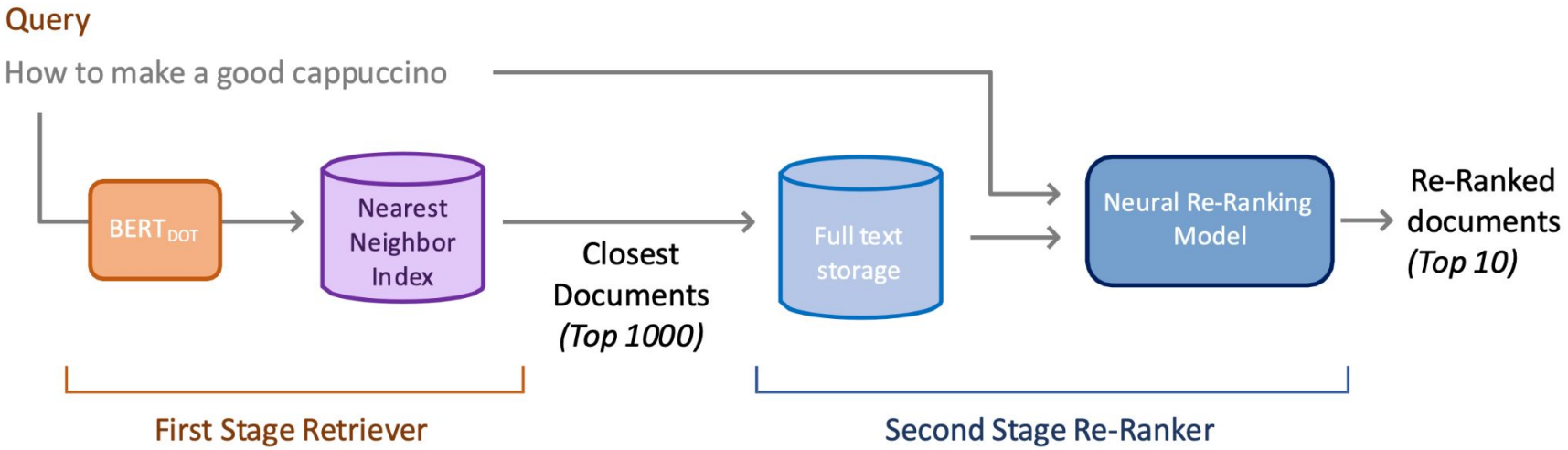
Introduction to LLM

Practice Session 13

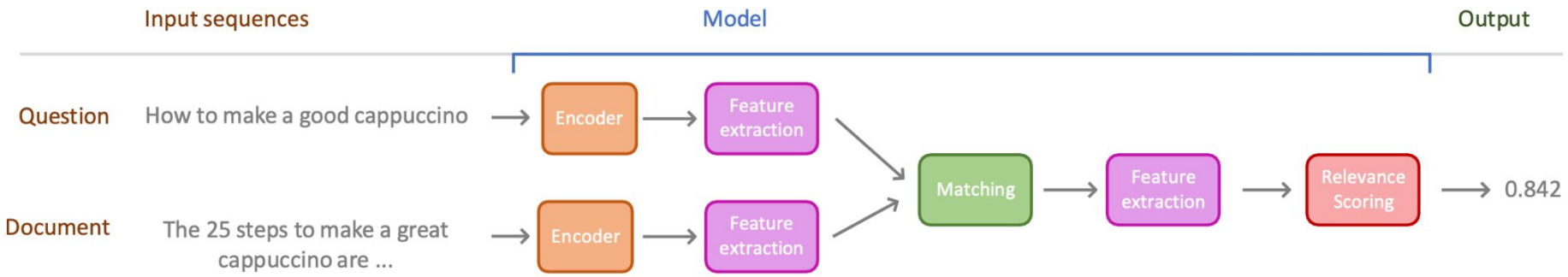
Rerankers, RAG, and Agents

High-level Retriever + Reranker Pipeline



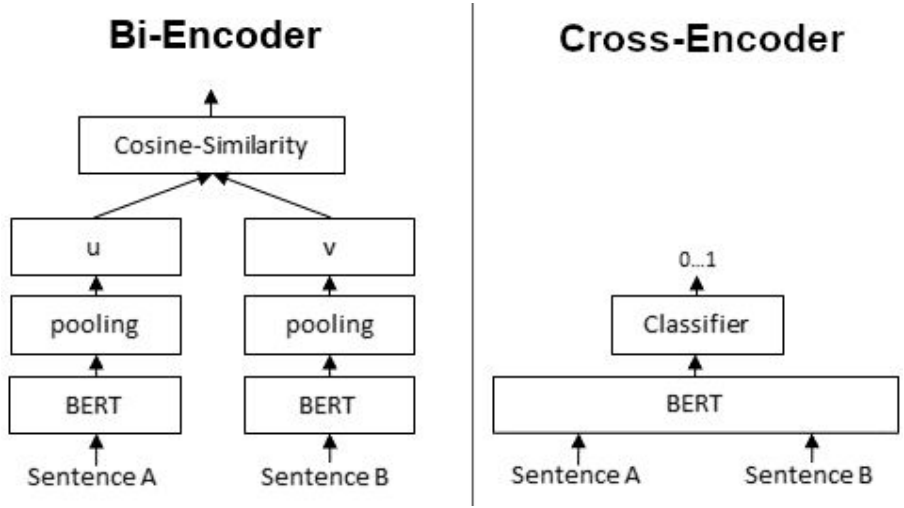
- First stage retrievers can be:-
  - Dense retrievers, e.g., BERT-DOT
  - Sparse retrievers, e.g., BM25
  - Hybrid retrievers (combination of both)

Reranker Main Idea (Interaction Modelling)



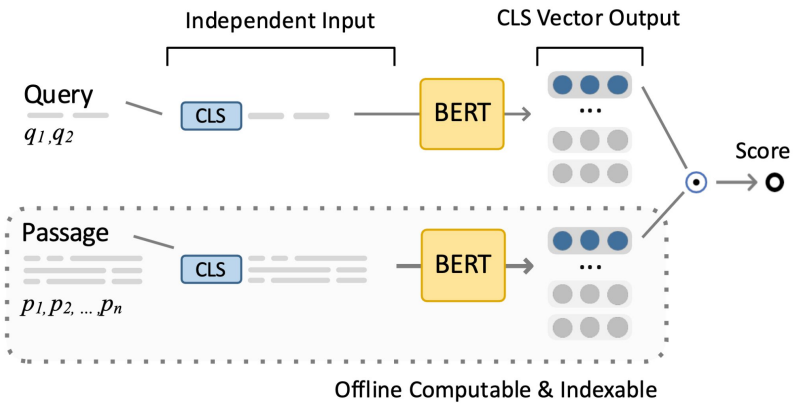
- Rerankers model the interaction between the query and the document
  - Dense retrieval treat the query encoding and document encoding separately (no interaction)
  - Reranker learns to generate one encoding for both query and document together (interaction)
    - This one encoding for both is now used to generate one scalar relevance score

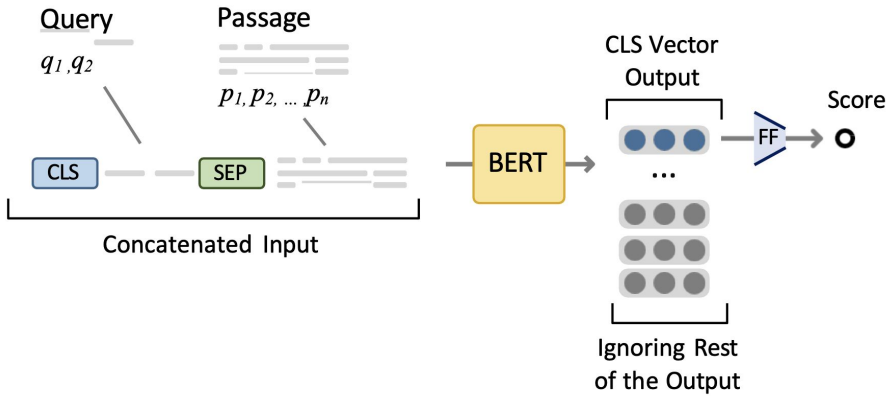
Bi-encoder (Dense Retriever) vs Cross-encoder (Reranker)



- Source: [Article Link]
- Cross-encoders takes both the query (sentence A) and the document (sentence B) simultaneously
    - It learns a classifier (simple NN) to generate one output (score) between 0 and 1

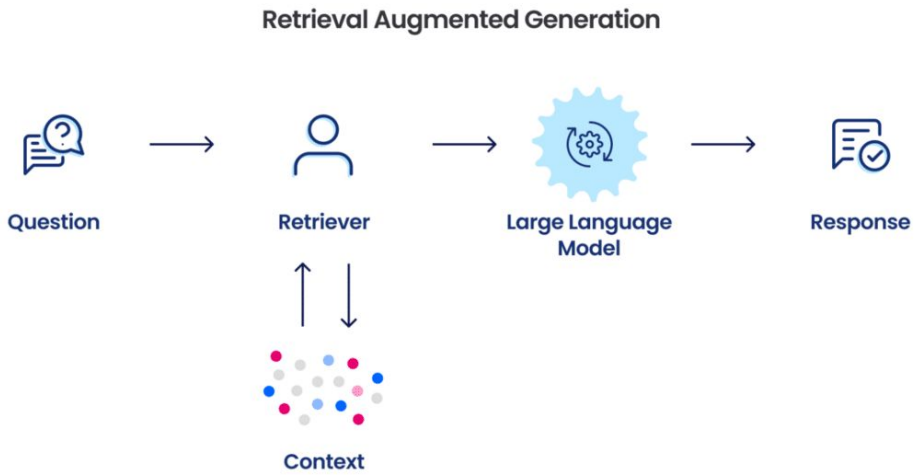
Bi-encoder (BERT-DOT) vs Cross-encoder (BERT-CAT)





- Rerankers (Cross-encoder) is slower as it must be done online (at query time)
  - It's better than bi-encoders in detecting relevance between query and document
  - It acts as a verifier that checks only the top k (e.g., 100) documents from bi-encoders and re-rerank them

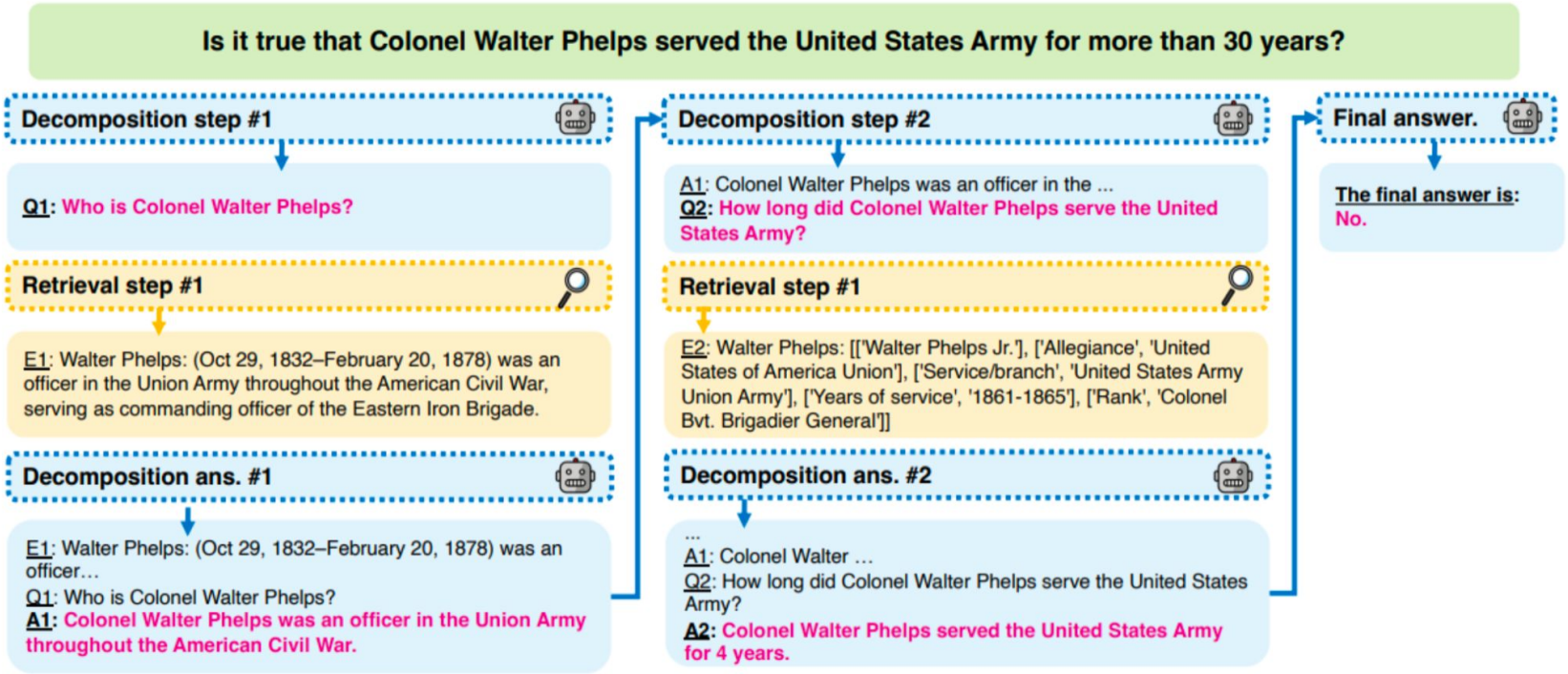
Retrieval-Augmented Generation (RAG)



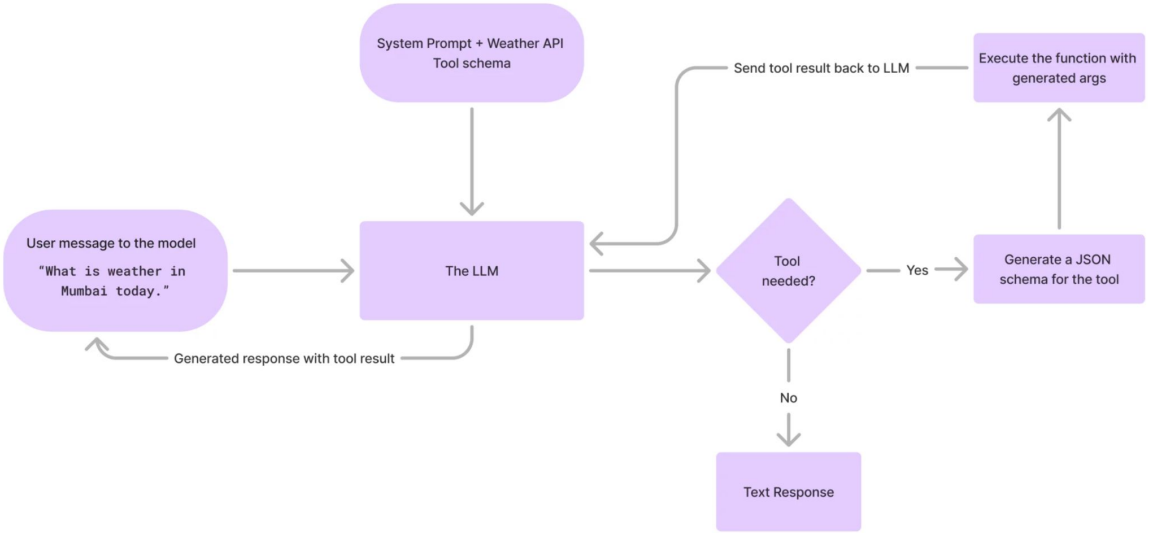

```
Context information is below.
---------------------
{context_str}
---------------------
Given the context information and not
prior knowledge, answer the query.
Query: {query_str}
Answer:
```

- Source: [Article Link]
- Step 1: retrieve top N documents using some Retriever pipeline.
- Step 2: write the query with the top N documents (context) for the LLM.

RAG (Interleaving Decomposition) for Distraction Mitigation



Tool Use with LLMs



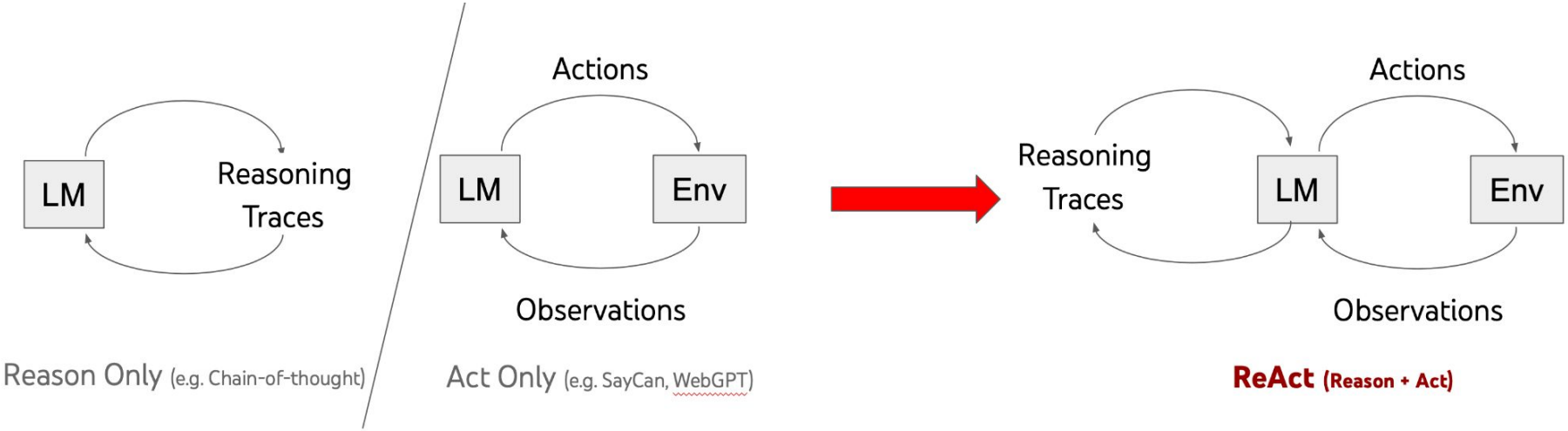

```python
def add(a: int, b:int) -> int:
    """Adds two integers together"""
    return a + b
```

```json
{
    "type": "function",
    "function": {
        "name": "add",
        "description": "Adds two integers together",
        "strict": true,
        "parameters": {
            "type": "object",
            "required": [
                "a",
                "b"
            ],
            "properties": {
                "a": {
                    "type": "integer",
                    "description": "The first integer to add"
                },
                "b": {
                    "type": "integer",
                    "description": "The second integer to add"
                }
            },
            "additionalProperties": false
        }
    }
}
```

Tool Schema (can be simple string)

- Source 1: Article Link, Source 2: Article Link
- LLMs are fine-tuned on numerous tool usage examples (mostly in JSON)

ReAct: Synergizing Reasoning and Acting in Language Models



- Source 1: [Article Link]

- LLMs are strong at reasoning, but weak at exact arithmetic calculations, or answering based on up-to-date information (training data has a cut-off date) -> Solution: use both thinking + actions (tools)

Timeline

Python & ML Foundations

- Train/Dev/Test
- Hyperparameters
- Loss Fn.

CL/NLP Basics

- Morphology
- Syntax
- PoS
- Parsing
- Semantics
- Lemma
- Ambiguity

Word Representation I:

- Basics
- Evaluation
- Inverted Index
- TF-IDF
- BM25
- LSA
- nDCG

Word Representation II:

- Statistical Methods
- Cosine Similarity
- Word2Vec
- CBOW & Skip-gram
- GloVe

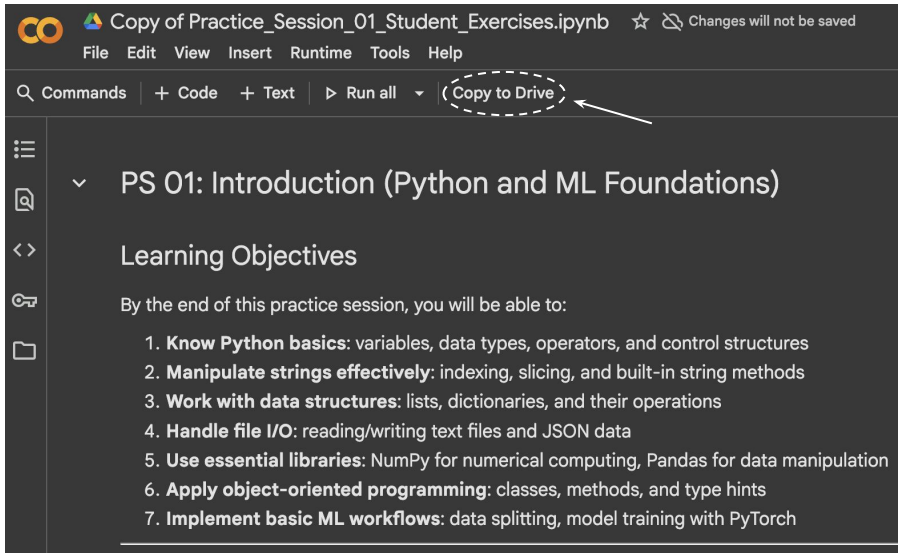
Dense Retrieval QA, and ANN

- Bi-encoders
- BERT-DOT model
- Hard Negatives
- Margin Loss Training
- Span QA
- Indexing Comparison

Rerankers, RAG, and Agents

- BERT-CAT
- RAG
- Agent
- Tool Use
- ReAct

PS13: Colab Notebook (Available on Moodle)



- https://colab.research.google.com/drive/1CQHvt18Y_tujmPTNGeLxLrgzmAxijyvM
- Before running any cell, please choose T4 GPU by clicking on "Runtime" in the main menu then on "Change runtime type", and disconnect from it when you finish using it.