The goal of this evaluation framework is to **assess the effectiveness of a retrieval or reranking pipeline** in the context of **Retrieval-Augmented Generation (RAG)** to create a chatbot. Given a set of user queries and their corresponding retrieved or reranked documents, the framework uses an LLM (e.g., gpt-4o-mini or gpt-4.1-mini) to determine whether a relevant answer is present within the top-k results.

🧪 Evaluation Method

For each query in the test set:

1. **Retrieve and rerank documents** using the selected embedding and reranking models.
2. **Label the relevance** of top-k documents using an LLM (binary classification: YES/NO).
3. **Compute metrics** including **Answer Presence@k:** whether any relevant document is present in top-k.

By averaging **Answer Presence@k** across all queries, the framework estimates the likelihood that a system can **retrieve at least one answerable document**, which is crucial for effective RAG performance.

📘 Evaluator Class

The Evaluator class is designed to assess the **relevance of retrieved documents** with respect to a given query using an OpenAI language model (default: gpt-4o-mini). It generates binary labels ("YES"/"NO") for each document's relevance and computes standard evaluation metrics like precision, recall, and accuracy.

🔧 Class Initialization

```
Evaluator(api_key: str, base_url: str, model: str = "gpt-4o-mini",
prompt_template: str = None)
```

Parameters:

- api_key (str): Your OpenAI API key for authentication.
- base_url (str): The base URL of the OpenAI-compatible API endpoint.
- model (str, optional): Model name to be used for evaluation. Defaults to "gpt-4o-mini".

- prompt_template (str, optional): Custom prompt template. If not provided, a default binary relevance classification prompt is used.

📩 Method: get_relevance_labels

```
get_relevance_labels(docs: list[dict], query: str, top_k=None) -> dict
```

Description:
Sends each document-query pair to the model and determines whether the document is relevant to the query based on a "YES"/"NO" response.

Prompt:

```
prompt = f"""Here are a question and a retrieved passage from a text corpus from
the same domain as the question.

Can you judge whether an answer to the question can be derived from the retrieved
passage, simply answer either “YES” or “NO”.

<binary>

Question: {query}; Retrieved Passage: {doc_text}"""
```

Parameters:

- docs (list[dict]): A list of documents, where each item is a dictionary with at least a "document" key containing the document text.
- query (str): The user query for which relevance is being assessed.
- top_k (int, optional): Only evaluate the top k documents from the list.

Returns:

- A dictionary {document_text: relevance_label} where:
    - relevance_label is 1 if the model says "YES", otherwise 0.

Side Effects:

- Updates internal state:
    - self.relevance_dict: Latest document-to-label map.

- self.relevance_labels: List of relevance labels in ranked order.

📊 Method: compute_metrics

```
compute_metrics(self, ks: list[int] = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], show = False) -> dict
```

Description:
Computes evaluation metrics (Precision, Recall, Accuracy, Answer Presence) at multiple cutoff values of top-k using the most recent labels from get_relevance_labels.

Parameters:

- ks (list[int], optional): List of cutoff values k for computing top-k metrics.
- show (bool, optional): If True, prints metric values for each k.

Returns:

- Dictionary of metrics keyed by @k, each containing:
  - "Accuracy ": TP / k
  - "Number of Relevant Docs": Number of true positives in top-k
  - "Answer Presence": Binary 1/0 depending on whether any relevant doc is in top-k

Raises:

- ValueError: If get_relevance_labels() has not been called before metrics computation.

📦 Attributes

- client: Instance of OpenAI, initialized with provided API key and base URL.
- model: Model used for document relevance evaluation.
- prompt_template: Prompt used to instruct the model for binary relevance classification.
- relevance_dict: Last computed {document_text: label} mapping.
- relevance_labels: Ordered list of latest labels (1 for relevant, 0 for not) used for metrics.

🧠 Example Usage

```
evaluator = Evaluator(api_key="...", base_url="...")

docs = [
    {"document": "The capital of France is Paris."},
    {"document": "Bananas are rich in potassium."}
]
query = "What is the capital of France?"

labels = evaluator.get_relevance_labels(docs, query)
metrics = evaluator.compute_metrics(show=True)
```

📈 Evaluating Embedding Models and Rerankers Using the Evaluator Class

🔍 Purpose

The Evaluator class can be extended beyond a single query-document pair. When used with a **test set** containing **multiple queries**, it allows you to compare the performance of different **embedding models**, **reranking strategies, etc.** in information retrieval pipelines.

