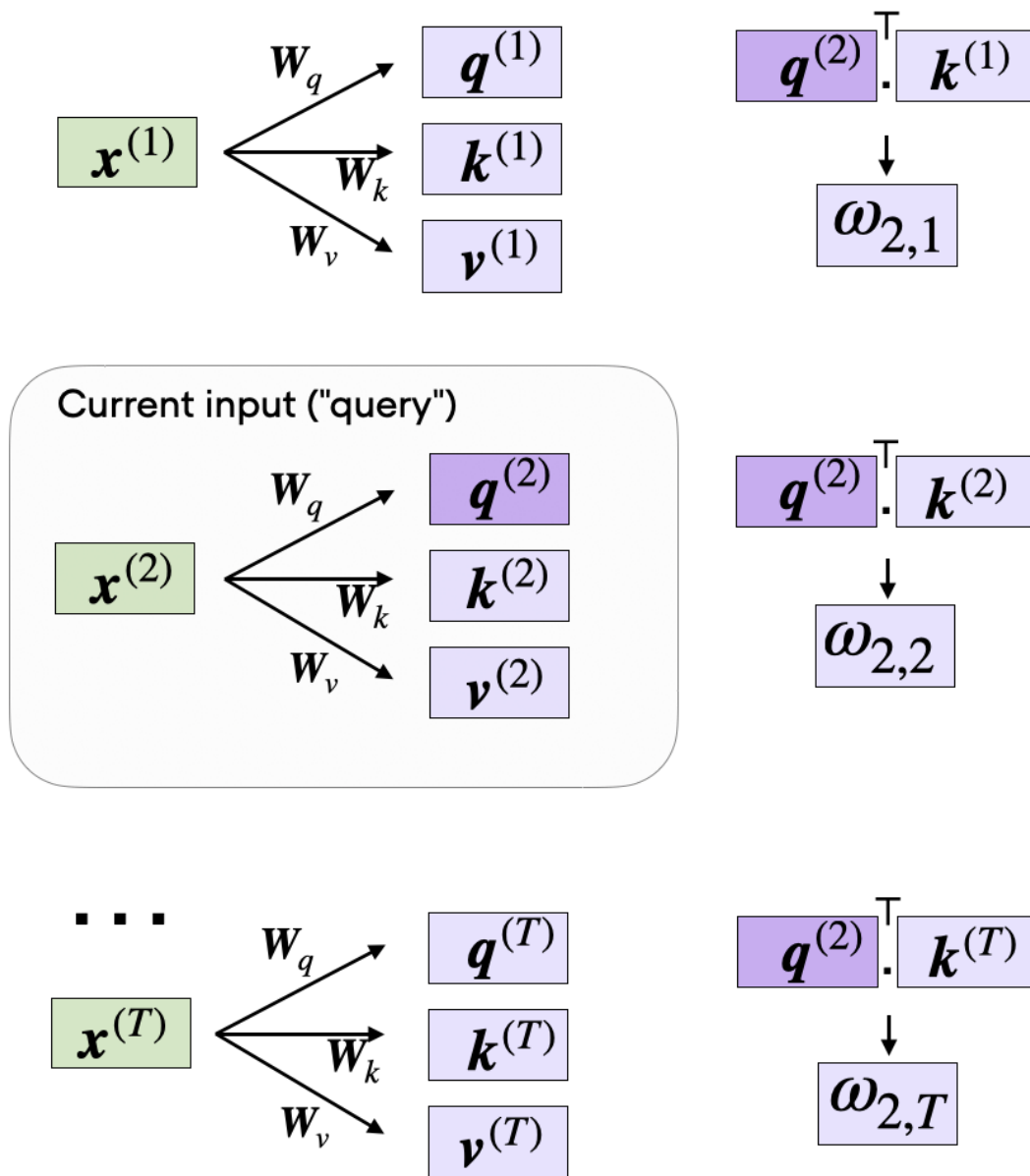
Self-attention utilizes three weight matrices, query weight matrix  $\mathbf{W}^q \in \mathbb{R}^{d \times d_k}$ , key weight matrix  $\mathbf{W}^k \in \mathbb{R}^{d \times d_k}$ , and value weight matrix  $\mathbf{W}^v \in \mathbb{R}^{d \times d_v}$  (usually  $d_k = d$ , the model size). These matrices are adjusted as model parameters during training, and used to project the inputs  $\mathbf{X} \in \mathbb{R}^{L \times d}$  into query, key, value components of the sequence, respectively.

- query matrix  $\mathbf{Q} \in \mathbb{R}^{L \times d_k} = \mathbf{X}\mathbf{W}^q$  with query vector (column vector)  $\mathbf{q}_i \in \mathbb{R}^L$
- key matrix  $\mathbf{K} \in \mathbb{R}^{L \times d_k} = \mathbf{X}\mathbf{W}^k$
- value matrix  $\mathbf{V} \in \mathbb{R}^{L \times d_v} = \mathbf{X}\mathbf{W}^v$

where:

- $L$  = The segment length of input sequence
- $d_k = d$  = The model size, the embedding vector size
- $d_v$  = The size of the value vector, which might be arbitrary
- $d_k$  = Vector dimension of vectors
- $i$  = index of the element/token in the input sequence

[DIFF] Calculate unnormalized alignment weights with dot product

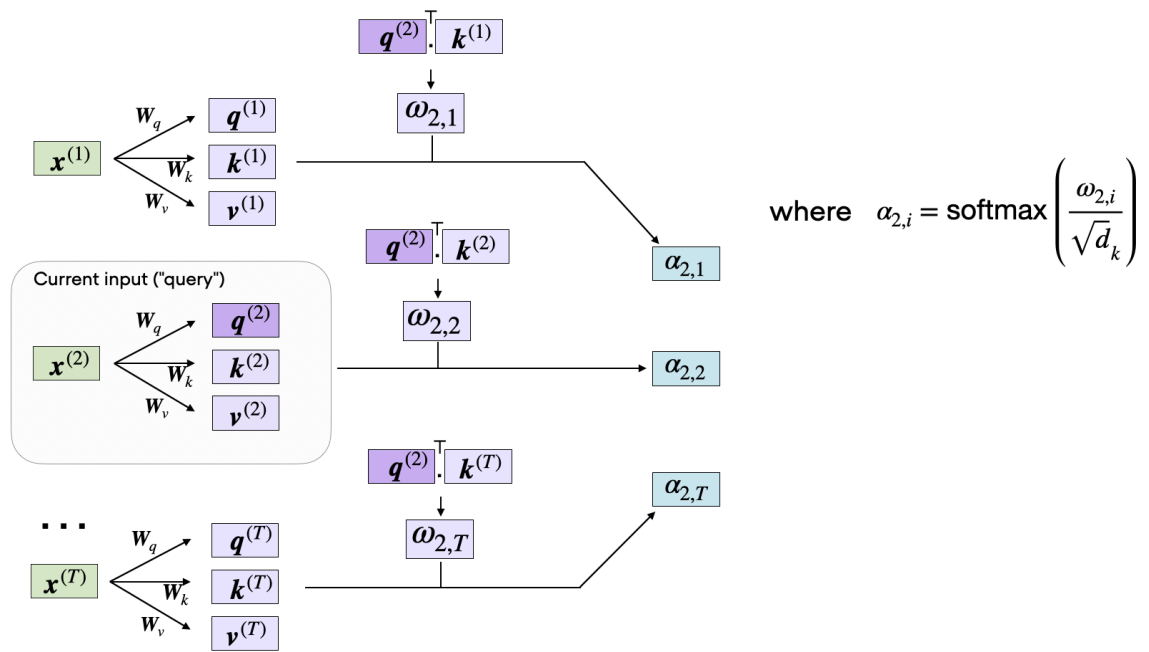


Once we obtain the vectors  $\mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i$  are used to compute *alignment weights*, which determine how much a particular element (position  $i$ ) is similar to another element (position  $j$ ) in the sequence, by computing the dot product between query and key sequences

$$w_{ij} = \mathbf{q}_i \mathbf{k}_j^\top$$

Note that the dot product is a similarity measure between two vectors. The higher the dot product, the more similar the two vectors are. Alignment weights here are raw similarity scores, unnormalized to a probability distribution.

