Ensemble Retriever

The `EnsembleRetriever` takes a list of retrievers as input and ensemble the results of their `get_relevant_documents()` methods and rerank the results based on the Reciprocal Rank Fusion algorithm.

By leveraging the strengths of different algorithms, the `EnsembleRetriever` can achieve better performance than any single algorithm.

The most common pattern is to combine a sparse retriever (like BM25) with a dense retriever (like embedding similarity), because their strengths are complementary. It is also known as "hybrid search". The sparse retriever is good at finding relevant documents based on keywords, while the dense retriever is good at finding relevant documents based on semantic similarity.

```
%pip install --upgrade --quiet  rank_bm25 > /dev/null
```

```
from langchain.retrievers import
BM25Retriever, EnsembleRetriever
from langchain_community.vectorstores import
FAISS
from langchain_openai import OpenAIEmbeddings
```

```
doc_list_1 = [
    "I like apples",
    "I like oranges",
    "Apples and oranges are fruits",
]

# initialize the bm25 retriever and faiss
retriever
bm25_retriever = BM25Retriever.from_texts(
    doc_list_1, metadatas=[{"source": 1}] *
len(doc_list_1)
)
bm25_retriever.k = 2

doc_list_2 = [
    "You like apples",
    "You like oranges",
]

embedding = OpenAIEmbeddings()
faiss_vectorstore = FAISS.from_texts(
    doc_list_2, embedding, metadatas=
[{"source": 2}] * len(doc_list_2)
```

```
)
faiss_retriever =
faiss_vectorstore.as_retriever(search_kwargs=
{"k": 2})

# initialize the ensemble retriever
ensemble_retriever = EnsembleRetriever(
    retrievers=[bm25_retriever,
faiss_retriever], weights=[0.5, 0.5]
)
```

```
docs = ensemble_retriever.invoke("apples")
docs
```

```
[Document(page_content='You like apples',
metadata={'source': 2}),
 Document(page_content='I like apples',
metadata={'source': 1}),
 Document(page_content='You like oranges',
metadata={'source': 2}),
 Document(page_content='Apples and oranges
are fruits', metadata={'source': 1})]
```

Runtime Configuration

We can also configure the retrievers at runtime. In order to do this, we need to mark the fields as configurable

```
from langchain_core.runnables import
ConfigurableField
```

```
faiss_retriever =
faiss_vectorstore.as_retriever(
    search_kwargs={"k": 2}
).configurable_fields(
    search_kwargs=ConfigurableField(
        id="search_kwargs_faiss",
        name="Search Kwargs",
        description="The search kwargs to
use",
    )
)
```

```
ensemble_retriever = EnsembleRetriever(
    retrievers=[bm25_retriever,
faiss_retriever], weights=[0.5, 0.5]
)
```

```
config = {"configurable":
{"search_kwargs_faiss": {"k": 1}}}
docs = ensemble_retriever.invoke("apples",
```

```
    config=config)
docs
```

Notice that this only returns one source from the FAISS retriever, because we pass in the relevant configuration at run time