✅ Evaluation Strategy

For each query in the test set:

1. Use Evaluator.get_relevance_labels() to label the top-k documents for that query.
2. Use Evaluator.compute_metrics() to calculate metrics like **Precision**, **Recall**, and especially **Answer Presence**.
3. Store the **Answer Presence at each k** for every query.

After processing all queries, compute the **average Answer Presence at each top-k**

⚙️ What can be Evaluated?

- **Embedding Model**
  The vector embedding model used to encode and retrieve documents:
    - Examples: OpenAI, BGE

- **Top-K** (top_k)
  The number of **retrieved documents** to evaluate per query (e.g., 20, 30).
  These represent the initial output of the retriever.

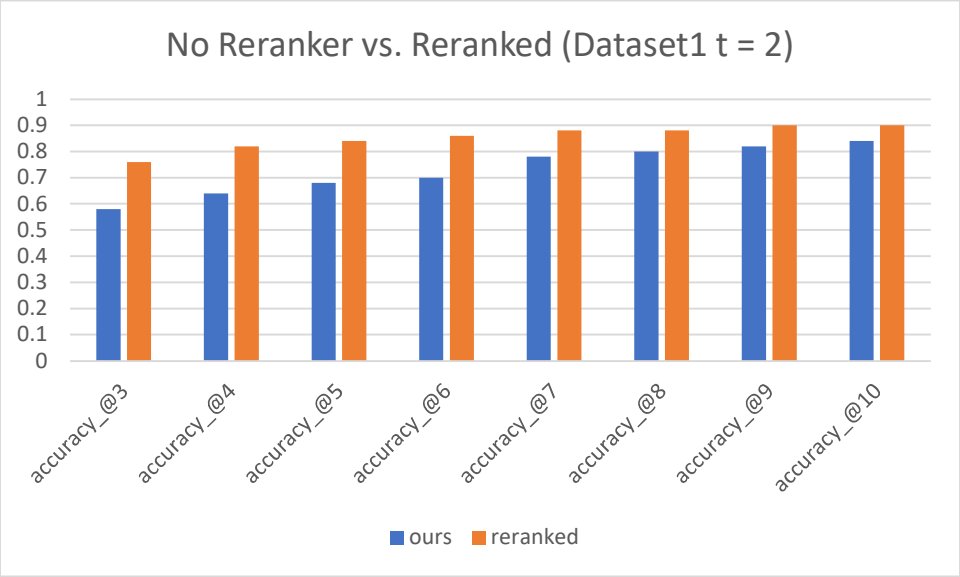
- **Top-N** (top_n)
  The number of **documents sent to the LLM** for final answer generation.
  These are typically selected after reranking (N ≤ K).

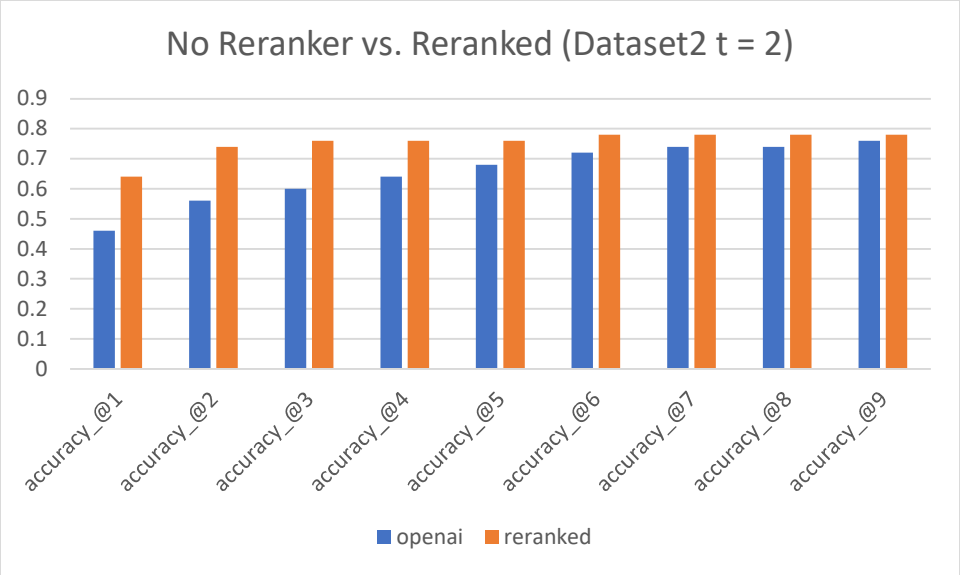
- **Semantic Chunker Threshold**
  Controls how documents are chunked semantically (results below use std).
  Threshold values usually range from **1.5 to 3.0**, affecting chunk size and coherence.