Get alignment scores with scaled dot product and softmax



The embedding for an element is usually high-dimensional. To prevent the attention weights from becoming too small or too large, which could lead to numerical instability or affect the model's ability to converge during training, we scale the *alignment weights* down by  $\frac{1}{\sqrt{d_k}}$ .

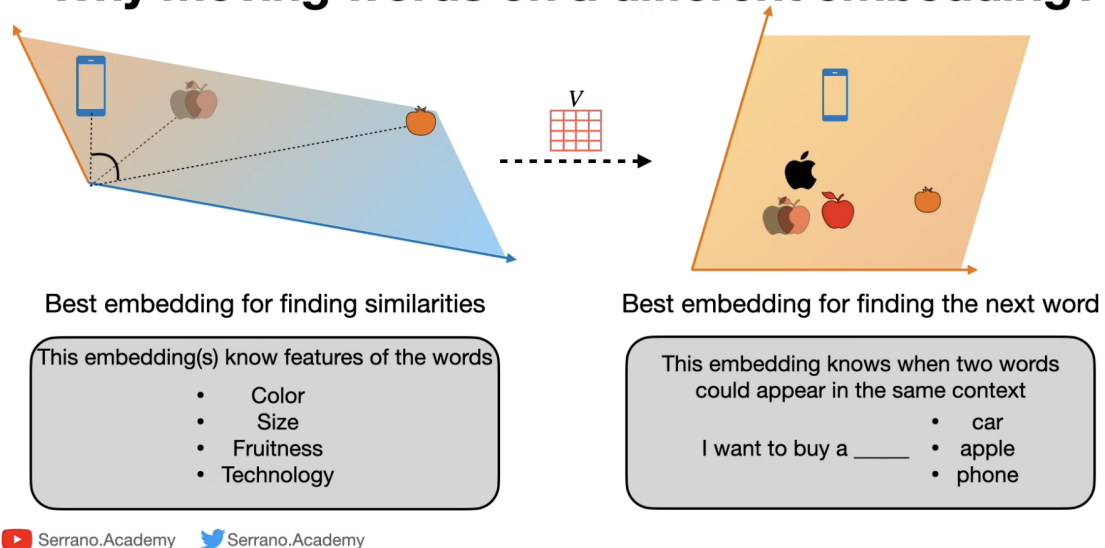
We also pass them through a softmax function, so that the resulting *alignment scores*  $a_{ij}$  follow a probability distribution.

$$a_{ij} = \text{softmax} \left( \frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{d_k}} \right)$$

Again, this score  $a_{ij}$  quantifies a particular element (position  $i$ ) is similar to another element (position  $j$ ) in the sequence.

[DIFF] Compute attention scores by multiplying with value vector

# Why moving words on a different embedding?



Until now, the calculated alignment scores revolve around how the current element  $i$  is similar to an arbitrary element  $j$ . To account for ALL other input elements  $j$  relationship to the current element  $i$ , we calculate the weighted sum of alignment scores  $a_{ij}$  and multiplying with the value vector  $\mathbf{v}_j$ , to obtain the *context vector*  $\mathbf{z}_i$  around the current element  $i$

$$\mathbf{z}_i = \sum_{j=1}^d a_{ij} \mathbf{v}_j$$

This context vector represents aggregated information from all other elements in the sequence, determined by how much the current element  $i$  attend to them. This vector effectively captures the relevant information from the entire sequence needed to generate the output at that particular position  $i$ , taking into account both the importance and the content of each part of the input.

## Single head attention

To generalize, the formula for self-attention mechanism is

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left( \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} \right) \cdot \mathbf{V}$$

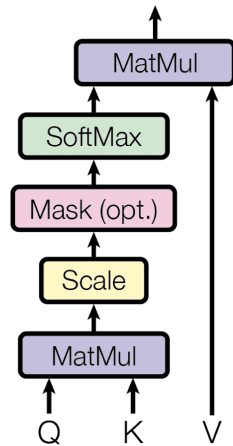
1. Create queries  $\mathbf{Q}$ , keys  $\mathbf{K}$ , and values  $\mathbf{V}$
2.  $\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}$ : Compute scaled dot product of matrices  $\mathbf{Q}$  and  $\mathbf{K}$  to get the alignment weights.  
This encodes the similarity between the input and the target output
3.  $\text{softmax}()$ : Pass the alignment scores computed previously to a softmax function, so that they follow a probability distribution: every score is between 0 and 1, a valid probability.



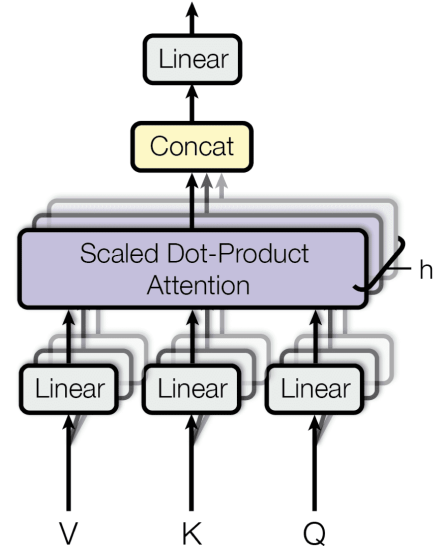
4.  $\cdot V$ : Matrix-multiply by the matrix  $V$  (values) to calculate attention scores with context. Higher attention score assigned to a certain word means that it will have stronger influence on the next word in the decoder's output.

## [DIFF] Multi-head attention

### Scaled Dot-Product Attention



### Multi-Head Attention



Multi-head attention involves  $h$  heads of self-attention mechanism described above, each consisting of query, key, and value matrices. The output of each head is concatenated and linearly transformed to obtain the final output.

Each head focuses on different parts of the sentence to understand its full meaning. This allows the model to concurrently attend to information from different positions of the input data.

## [NEW] Cross-attention

In self-attention, each token in a sequence attends to all other tokens in the same sequence to compute a representation. In contrast, cross-attention extends this concept across two different sequences.

Here, one sequence (the query) attends to another (the context or key-value pairs), enabling the model to incorporate information from an external source into its representation. This is especially useful in this RAG pipeline models where the query might be a user's input, and the context is relevant information retrieved from a database or document.

Let  $Q$  be the query matrix derived from one data stream (e.g., a user's query in a retrieval system), and  $K$  and  $V$  be the key and value matrices derived from a different data stream (e.g., retrieved documents). Cross-attention is computed similarly as single-head attention above:

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

In the context of a retrieval-augmented generation pipeline, cross-attention allows the model to focus on specific parts of the retrieved information (the context) that are most relevant to the query. This mechanism enables the model to integrate and synthesize information from external sources, enhancing its ability to generate more informed and contextually relevant responses.

## Generator Advanced 1: From Transformer to GPT-3.5

GPT-3.5 builds upon the Transformer model described above, by incorporating several improvements and innovations that lead to better performance and more accurate text generation, including:

1. **Scale:** GPT-3.5 has 175 billion parameters, which is over 10 times the size of its predecessor, GPT-2, and more than 45 times the size of the original Transformer model. This increased size allows GPT-3.5 to capture more complex patterns and representations in the text data, enabling it to generate more accurate and contextually relevant responses.
2. **Training Data:** GPT-3.5 has been trained on a massive amount of text data, including almost all of the public web, amounting to around 45 TB of text data. This extensive training enables GPT-3.5 to learn more advanced patterns and generate more human-like text.
3. **Attention Mechanism:** While the original Transformer model uses self-attention, GPT-3.5 incorporates a more advanced attention mechanism that allows the model to selectively focus on segments of input text it predicts to be the most relevant. This improved attention mechanism contributes to the better performance of GPT-3.5 compared to the original Transformer model.
4. **Decoder-only Architecture:** GPT-3.5, like its predecessors, uses a decoder-only architecture. This means that it generates text based on the input it receives, without using an encoder to process the entire input sequence. This approach simplifies the model and allows for better control over the generated text.

## RAG pipeline: Hybrid retrieval and reranking

To summarize, the step-by-step process of RAG is as follows:

1. Prepare the source data by converting it into a vector database of dense embeddings. These dense embeddings contain the semantic information of the sentences and context around them.

2. Connect the vector database to the retrieval model, which uses the embeddings to find the most relevant documents to the user query.
3. Connect the retrieval model to the reranking model, which reorders the retrieved documents based on their relevance to the user query.
4. Connect the reranking model to the LLM, which generates the response based on the user query and the retrieved documents.
5. Send the query through several steps of the pipeline, from embedding to retrieval-reranking to response generation, to produce the final response.
6. Repeat the process (step 2-5) for each user query and evaluate the generated responses. Evaluations including semantic similarity, ROUGE, and LLM evaluations.

## LLM: GPT-3.5 through OpenAI API

```
In [3]: import openai
from llama_index.llms import OpenAI
from llama_index.llms.llama_utils import completion_to_prompt, messages_to_prompt

from src.utils.get_keys import get_openai_api_key

openai.api_key = get_openai_api_key()

llm = OpenAI(model="gpt-3.5-turbo-1106", temperature=0.1, max_tokens=500)
```

## Sentence embedding: Turn sentences into dense, high-dimensional vectors of numbers

```
In [4]: from llama_index.embeddings import HuggingFaceEmbedding

# dimension 512
embedding = HuggingFaceEmbedding(model_name="sentence-transformers/all-MiniLM-L6-v2")

/Users/chiffonng/Downloads/school/CS156/venv/lib/python3.11/site-packages/tqdm/auto.py:21: TqdmWarning: IPProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

## Sentence Window Retrieval: Get context window around the sentences, parse to nodes and index to a vector database

```
In [5]: from llama_index.node_parser import SentenceWindowNodeParser
from llama_index import (
    VectorStoreIndex,
    StorageContext,
    ServiceContext,
    load_index_from_storage,
```

```

)

INDEX_DIR = Path("./data/final/sentence_index")

def build_sentence_window_index(
    documents: List[Document],
    window_size: int = 3,
    llm=llm,
    embed_model: HuggingFaceEmbedding = embedding,
    index_dir: Path | str = INDEX_DIR,
) -> VectorStoreIndex:
    """
    Get the sentences and context around them from the list of documents and

    Args:
        documents (List[Document]):
            list of documents to extract sentences from
        window_size (int, default = 3):
            number of sentences to include around the target sentence,
            defaults to 3, meaning 1 sentence before and 1 sentence after.
        llm:
            language model to use for sentence generation
        embed_model (HuggingFaceEmbedding, default = sentence-transformers/a
            embedding model to use for sentence similarity
        index_dir (Path | str):
            directory to save the index to
    """

    # create the sentence window node parser w/ default settings
    sent_context_parser = SentenceWindowNodeParser.from_defaults(
        window_size=window_size,
        window_metadata_key="window",
        original_text_metadata_key="original_text",
    )

    sentence_context = ServiceContext.from_defaults(
        llm=llm,
        embed_model=embed_model,
        node_parser=sent_context_parser,
    )

    # if the index doesn't exist, build it
    if not index_dir.exists():
        sentence_index = VectorStoreIndex.from_documents(
            documents, service_context=sentence_context
        )

        sentence_index.storage_context.persist(persist_dir=index_dir)
    # otherwise, load it from storage
    else:
        sentence_index = load_index_from_storage(
            StorageContext.from_defaults(persist_dir=index_dir),
            service_context=sentence_context,
        )
    return sentence_index

```

```
sentence_index = build_sentence_window_index(
    md_docs, window_size=3, index_dir=INDEX_DIR
)
```

## Retrieval & Reranking: Find $k$ most relevant text chunks to the question, then rerank and narrow down to $n$ results

In this code cell, I do several steps:

1. Set up the retriever and reranker to find seven relevant chunks, and rerank them to three most relevant chunks to the question.
2. Set up the LLM to generate the response based on the question, the prompt, and the context of three relevant chunks. I allow the LLM to use its own knowledge to generate the response *only when* my knowledge base does not have the answer. However, it has to flag the response as "generated" so that I can review it later.

```
In [6]: import nest_asyncio

from llama_index import ServiceContext, VectorStoreIndex, StorageContext
from llama_index.indices.postprocessor import MetadataReplacementPostProcessor
from llama_index.indices.postprocessor import SentenceTransformerRerank
from llama_index.prompts import PromptTemplate

nest_asyncio.apply()

text_qa_template = """You are an AI assistant that answers questions in a factious way.
Context information is {context_str}\n
You try to answer question {query_str} using the context information given.\n
If the context isn't helpful, you first state that "The answer is not available" and then you generate an answer.

def get_sentence_window_query_engine(
    sentence_index,
    similarity_top_k=7,
    rerank_top_n=3,
    text_qa_template=text_qa_template,
):
    # define postprocessors
    postproc = MetadataReplacementPostProcessor(target_metadata_key="window")
    rerank = SentenceTransformerRerank(
        top_n=rerank_top_n, model="cross-encoder/ms-marco-MiniLM-L-12-v2"
    )

    sentence_window_engine = sentence_index.as_query_engine(
        response_mode="compact",
        text_qa_template=PromptTemplate(text_qa_template),
        similarity_top_k=similarity_top_k,
        node_postprocessors=[postproc, rerank],
```

```

    )
    return sentence_window_engine

query_engine = get_sentence_window_query_engine(sentence_index=sentence_index)

```

## [DIFF] Question & Answering

I prepared a list of questions, divided into four different tasks, to test the system:

1. extract: questions include keywords and can be easily answered by retrieving one document (it can correspond to one or multiple chunks)
2. summarize: questions include keywords and require the system to summarize across one or multiple documents.
3. infer: questions requiring comprehensive understanding of multiple documents (at least 10, which are larger than the number of retrieved contexts). I expect the system to perform less well on these questions, because they requires it to infer from their understanding.
4. use external knowledge: questions go beyond the knowledge base, requiring the system to flag and answer the question using its own knowledge.

We input the questions into LLM (along with the relevant external information) and evaluate the responses later.

```

In [59]: import json
import pandas as pd

pd.set_option("max_colwidth", 1000)

def display_result_table(results: pd.DataFrame | dict) -> pd.DataFrame:
    """
    Left align the text in a dataframe.

    Args:
        df (pd.DataFrame):
            dataframe to left align

    Returns:
        pd.DataFrame:
            left aligned dataframe
    """
    if isinstance(results, dict):
        results = pd.DataFrame.from_dict(results)

    left_aligned_df = results.style.set_properties(**{"text-align": "left"})
    left_aligned_df = left_aligned_df.set_table_styles(
        [dict(selector="th", props=[("text-align", "left")])]
    )
    return left_aligned_df

```

```

with open("../src/evaluation/qa_pairs.json", "r") as f:
    honest_qa_pairs = json.load(f)["honest"]

# get the questions and answers
honest_questions = [pair["query"] for pair in honest_qa_pairs]
honest_ground_truths = [pair["response"] for pair in honest_qa_pairs]

eval_results = {} # reset
eval_results = {
    "task": [pair["task"] for pair in honest_qa_pairs],
    "query": honest_questions,
    "ground_truth": honest_ground_truths,
}

# display task and query in a table or dataframe, but not the ground truth
display_result_table(pd.DataFrame.from_dict(eval_results).drop(columns=["gro

```

Out[59]:

	task	query
0	extract	What is a probability?
1	extract	In marketing 4P, what factors affect price?
2	extract	What does microenomics concern?
3	extract	Explain gradient descent in simple words.
4	extract	How do I interpret regression coefficients?
5	summarize	Provide an overview of most important mechanisms for transformers.
6	summarize	How should I evaluate a synthetic control case study?
7	summarize	How to identify which critical point is local minimum or maximum?
8	summarize	What are some methods to measure how well a Bayesian model estimates the posterior?
9	infer	Based on the NLP techniques I already learned, what should I learn next?
10	infer	From my notes, what did I learn in the business classes?
11	external	Why do we need to change the basis?
12	external	What does alignment research entail?
13	external	How do we use linear parameter model to build non-linear models?

In [8]:

```

import time
import random

from llama_index import Response
from openai import RateLimitError
from tenacity import (
    retry,
    stop_after_attempt,
    wait_random_exponential,
    retry_if_exception_type,
    after_log,
)

```

```

# exponential backoff to handle rate limits
@retry(
    stop=stop_after_attempt(3),
    wait=wait_random_exponential(multiplier=10, min=60, max=200),
    retry=retry_if_exception_type(RateLimitError),
    # after=after_log(logger, logging.DEBUG),
)
def ask(question: str) -> Response | None:
    """
    Ask a question to the model
    """
    response = query_engine.query(question)
    return response

model_responses = []

# Send the questions to the LLM
for question in honest_questions:
    model_answer = ask(question)
    model_responses.append(model_answer)
    time.sleep(10 + 10 * random.random())

```

```

In [60]: # check that all questions are answered
assert sum([response is None for response in model_responses]) == 0

eval_results["response"] = [response.response for response in model_responses]
eval_results.keys()

```

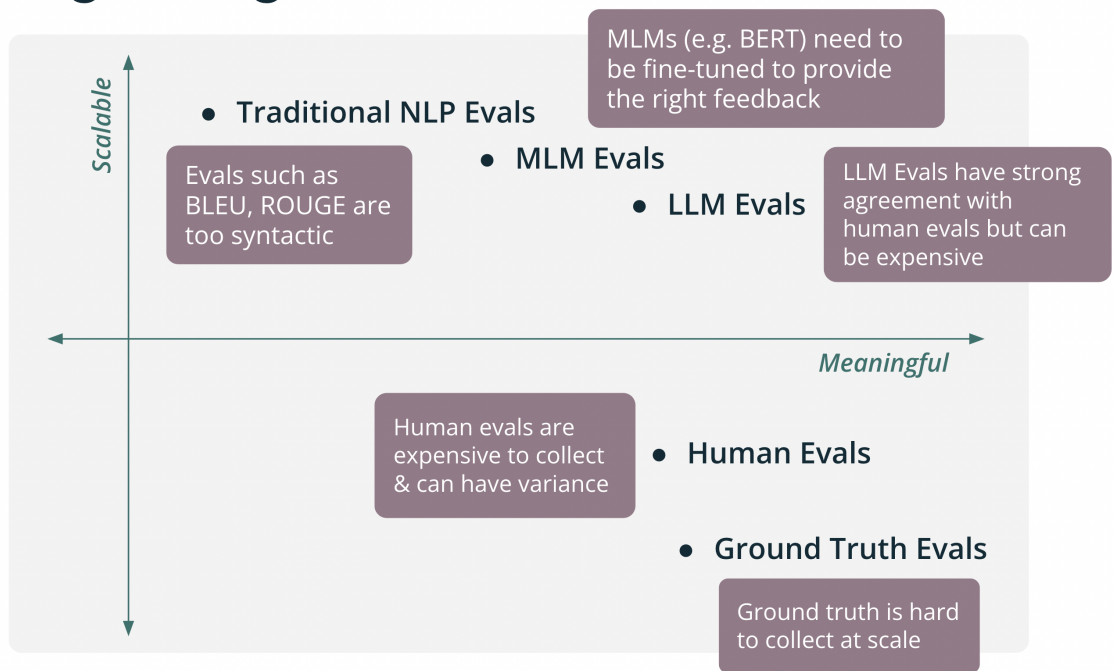
```

Out[60]: dict_keys(['task', 'query', 'ground_truth', 'response'])

```

## [NEW] Evaluation





To evaluate the performance of RAG, there is a spectrum of evaluations, differing in terms of meaningful (how much information can be used to improve the model) and scalable (how much effort is required to collect the data) evaluations.

In this section, I explore several metrics across this spectrum, including

- Ground truth evals: compare the model answer with the expert answer (it's me!)
- LLM evals: using LLM to evaluate the model answer, in terms of its relationship between the query, the retrieved information, and the human answer
- Embedding evals (Semantic similarity): using cosine similarity to compare the embedding of the model answer with the embedding of human answer

## [NEW] Ground Truth Evals: Compare the model answer with the human answer

Use an external LLM to evaluate the similarity between the model answer and the human answer. It's referred as "Correctness" in the LlamaIndex framework.

```
In [13]: from llama_index.evaluation import CorrectnessEvaluator

# Using the same LLM
eval_context = ServiceContext.from_defaults(llm=llm)

correctness_evaluator = CorrectnessEvaluator(service_context=eval_context)
```

## [NEW] Embedding Evals: Cosine similarity to measure the semantic similarity between the model answer and human answer

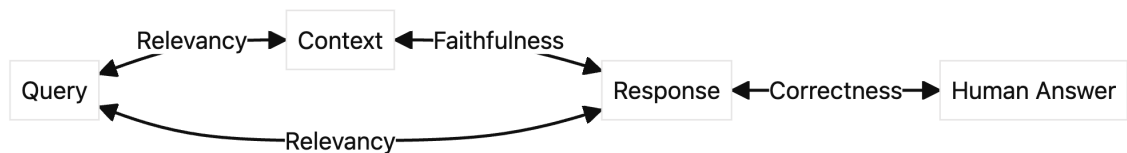
```
In [61]: from llama_index.evaluation import SemanticSimilarityEvaluator

cos_sims = []
cosine_similarity_evaluator = SemanticSimilarityEvaluator(
    service_context=ServiceContext.from_defaults(embed_model=embedding),
    similarity_threshold=0.6,
)

for model_answer, human_answer in zip(model_responses, honest_ground_truths):
    result = await cosine_similarity_evaluator.aevaluate(
        response=model_answer, reference=human_answer
    )
    cos_sims.append(result.score)

eval_results["semantic_similarity"] = cos_sims
```

## [NEW] LLM Evals



At the most basic level, RAG pipeline should retrieve and reliably use the information needed to answer questions it is intended for. To evaluate honesty, I will use the following metrics:

- **Faithfulness (Hallucination):** Does the model ground its response in the context (Yes=1, No=0)? In other words, it checks if the model's answer is consistent with the information present in the context it was given.
- **Relevancy (Answer & Context Relevancy):** Are the retrieved context and the generated answer relevant to the query (Yes=1, No=0)? This is important to ensure that the model's responses are not just correct and faithful, but also relevant to the user's query.
- **Correctness:** From 0 to 1, how well does the generated answer match the reference (e.g. ground-truth) answer? The correctness evaluator compares the generated answer to the reference answer and outputs a score between 1 and 5, where 1 is the worst and 5 is the best, but I scale down the score to 0-1 to match other metrics.

```
In [25]: from llama_index.evaluation import (
    BatchEvalRunner, # batch metrics, but can't control rate limit
    RelevancyEvaluator,
    FaithfulnessEvaluator,
    EvaluationResult, # attributes: passing, score, feedback
)

relevancy_evaluator = RelevancyEvaluator(service_context=eval_context)
```

```

faithfulness_evaluator = FaithfulnessEvaluator(service_context=eval_context)

evaluators_runner = BatchEvalRunner(
    {"faithfulness": faithfulness_evaluator,
     "relevancy": relevancy_evaluator,
     "correctness": correctness_evaluator,},
    workers = 3,
)

honest_eval_results = await evaluators_runner.aevaluate_responses(
    queries=honest_questions,
    responses=model_responses,
    reference=honest_ground_truths,
)

```

## [NEW] Performance

### [NEW] Performance by Question

```

In [68]: def scale_score_to_01(score):
          return (score - 1) / (5 - 1)

for metric in honest_eval_results.keys():
    if metric in ["relevancy", "faithfulness"]:
        eval_results[metric] = [int(result.passing) for result in honest_eval_results[metric]]
    else:
        eval_results[metric] = [scale_score_to_01(result.score) for result in honest_eval_results[metric]]

display_result_table(eval_results)

```

Out[68]:	task	query	ground_truth	response	semantic_similarity	faithfulness	r
				<p>The answer is not available in your knowledge base. I will use my own knowledge to answer the question.</p> <p>Probability is a measure of the likelihood of an event occurring. It is represented by a number between 0 and 1, where 0 indicates impossibility of the event and 1 indicates certainty. In a naive definition, it assumes that all outcomes are equally likely and there are a finite number of outcomes. In a non-naive definition, it is based on a sample space and an event, and follows the Sum Rule. Conditional probability, on the other hand, is the probability of an event occurring given that another event has occurred. It is calculated by asking, "What is the probability of observing an outcome from event A given that we observed an outcome from event B?" This concept is important for understanding the likelihood of events in relation to each other.</p>			
0	extract	What is a probability?	<p>A probability presents how likely an event is to occur with a number between 0 and 1, where, roughly speaking, 0 indicates impossibility of the event and 1 indicates certainty. The higher the probability of an event, the more likely it is that the event will occur.</p>		0.847701	1	1

task	query	ground_truth	response	semantic_similarity	faithfulness	r
1	extract	In marketing 4P, what factors affect price?	<p>Several factors affect price: supply costs, discounts, competitors' prices. However, the optimal approach should be customer-based. We want to quantify the range of willingness to pay and change prices by segments. If we just do cost-based pricing, we miss out on customer needs.</p> <p>In marketing 4P, several factors affect price, including supply costs, discounts, and competitors' prices. However, the optimal approach should be customer-based. It's important to quantify the range of willingness to pay and change prices by segments. If we just do cost-based pricing, we may miss out on customer needs. Other pricing strategies to consider include outcome-based pricing, value-based pricing, bundling, upselling, and cross-selling.</p>	0.918157	1	1

task	query	ground_truth	response	semantic_similarity	faithfulness	r
2	extract	What does microeconomics concern?	Microeconomics is the study of how individuals, households, and firms make decisions about using limited resources—not just buying, selling, or producing them	0.967222	1	1
3	extract	Explain gradient descent in simple words.	Gradient descent is an iterative, greedy algorithm that finds a LOCAL minimum or maximum of a differentiable function (e.g. cost function) by iteratively adjusting in the direction of steepest descent. Intuitively, gradient descent is like finding the lowest point in a hilly landscape. Imagine you're on a hill, and you want to reach the lowest point (minimum). You start at any random spot on the hill, look around, feel the slope, and walk small steps in the steepest	0.929030	1	1

task	query	ground truth direction. You repeat this	response	semantic_similarity	faithfulness	reward
		process until you reach the lowest point, where the slope is nearly flat. You've found the minimum!	with respect to the parameters. The gradient points in the direction of the steepest increase, so the algorithm moves in the opposite direction to decrease the cost. By taking small steps in the direction of the negative gradient, the parameters are updated until the algorithm converges to the minimum cost. It's important to choose an appropriate learning rate, as a too small rate can make the algorithm slow, while a too large rate can cause it to overshoot the minimum. Experimenting with different learning rates and plotting the cost function over iterations can help in understanding how gradient descent works and finding the optimal learning rate.			
4	extract	How do I interpret regression coefficients?	The interpretation of regression coefficients involves understanding the relationship between the independent and dependent variables in a regression model. The slope of the coefficient represents the change in the	0.841730	1	1

task	query	ground truth	response	semantic_similarity	faithfulness	r
		<p>the difference between the predicted and actual value (aka the uncertainty of the regression model. Interaction term indicate whether there is a joint effect of some predictors on the variable.</p>	<p>dependent variable for a one-unit increase in the independent variable, while the intercept represents the value of the dependent variable when all independent variables are set to 0. The error term, or residual, is the difference between the observed value and the value predicted by the regression model. The prediction interval is a range of values within which we can predict the dependent variable with a certain level of confidence. In hypothesis testing for coefficients, we test whether a coefficient is statistically significant, which indicates that a difference in the independent variable is truly correlated with a difference in the dependent variable. The p-value is used to assess the probability of observing a substantial association between the predictor and the response by chance, assuming the null hypothesis is true. Overall, interpreting regression coefficients involves understanding</p>			



task	query	ground_truth	response	semantic_similarity	faithfulness	r
			the impact of the independent variables on the dependent variable and determining the statistical significance of these relationships.			
5	summarize	Provide an overview of most important mechanisms for transformers.	<p>The most important mechanisms for transformers include self-attention mechanisms, multi-head attention, positional encoding, and feedforward neural networks. These mechanisms enable transformers to handle sequence data effectively, by learning the relationships between different elements of the sequence and the relative positions of these elements, and by paying attention to relevant parts of the sequence when making predictions.</p> <p>The most important mechanisms for transformers include self-attention, positional encoding, and the encoder-decoder architecture. Self-attention allows the model to focus on different parts of the input sequence and capture long-range dependencies more effectively. Positional encoding is used to preserve the order of the input sequence. The encoder-decoder architecture is utilized, where the encoder processes the input sequence and the decoder generates the output sequence. These mechanisms make the Transformer model faster, more efficient, and effective in capturing long-range dependencies in natural language processing tasks.</p>	0.907098	1	1
6	summarize	How should I evaluate a synthetic	We can conduct placebo studies to ensure that	0.920340	1	1
			To evaluate a synthetic control case study, you			

task	query	case study?	ground truth	response	semantic_similarity	faithfulness	r
			the treatment effect is only observed by the treated unit in the specific time. This can be done through in-time placebo (reapplying the treatment to a different time period before the intervention) or in-space placebo (reapplying the treatment to a comparison unit). Further, we can perform robustness tests by iteratively excluding each comparison unit with positive weights (leave-one-out), reconstructing the synthetic control, and re-estimating the treatment effect to see if the conclusion still holds.	should first assess the validity of the synthetic control by checking the similarity of the treated unit's pre-treatment characteristics to the synthetic control. This involves comparing the relevant characteristics of the treated unit before it underwent the treatment with the weighted combination of comparison units that make up the synthetic control. Next, you should compare the actual post-treatment outcome of the treated unit with the counterfactual outcome predicted by the synthetic control. This comparison will help determine the effectiveness of the synthetic control in estimating the treatment effect. Additionally, you can refine the synthetic control by adjusting the set of covariates, exploring alternative weighting schemes, or incorporating additional control units. This step can help improve the accuracy of the synthetic control and enhance the validity of the study. In the			

task	query	ground_truth	response	semantic_similarity	faithfulness	r
			<p>context of placebo studies, it is important to ensure that the individual treatment effect is only observed by the treated unit in the specific time. This can be done through in-time placebo studies, where the treatment is reapplied to a different time period before the intervention. This helps to assess the true impact of the treatment on the treated unit. Overall, evaluating a synthetic control case study involves assessing the validity of the synthetic control, comparing post-treatment outcomes, and refining the control as needed to ensure accurate and reliable results.</p>			

task	query	ground_truth	response	semantic_similarity	faithfulness	r
7	summarize	How to identify which critical point is local minimum or maximum?	<p>Based on the context information provided, to identify whether a critical point is a local minimum or maximum, you can follow these steps: 1. Set all first derivatives to 0 to find critical points. 2. Test whether the critical points are maximum or minimum using the second derivative test. 3. Compare the function value at the local extremum with the function value at the endpoints. If the second derivative is positive at a critical point, then it is a local minimum. If the second derivative is negative at a critical point, then it is a local maximum. If the second derivative test is inconclusive, then the point is neither a local minimum nor a local maximum. This process helps in determining whether a critical point is a local minimum or maximum.</p> <p>We can use the second derivative test. If the second derivative is positive, the critical point is a local minimum. If the second derivative is negative, the critical point is a local maximum. If the second derivative is zero, the test is inconclusive.</p>	0.940739	1	1

task	query	ground_truth	response	semantic_similarity	faithfulness	r
8	summarize	What are some methods to measure how well a Bayesian model estimates the posterior?	<p>Some methods to measure how well a Bayesian model estimates the posterior include computing the maximum likelihood estimation, sampling using Markov chain Monte Carlo methods, and fitting using variational inference. Additionally, one can try using a distribution with fewer unknown parameters to reduce the computational cost and express uncertainty in the model. Another method is to build a prior-predictive distribution before seeing the current data and then compute the posterior-predictive distribution to generate samples from it. If there is enough data, the posterior distribution can also be approximated by a normal distribution, and the "discrepancy" between the model solution of the posterior and the normal approximation can be quantified using KL divergence and minimized.</p> <p>1. Model evidence: Quantifies the probability of observed data under the model, indicating how well a Bayesian model fits the observed data.</p> <p>2. Kullback-Leibler divergence: Quantifies the difference between two probability distributions, typically a true/observed distribution and an estimated model distribution.</p> <p>3. Log Point-wise Predictive Density (LPPD): Represents the difference between the KL divergence of the target and model and the KL divergence of the target and itself. It is a measure of how well the model fits the data.</p>	0.783855	1	1

task	query	ground_truth	response	semantic_similarity	faithfulness	r
9	infer	<p>Delve deeper into advanced transformer models beyond BERT. Learn how to fine-tune pre-trained models like BERT or GPT for specific downstream tasks. Explore models that handle multiple modalities, such as images and text. Models like CLIP (Contrastive Language-Image Pre-training) and ViT (Vision Transformer) can provide insights into cross-modal representations. Gain an understanding of ethical considerations and biases in NLP models. Explore methods to detect and mitigate biases in language models.</p>	<p>Based on the NLP techniques you have already learned, you may want to consider learning about the attention mechanism in sequence-to-sequence models. This mechanism allows the model to focus on learning enough previous words to predict the next word, which can be particularly useful for tasks like machine translation and text generation. Additionally, you may also want to explore named entity recognition (NER) techniques, which can help identify important named entities in the text, such as people, places, and dates. These techniques can be used alone or alongside topic identification after text preprocessing.</p>	0.795548	1	1

task	query	ground_truth	response	semantic_similarity	faithfulness	r
10	infer	From my notes, what did I learn in the business classes?	<p>In marketing, you learned about customer-centric thinking, such as customer segmentation and customer lifetime value, and popular frameworks, such as five competitive forces and marketing 4Ps. In accounting, you learned about three financial statements that are used to evaluate a company's financial performance: balance sheet, income statement, and cash flow statement. In finance, you learned about the time value of money, valuation, options, and risk management. You also learned the basics of economics.</p> <p>The answer is not available in your knowledge base. I will use my own knowledge to answer the question. From your notes in the business classes, you learned about the balance sheet, which includes assets, liabilities, and stockholders' equity. You also learned about the accounting cycle, which involves analyzing transactions, creating journal entries, and posting to T-accounts. Additionally, you learned about debit and credit bookkeeping and common accounts. Finally, you learned about specific accounts that are increased with a credit, such as revenue and additional paid-in-capital.</p>	0.860001	1	1

task	query	ground_truth	response	semantic_similarity	faithfulness	r
11	external	Why do we need to change the basis?	<p>The answer is not available in your knowledge base. I will use my own knowledge to answer the question. We need to change the basis in linear algebra for a few reasons. One reason is that working with a different basis can make certain calculations or transformations easier to perform. Additionally, changing the basis can provide a different perspective on the vectors or linear transformations, which can be useful for understanding their properties and behavior. Finally, in some cases, a specific basis may be more natural or intuitive for a given problem, so changing the basis allows us to work with vectors and linear transformations in a more convenient way.</p> <p>The answer is not available in your knowledge base. I will use my own knowledge to answer the question. The change of basis is necessary when we want to work with a different linear transformation or when we want to represent vectors in a different basis. It allows us to convert matrices and vectors from one basis to another.</p>	0.945550	1	1



task	query	ground_truth	response	semantic_similarity	faithfulness	r
12	external	What does alignment research entail?	<p>The answer is not available in your knowledge base. I will use my own knowledge to answer the question. Alignment research entails the process of gathering information directly from customers and prospective customers in the target market to gain conclusive results. This type of research may involve higher costs and take longer to conduct than secondary research, but it provides specific and accurate data. It can guide primary research by providing insights, but there may be bias when interpreting the source's data for our needs.</p>	0.957372	1	1
13	external	How do we use linear parameter model to build non-linear models?	<p>The answer is not available in your knowledge base. I will use my own knowledge to answer the question. We can use linear parameter model to build non-linear models by adding non-linear terms to the model. For example, we can add a quadratic term to the model to capture the non-linear relationship between the</p>	0.927523	0	1

task	query	ground truth dependent variable and the independent variable.	response	semantic_similarity	faithfulness	rank
			can include higher-order terms of the predictor variables (e.g., $x^2$ , $x^3$ ) to capture non-linear relationships between the predictors and the response variable. Spline regression involves dividing the range of the predictor variable into segments and fitting separate linear models within each segment, allowing for non-linear relationships within each segment. Another approach is to transform the predictor variables using functions such as logarithmic, exponential, or power transformations to capture non-linear patterns in the data. These techniques allow you to use a linear parameter model to capture non-linear relationships between the predictors and the response variable.			

Log the model inferences in an external database to save time and avoid re-running the model.

```
In [80]: eval_results_df = pd.DataFrame.from_dict(eval_results)
eval_results_df.to_csv("src/evaluation/eval_results.csv")
```

## [NEW] Overall Performance

```
In [97]: metrics = eval_results_df.select_dtypes(include=['number']).columns.tolist()
metrics
```

```
Out[97]: ['semantic_similarity', 'faithfulness', 'relevancy', 'correctness']
```

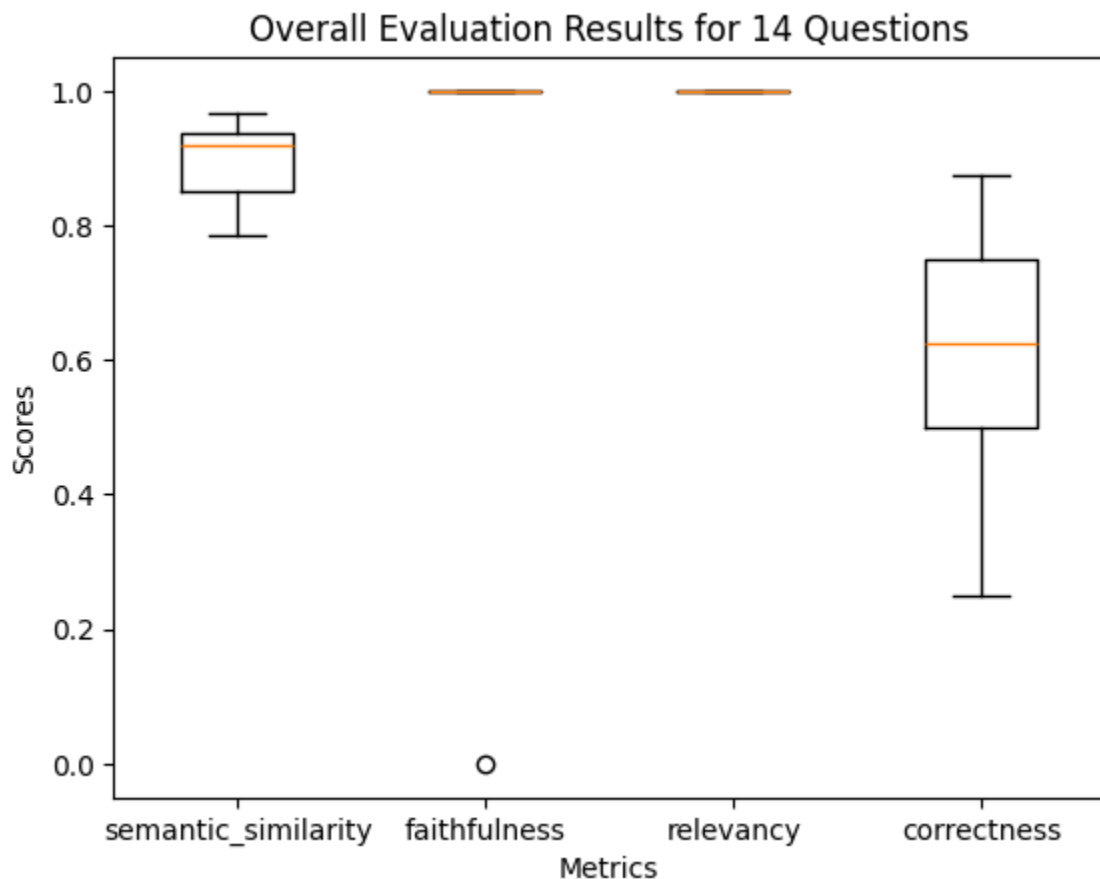
```
In [100]: import matplotlib.pyplot as plt
import numpy as np

plt.boxplot(eval_results_df[metrics], labels=metrics)

# Add labels for x-axis and y-axis
plt.xlabel("Metrics")
plt.ylabel("Scores")

# Set the title
plt.title(f"Overall Evaluation Results for {len(honest_questions)} Questions")

# Show the plot
plt.show()
```



This RAG pipeline is performing quite well in certain aspects. The average semantic similarity score is high at approximately 0.896, indicating that the model's answers are semantically aligned with the reference answers most of the time. Faithfulness, which assesses if the model's answers are supported by the given context, also scores high, with a

perfect score for 13/14 questions and an average of around 0.929. Remarkably, relevancy is consistently scored at 1.0 across all questions, showcasing that the retrieved context and the generated answers are always relevant to the queries.

However, there is room for improvement in the correctness metric, which has a lower average score of about 0.580. This suggests that while the model's answers are relevant and based on the context, they do not always align closely with the human reference answers. The variability in this metric, with scores ranging from 0.25 to 0.875, indicates that some answers are well-aligned with human judgments, while others deviate significantly. Moving forward, focusing on improving the correctness of the answers would be beneficial in enhancing the overall performance of the pipeline.

## [NEW] Performance by Task

Since I have divided the questions into four tasks (or types of questions), I want to also evaluate the performance of the pipeline by task. The results are shown in the table below.

```
In [103]: task_performance_df = eval_results_df.groupby('task').mean(numeric_only = True)
task_performance_df["num_questions"] = eval_results_df.groupby('task').count()
task_performance_df
```

```
Out[103]:
```

	semantic_similarity	faithfulness	relevancy	correctness	num_questions
task					
summarize	0.888008	1.000000	1.0	0.656250	4
extract	0.900768	1.000000	1.0	0.625000	5
external	0.943482	0.666667	1.0	0.541667	3
infer	0.827774	1.000000	1.0	0.375000	2

Evaluating the average performance by task, I observe that:

- Semantic Similarity: the 'external' task stands out with the highest average similarity score of 0.943. This makes sense, because for 'external knowledge' task, we essentially compare two LLM-generated responses. On the other hand, the 'infer' task has the lowest average semantic similarity score at 0.828, indicating that generated responses for this task may not be as semantically aligned with the reference.
- Faithfulness: all tasks exhibit a perfect faithfulness score of 1, implying that the model consistently grounds its responses in the given context. This suggests that the model maintains a high level of accuracy in representing context information across all tasks.
- Relevancy: all tasks, except 'external,' have a perfect relevancy score of 1. This indicates that, generally, responses for these tasks are relevant both to the query and the retrieved information. Questions tagged with 'external' mean they're beyond my knowledge base, thus scoring lower relevancy.

- **Correctness:** the 'summarize' task has the highest average score of 0.65 (and 'extract' task comes second), indicating that the model provides answers that closely match the reference for these tasks. It's surprising that 'external' questions score higher than 'infer' questions ( $0.5 > 0.4$ ), because the former is beyond my knowledge base and the latter is within my knowledge base. This may be due to the fact that the 'infer' task requires the model to infer from its understanding of the context, which may not be as accurate as just using the model's own knowledge.

In summary, the analysis reveals that the tasks vary in their performance across different metrics. The 'external' task excels in semantic similarity but lags in relevancy. The 'summarize' task showcases high correctness and perfect faithfulness and relevancy scores. Meanwhile, the 'infer' task struggles with correctness and semantic similarity but maintains perfect faithfulness and relevancy. Since this is a RAG pipeline on my knowledge base, I will continue improve the model's performance on the 'summarize', 'extract', and 'infer' tasks, which are more relevant to my needs of asking questions about my knowledge base.

## [DIFF] Conclusion

In conclusion, my RAG pipeline, designed to answer queries based on a knowledge base of Markdown notes, has demonstrated a commendable performance across a variety of metrics. It particularly excels in maintaining faithfulness to the retrieved context and ensuring the relevancy of responses to the input queries. However, there is an identified need to improve the correctness of the generated answers to ensure they align more closely with human-provided answers. As we move forward, the focus will be on enhancing the accuracy and fine-tuning the model's ability to deliver precise responses, thereby solidifying its utility and reliability as a question-answering system.

## Limitations & Future work directions

While the retrieval-augmented generation (RAG) pipeline has shown promising results, certain limitations have been identified that provide avenues for future work:

### Limitations:

1. **Correctness Variability:** The pipeline exhibits variability in the correctness metric, suggesting that while the model can generate relevant and contextually faithful responses, it does not consistently align with the ground truth provided by human answers.
2. **Semantic Depth:** Despite high relevancy scores, there is room to enhance the semantic depth of the responses, ensuring they not only match keywords and topics but also the subtler nuances of the questions.
3. **Scalability:** The current pipeline's performance under scaling, both in terms of the size of the knowledge base and the volume of queries, is yet to be fully tested and could

present challenges.

### Future Work Directions:

1. **More evaluations:** There are other types of evaluations that can be used to assess the performance of the pipeline, including Traditional NLP (BLEU, ROUGE) and LLM Evals (Helpfulness & Harmlessness).
2. **Refine responses based on Feedback:** Developing a feedback loop where the system learns from its incorrect responses by updating the knowledge base or its retrieval strategy.
3. **Diversified Data Sources:** Expanding the knowledge base beyond Markdown notes to include varied data sources could enhance the robustness of the pipeline, such as by incorporating PDFs, images, and videos.
4. **Deployment & Auto-ingestion:** Deploying the pipeline on a server and automating the ingestion of my knowledge base, to keep it up-to-date with my notes.

By addressing these limitations and pursuing the outlined future work, the RAG pipeline can be evolved into a more accurate, reliable, and user-centric question-answering system.

## AI Statement

I use Github Copilot as the pair programmer for basic syntax (RAG pipeline is beyond its training scope). I also use a combination of external source and perplexity.ai to write the explanation for the transformer architecture and attention mechanism.

## References

Alammar, J. (2019). *The illustrated transformer*. <http://jalammar.github.io/illustrated-transformer/>

AssemblyAI. (2021, November 27). *Transformers for beginners | What are they and how do they work* [Video]. YouTube. [https://www.youtube.com/watch?v=\\_UVfwBqcnbM](https://www.youtube.com/watch?v=_UVfwBqcnbM)

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP tasks. *arXiv (Cornell University)*. <https://arxiv.org/pdf/2005.11401>

Liu, J. (2023, August 16). Using LLM's for retrieval and reranking - LlamaIndex blog. *Medium*. <https://blog.llamaindex.ai/using-llms-for-retrieval-and-reranking-23cf2d3a14b6>

Liu, J., & Datta, A. (2023, November 29). *Building and Evaluating Advanced RAG with LlamaIndex and TruEra*. deeplearning.ai. <https://www.deeplearning.ai/short-courses/building-evaluating-advanced-rag/>

LlaMaIndex. (n.d.). *LlaMAIndex 0.9.15*. LlaMAIndex Documentation.

<https://docs.llamaindex.ai/en/stable>

NexaNodeLabs. (2023, October 30). Fine-Tuning vs RAG: Choosing the Right Approach for Your Question-Answering System. *Medium*. <https://medium.com/@nexanodelabs/fine-tuning-vs-rag-choosing-the-right-approach-for-your-question-answering-system-7c7fbf5d4131>

Raschka, S. (2023, February 9). *Understanding and coding the Self-Attention Mechanism of large Language models from scratch*. Sebastian Raschka, PhD. <https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html>

SBERT. (2022). *Retrieve & Re-Rank*. Sentence-Transformers Documentation. [https://www.sbert.net/examples/applications/retrieve\\_rerank/README.html](https://www.sbert.net/examples/applications/retrieve_rerank/README.html)

SBERT. (2023). *sentence-transformers/all-mpnet-base-v2*. Hugging Face. <https://huggingface.co/sentence-transformers/all-mpnet-base-v2>

Serrano, L., [Serrano.Academy]. (2023, August 31). *The math behind Attention: Keys, Queries, and Values matrices* [Video]. YouTube. [https://www.youtube.com/watch?v=UPtG\\_38Oq8o](https://www.youtube.com/watch?v=UPtG_38Oq8o)

TruEra. (n.d.). *Honest, harmless, helpful Evals*. TruLens. [https://www.trulens.org/trulens\\_eval/core\\_concepts\\_honest\\_harmless\\_helpful\\_evals/#trulens-implementation](https://www.trulens.org/trulens_eval/core_concepts_honest_harmless_helpful_evals/#trulens-implementation)

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All you Need. *arXiv (Cornell University)*, 30, 5998–6008. <https://arxiv.org/pdf/1706.03762v5>

Weng, L. (2023, January 27). The Transformer Family Version 2.0. *Lil'Log*. <https://lilianweng.github.io/posts/2023-01-27-the-transformer-family-v2/>

Yan, E. (2023, July 30). *Patterns for building LLM-based systems & products*. eugeneyan.com. <https://eugeneyan.com/writing/llm-patterns/#retrieval-augmented-generation-to-add-knowledge>

## Appendix

To recreate the virtual environment, run `python >= 3.10`

`python3.11 -m venv venv`  
and install the following packages

```
ipykernel >= 6.4.1  
llama-index >= 0.9.15  
matplotlib >= 3.8.0
```

```
nest-asyncio >= 1.5.1
openai >= 0.10.0
pydantic >= 2.5.0
pytest >= 6.2.5
python-dotenv >= 1.0.0
rouge_score >= 0.1.0
sentence_transformers >= 2.0
tenacity >= 8.2.3
transformers >= 4.10.0 # hugging face
```