- **Reranking Model**
  The reranker used to reorder retrieved documents based on query-document relevance:
    - Examples: FlashRank, Cohere, Jina Rank

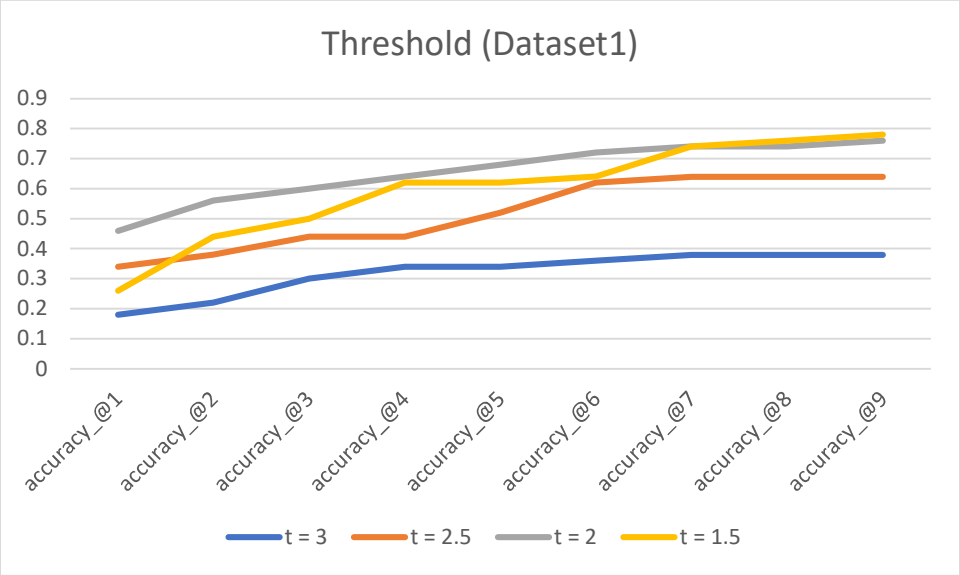
Results

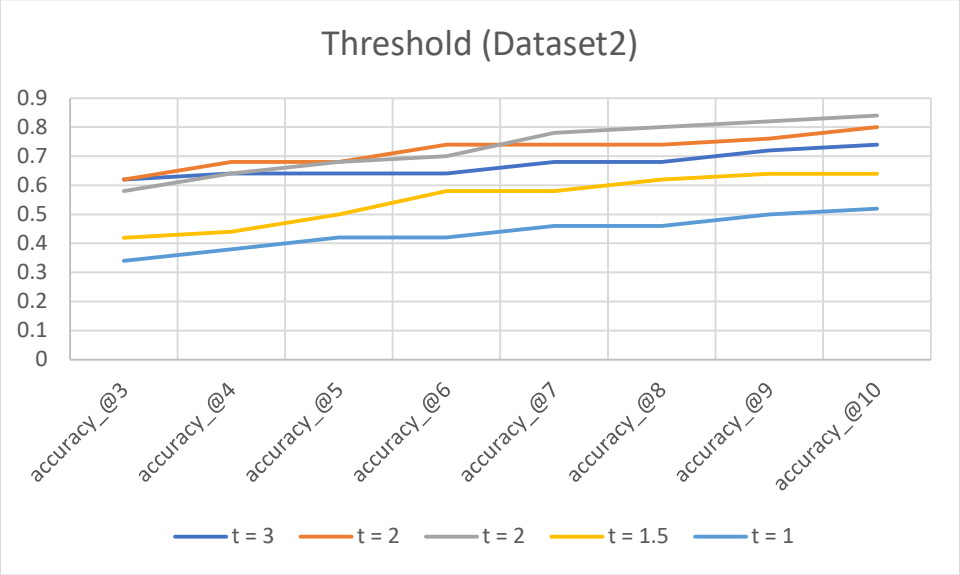
Reranker vs. No Reranker:



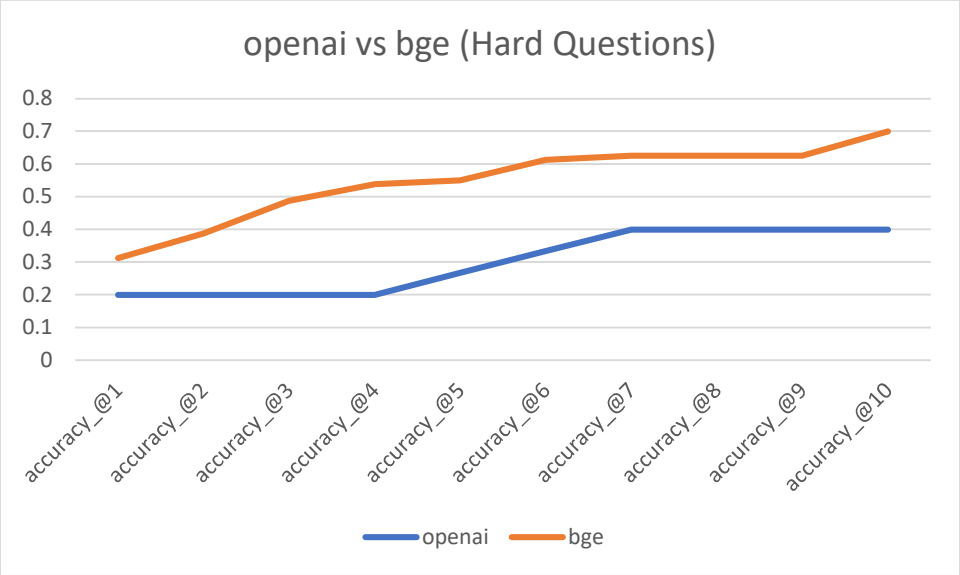


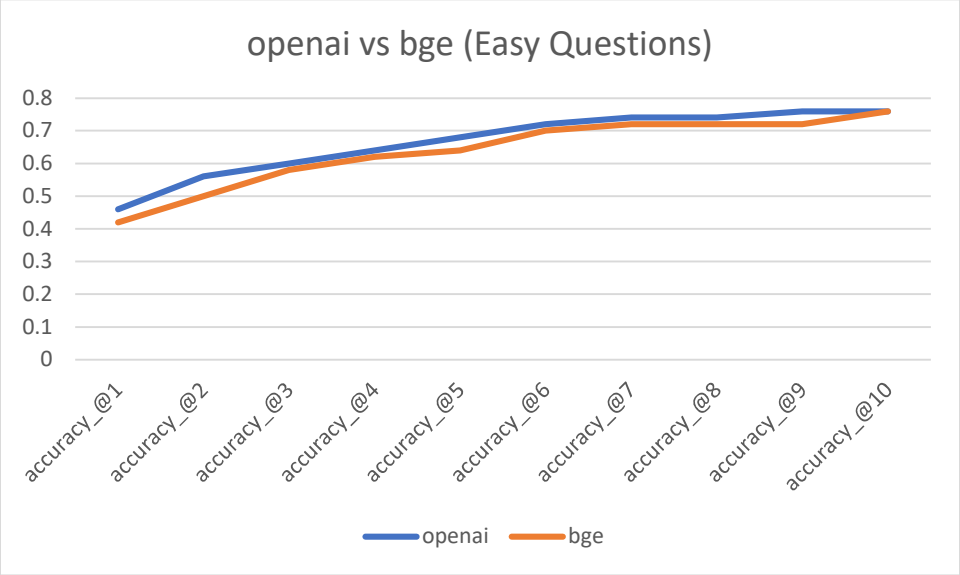
Threshold:


Threshold (Dataset1)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
accuracy_@1
accuracy_@2
accuracy_@3
accuracy_@4
accuracy_@5
accuracy_@6
accuracy_@7
accuracy_@8
accuracy_@9
t = 3
t = 2.5
t = 2
t = 1.5


Threshold (Dataset2)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
accuracy_@3
accuracy_@4
accuracy_@5
accuracy_@6
accuracy_@7
accuracy_@8
accuracy_@9
accuracy_@10
t = 3
t = 2
t = 2
t = 1.5
t = 1

bge vs openai: (t = 2)





4.1-mini vs 4o-mini: