Retrieval + Tools

Andrew Tang

COMS 4705 NLP

Columbia Engineering

Motivation

Outdated info:

Q: Who is the US President?

A: Joe Biden ❌

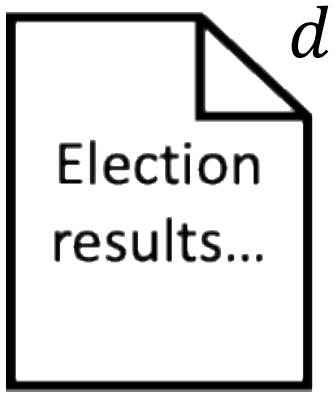
Hallucination / Vague:

Q: What is the population of NYC as of 2024?

A: About 8.3 million 👎

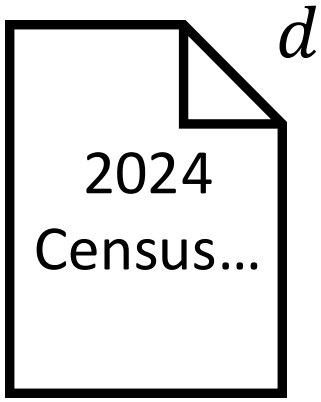
With Retrieval

Q: Who is the US President?

Retrieve(Q) -> 

A: Donald Trump

Q: What is the population of NYC as of 2024?

Retrieve(Q) -> 

A: 8,478,072 as of July 2024

Retrieval Augmented Generation (RAG)

- Suppose we have a question q and want answer a
- Before: $p_\theta(\, a \mid q \,)$
- RAG: $p_\theta(a \mid q, R(q))$, $R(q)$ retrieves a set of documents
- How should R work?
  - Relevant docs
  - Fast

Lewis 2020

Retrieval Augmented Generation (RAG)

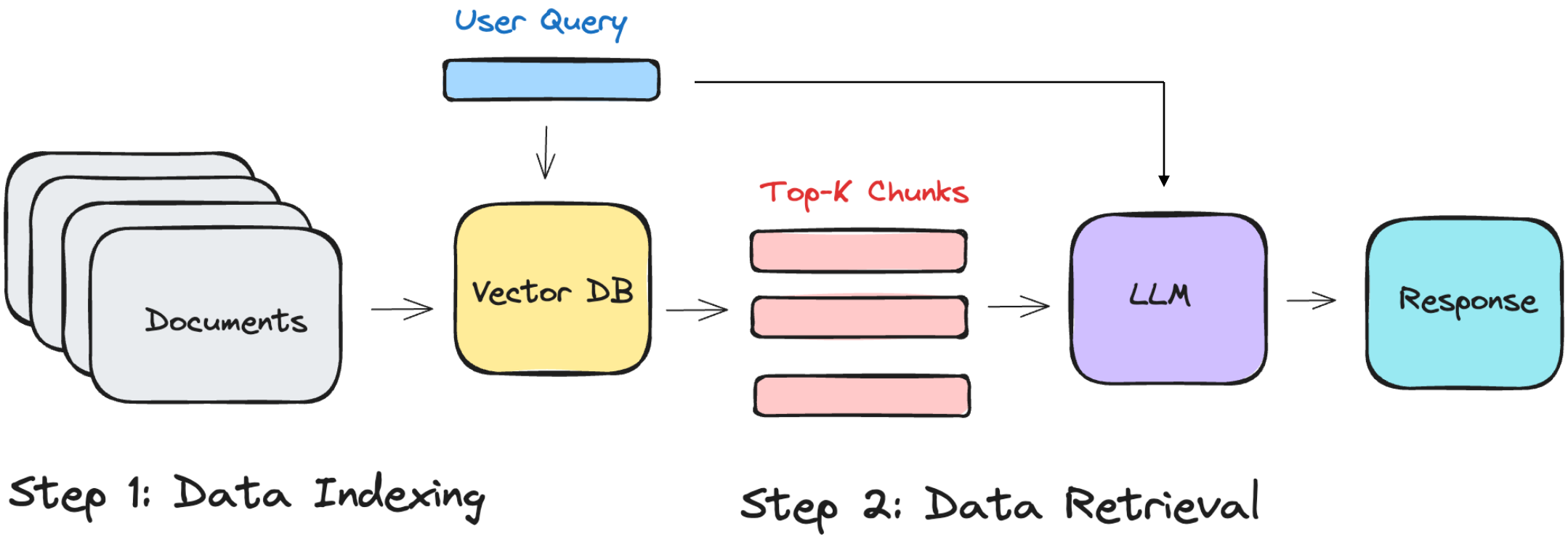
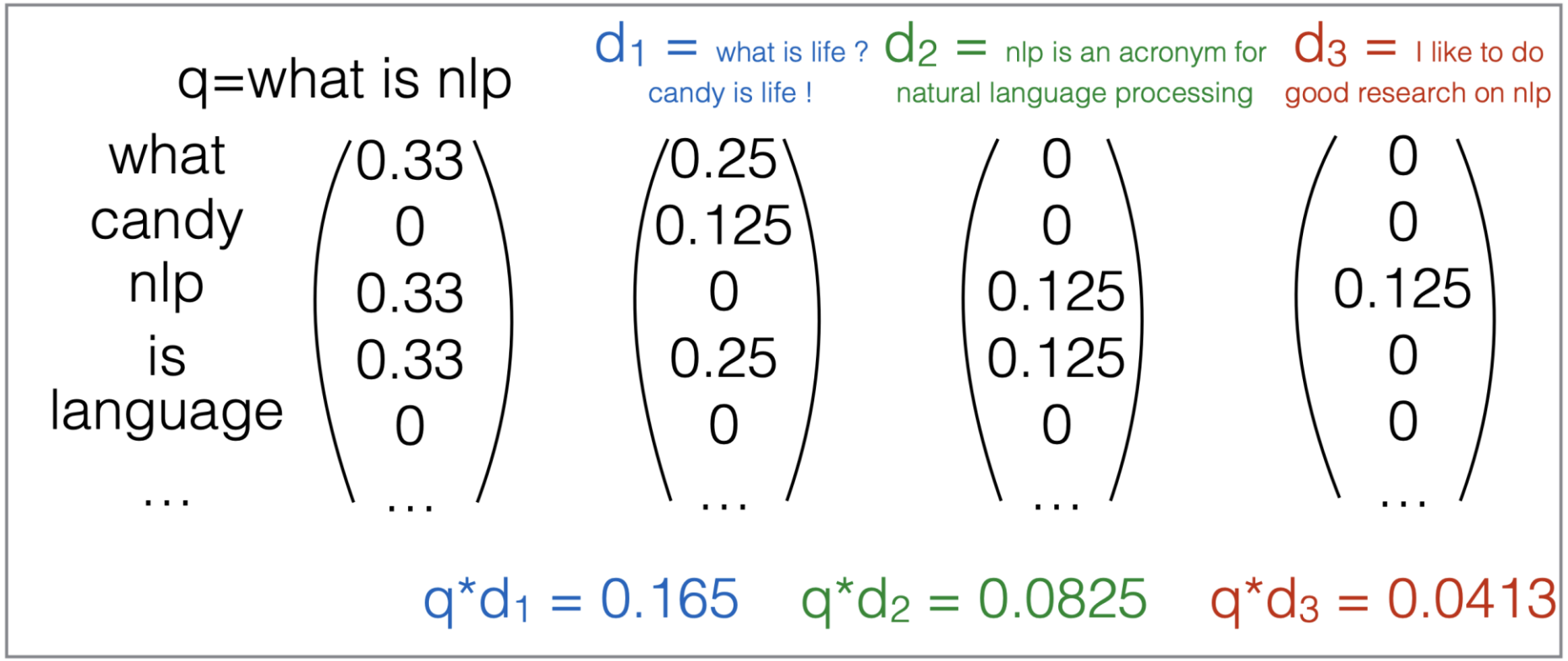


Fig from [1]

Sparse Retrieval

- Express query and doc as a bag-of-words (BoW) vector, normalized by length



What's the problem with this?

Fig from [2]

Better Sparse Retrieval

- Common words like “what” not important but contribute high score
- TF-IDF (term frequency $*$ inverse-doc frequency):
  - $\mathrm{TF}(w, d) = \frac{count(w,d)}{|d|}$
  - $\mathrm{IDF}(w, D) = \log \frac{|D|}{1+|\{d \in D \mid w \in d\}|}$ Number of docs with w
  - score = TF $*$ IDF

Do this per word to get a BoW vector

Problem with TF-IDF

- TF growth is linear (if w is 2x more frequent, score is 2x)
  - Saturation / over-emphasized
  - Doc length bias
- Improvement is BM25 (Best-Match 25)

$$\text{score}(w,d) = \text{IDF}(w,D) \cdot \frac{count(w,d) \cdot (k_1 + 1)}{count(w,d) + k_1 \cdot (1 - b + b \cdot \frac{|d|}{N_{avg}})}$$

  - k_1, b are hyperparams, N_{avg} is the average document length

Inverted Index

- Data structure for efficient sparse vector search

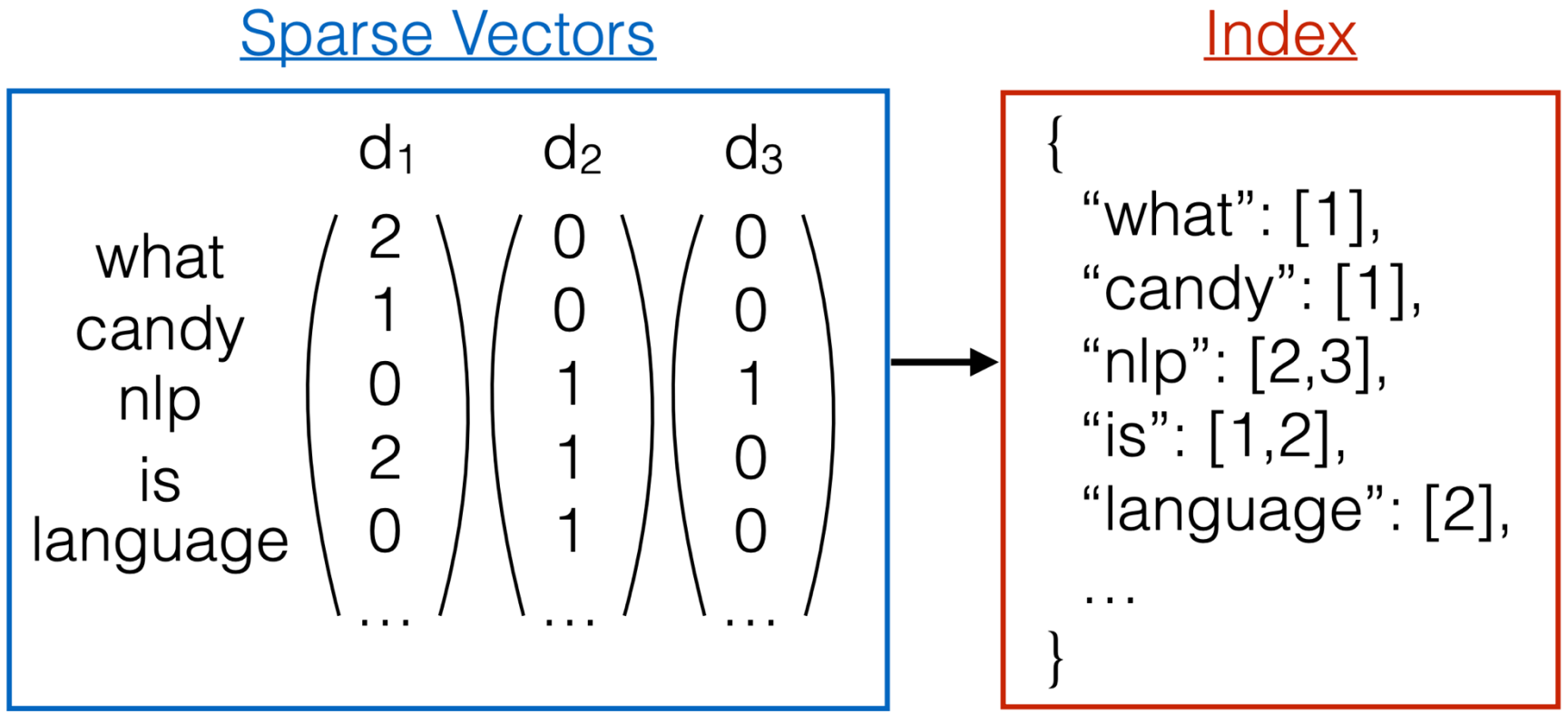


Fig from [2]

From Sparse to Dense Retrieval

Problems with sparse retrieval (TF-IDF & BM25):

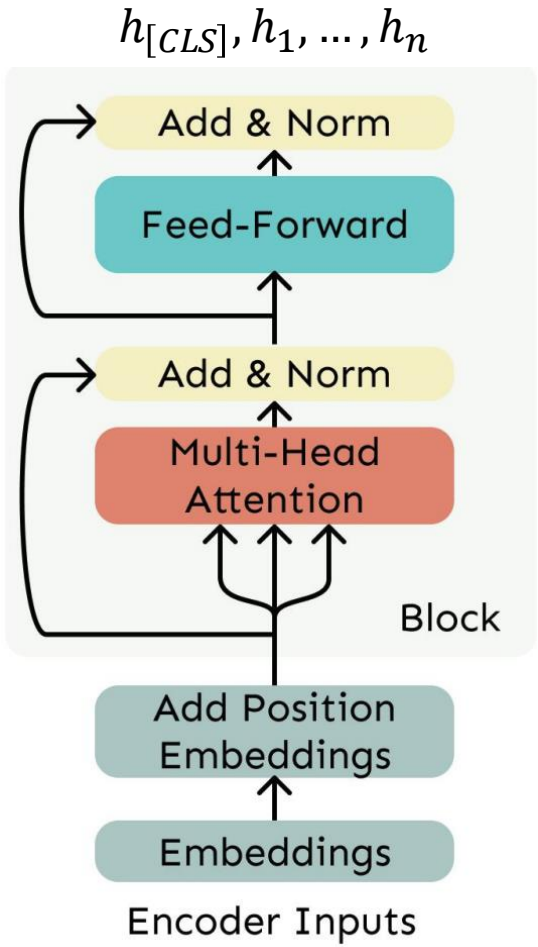
- No synonyms
- No context understanding (treats words independently)

Modern approach: dense embeddings

- Encode query/doc and find nearest neighbor
- Twist on transformer encoder

BERT Encoder

- Trained with Masked LM Objective
  - [MASK] some input tokens, predict
- Prepend input with special [CLS] token
- Take $h_{[CLS]}$ as the embedding for the input sequence
  - $E(d) = h_{[CLS]} \in \mathbb{R}^{768}$
- Why can't we directly use this for retrieval?



Learning Retrieval-oriented Embeddings

- Goal: Relevant query/docs are close together and vice-versa
- Idea: Train Encoder E using contrastive loss (NLL of positive doc)
- Data: tuples of $(q, d^+, d_1^-, \ldots, d_n^-)$

$$L(q) = -\log \frac{\exp(E(q)^\top E(d^+))}{\exp\left(E(q)^\top E(d^+)\right) + \sum_{j=1}^{n} \exp(E(q)^\top E(d_j^-))}$$

DPR (Karpukhin 2020)

Constructing contrastive examples

- Raw data: query + relevant doc pairs

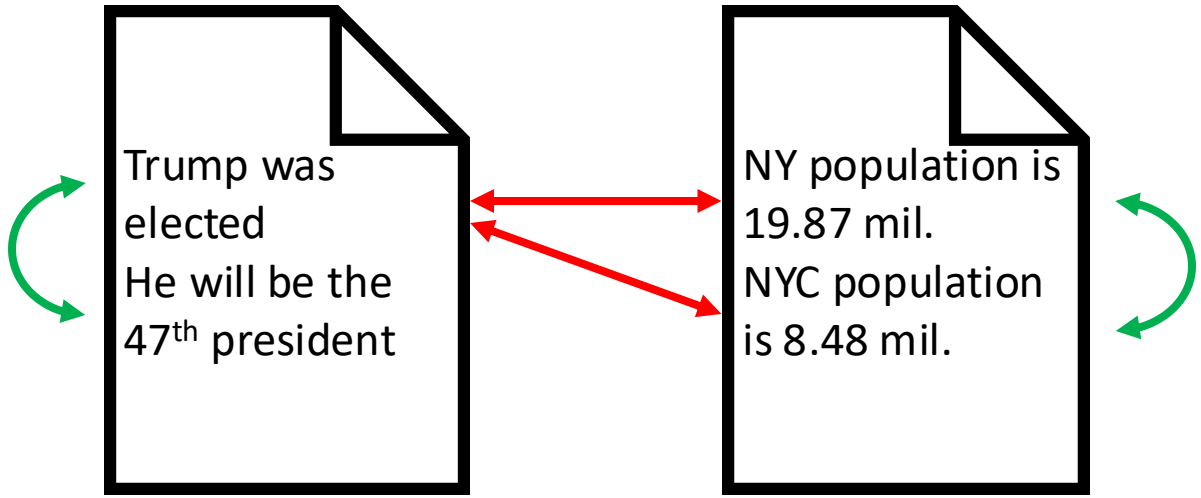
q = what is NLP,
d^+ = nlp is natural language processing
d_1^- = what is candy
d_2^- = I like NLP

- In-batch negatives:
  - Use every other doc in mini-batch as negative doc
  - Problem: Too easy, limited by batch size
- Hard negative:
  - Get most similar (but actually irrelevant) doc, via BM25 or current encoder
  - Problem: False negative

d_H^- = NLP is natural language processes

Unsupervised Contrastive Learning

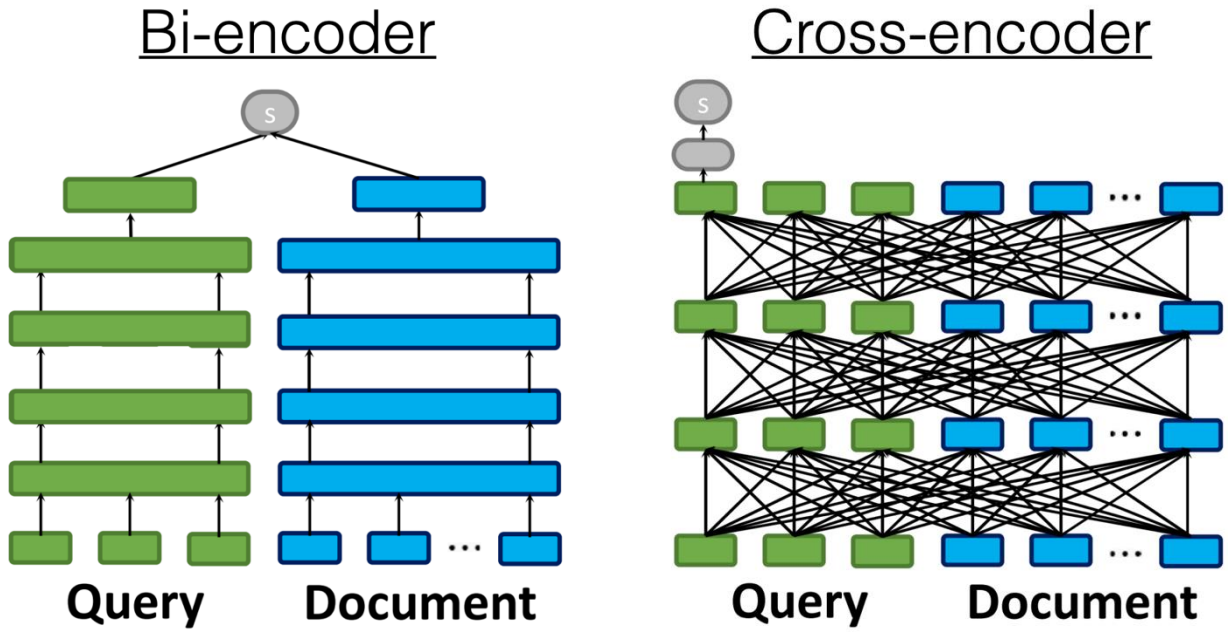
- So far required labelled data (question-doc pairs)
- Idea: use two random spans (chunks) from the *same doc* as positive pairs
  - Chunks from other docs are negatives



Contriever (Izacard 2022)

Cross-encoder

- Problem with (bi-)encoder:
  - Collapses q/doc to single vector
  - One-size-fits-all



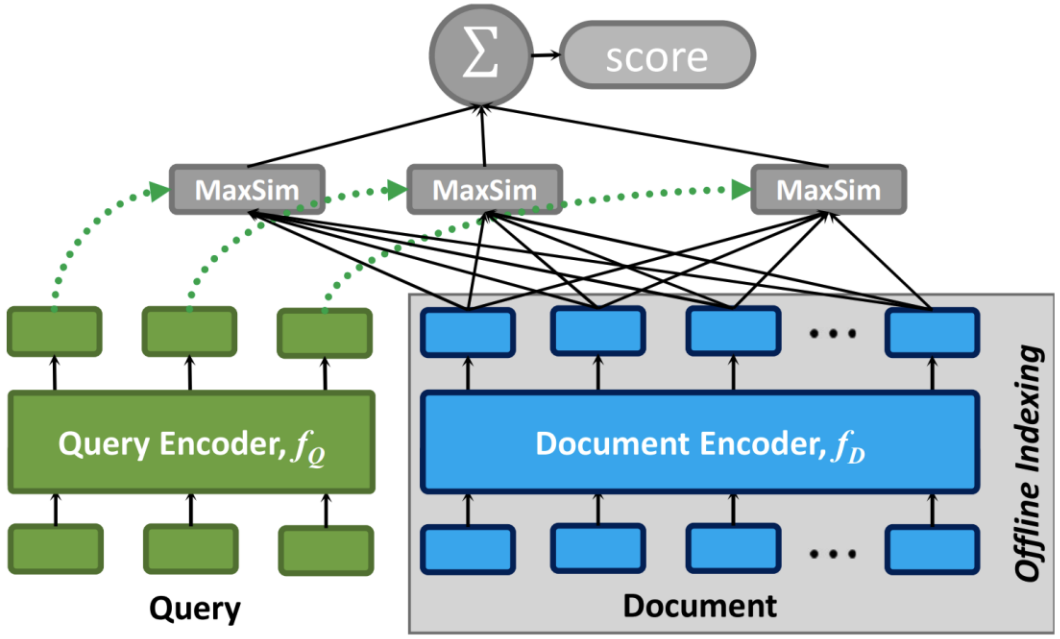
- Alternative:
  - Train encoder that directly scores query and doc
  - $sim(q, d) = E(q, d) = f(h_{[CLS]})$

- This is expensive – need a single forward pass per query-doc pair

Nogueira 2020

Token-level Dense Retrieval

- ColBERT
  - Convert query and doc into bag-of-embeddings $\{h_{q_i}\}_{i=1}^{|q|}, \{h_{d_j}\}_{j=1}^{|d|}$
  - $sim(q,d) = \sum_i \max_j h_{q_i}^\top h_{d_j}$
- In-between bi and cross-encoder
  - Cheaper (but worse) than cross
  - Better (but more costly) than bi



Khattab 2020

Context Assembly

- Simple: Context = query + retrieved docs
- Problem:
  - Massive context window
  - Conflicting or redundant docs hurt generation accuracy
- Two-stage retrieval:
  1. Fast (but noisy) retrieval (BM25, dense)
  2. Rerank with expensive cross-encoder (then truncate)

Diversify our Context Docs

- Maximal Marginal Relevance (MMR):

$$\arg\max_{d \in C \setminus S} [\lambda \cdot sim(q, d) - (1 - \lambda) \max_{d' \in S} sim(d, d')]$$

- C is our set of retrieved docs, S is our new set of docs
- Balance sim to query vs. sim to selected docs

Carbonell 1999

More Advanced RAG Topics

- Better Single-shot
  - Hybrid (BM25 + Dense + Rerank)
  - Query decomposition (into subqueries)
- Iterative Retrieval
  - Query -> Reason -> Subquery -> Retrieve -> Filter -> Generate

- End-to-End Retriever Training
  - Fine-tune query encoder via generator’s loss

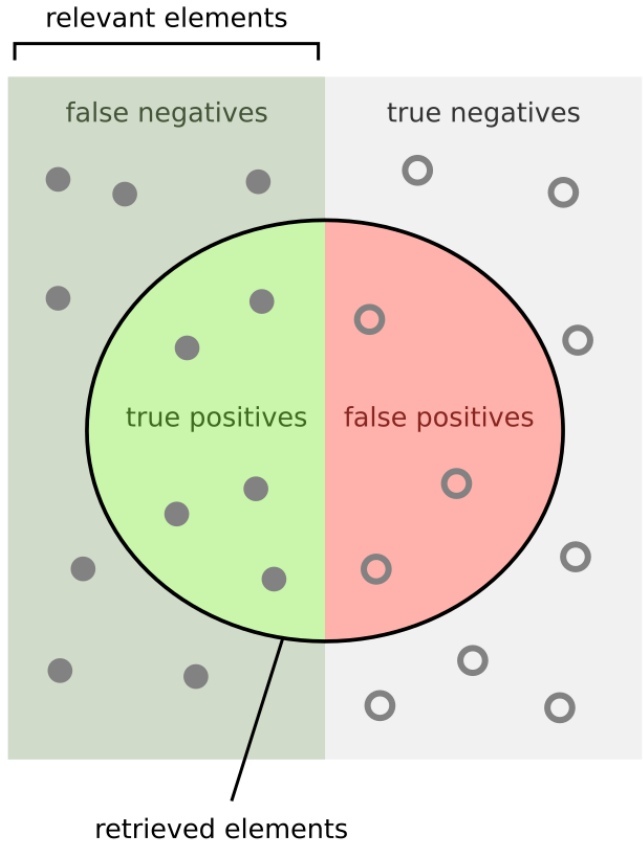
Evaluation Metrics

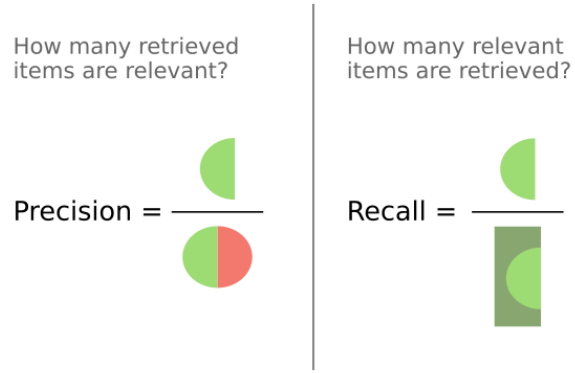
Retrieval

- Precision@k, Recall@k
- MRR@k (Mean Reciprocal Rank)
  - 1/r if d^+ is rank r

Generation

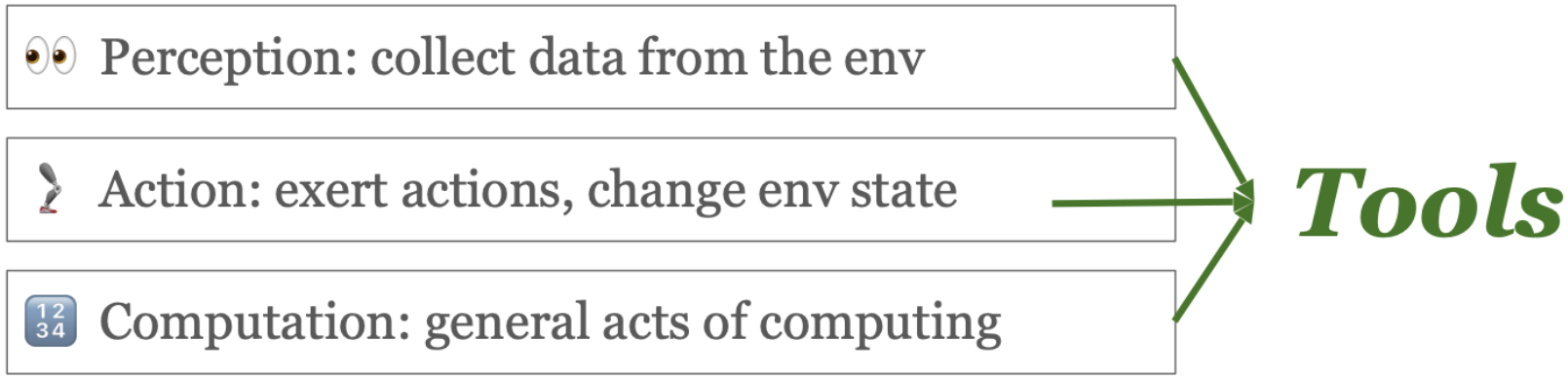
- Same as downstream task (i.e. QA)





Tool Use

Motivation



- Retriever is a tool
- Example tools: search, calculators, code exec, APIs (shopping plugin…)
- Reasoning + tool use => ↑ performance on QA, interactive env, etc..

Framework

- State s_t = context (chat history, reasoning trace, tool outputs, etc..)
- Actions $a_t \in$ {"call tool i with args"}
- Transition: $s_{t+1} = f(s_t, a_t)$
- Trajectory $\tau = s_1, a_1, \ldots, s_T, a_T$
- Want to learn θ that maximizes $\mathrm{E}_{\tau \sim \pi_\theta(a_t \mid s_t)}[r(\tau)]$

Toolformer

- Augment dataset with API calls
  - Insert API if its output decreases loss over next tokens
- Finetune

The New England Journal of Medicine is a registered trademark of **[QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society]** the MMS.

Out of 1400 participants, 400 (or **[Calculator(400 / 1400) → 0.29]** 29%) passed the test.

The name derives from "la tortuga", the Spanish word for **[MT("tortuga") → turtle]** turtle.

The Brown Act is California's law **[WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.]** that requires legislative bodies, like city councils, to hold their meetings open to the public.

(Schick 2023)

Your task is to add calls to a Question Answering API to a piece of text. The questions should help you get information required to complete the text. You can call the API by writing "[QA(question)]" where "question" is the question you want to ask. Here are some examples of API calls:

Input: Joe Biden was born in Scranton, Pennsylvania.

Output: Joe Biden was born in **[QA("Where was Joe Biden born?")]** Scranton, **[QA("In which state is Scranton?")]** Pennsylvania.

Input: Coca-Cola, or Coke, is a carbonated soft drink manufactured by the Coca-Cola Company.

Output: Coca-Cola, or **[QA("What other name is Coca-Cola known by?")]** Coke, is a carbonated soft drink manufactured by **[QA("Who manufactures Coca-Cola?")]** the Coca-Cola Company.

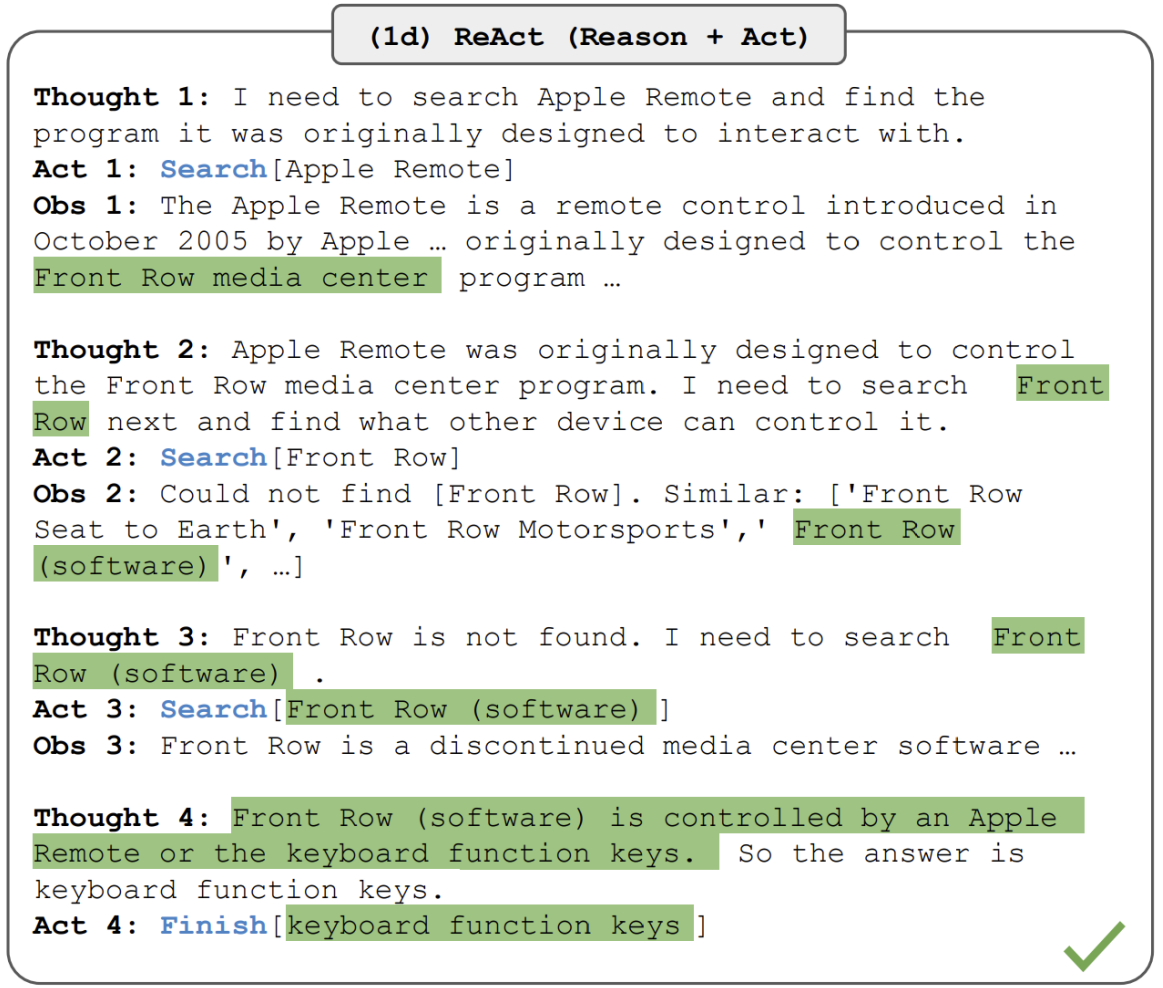
Input: x

Output:

- Where to insert API calls?
  - $p_i = p_M(\ [\ |\ \text{prompt}, x_{<i})$
  - Pick i's with highest p_i
- Obtain API calls for each i
  - Prefix: (prompt, $x_{<i}$, [ )
- Execute each API call
- Filter out API calls that decrease loss of next tokens

ReAct (Reason + Act)

- Prompting method
- “Thought” + “Action” + “Observation”



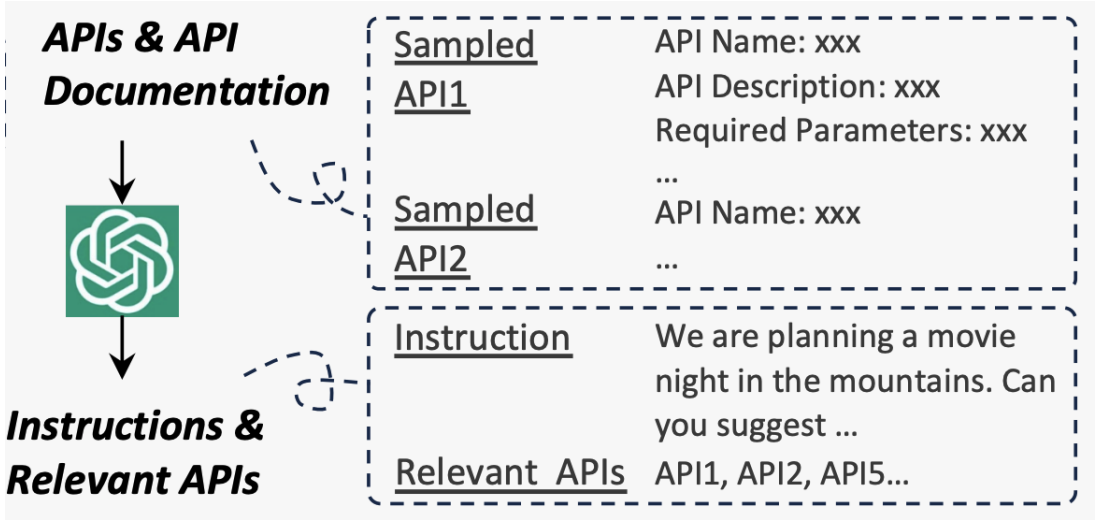
(Yao 2023)

ReAct Example Prompt

Question	Musician and satirist Allie Goertz wrote a song about the "The Simpsons" character Milhouse, who Matt Groening named after who?
Thought 1	The question simplifies to "The Simpsons" character Milhouse is named after who. I only need to search Milhouse and find who it is named after.
Action 1	Search[Milhouse]
Observation 1	Milhouse Mussolini Van Houten is a recurring character in the Fox animated television series The Simpsons voiced by Pamela Hayden and created by Matt Groening.
Thought 2	The paragraph does not tell who Milhouse is named after, maybe I can look up "named after".
Action 2	Lookup[named after]
Observation 2	(Result 1 / 1) Milhouse was named after U.S. president Richard Nixon, whose middle name was Milhous.
Thought 3	Milhouse was named after U.S. president Richard Nixon, so the answer is Richard Nixon.
Action 3	Finish[Richard Nixon]

What if we have 1000+ tools?

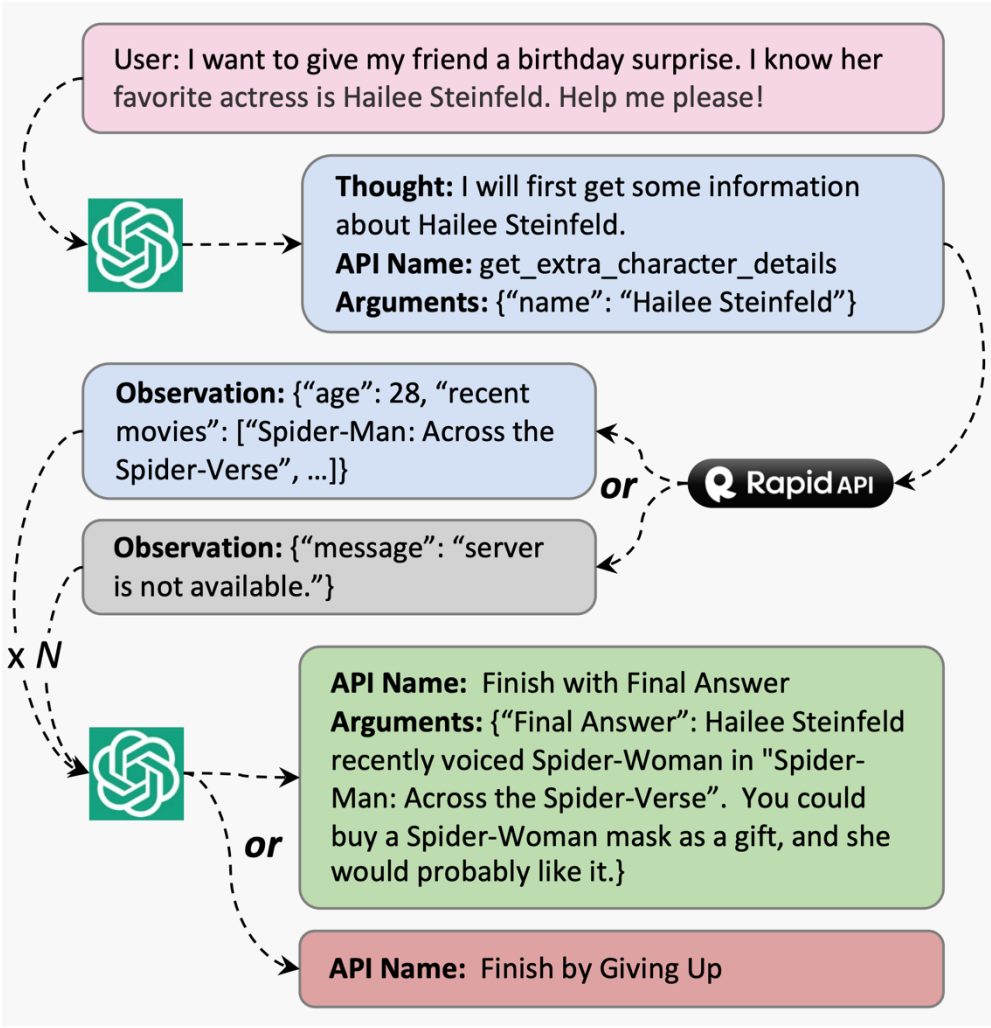
- Becomes a retrieval problem – what tool(s) to use given a prompt?
- Idea: Generate prompt+APIs data
- Finetune Encoder



ToolLLM (Qin 2023)

How to use multiple tools?

- Generate CoT+tool trajectories ReAct style
- Finetune LM



ToolLLM (Qin 2023)

Instruction: Find popular action movies available for streaming in the U.S.

APIs:

- GET_TopMovies(genre, country): list trending movies

- GET_StreamingAvailability(title, country): check where a movie is streaming

--- Trajectory ---

Thought 1: Get trending action movies.

Action 1: GET_TopMovies{"genre":"action","country":"us"}

Obs 1: ["John Wick 4","Mission Impossible 7","Fast X"]

Thought 2: Check streaming for "John Wick 4".

Action 2: GET_StreamingAvailability{"title":"John Wick 4","country":"us"}

Obs 2: {"Netflix":true,"Hulu":true}

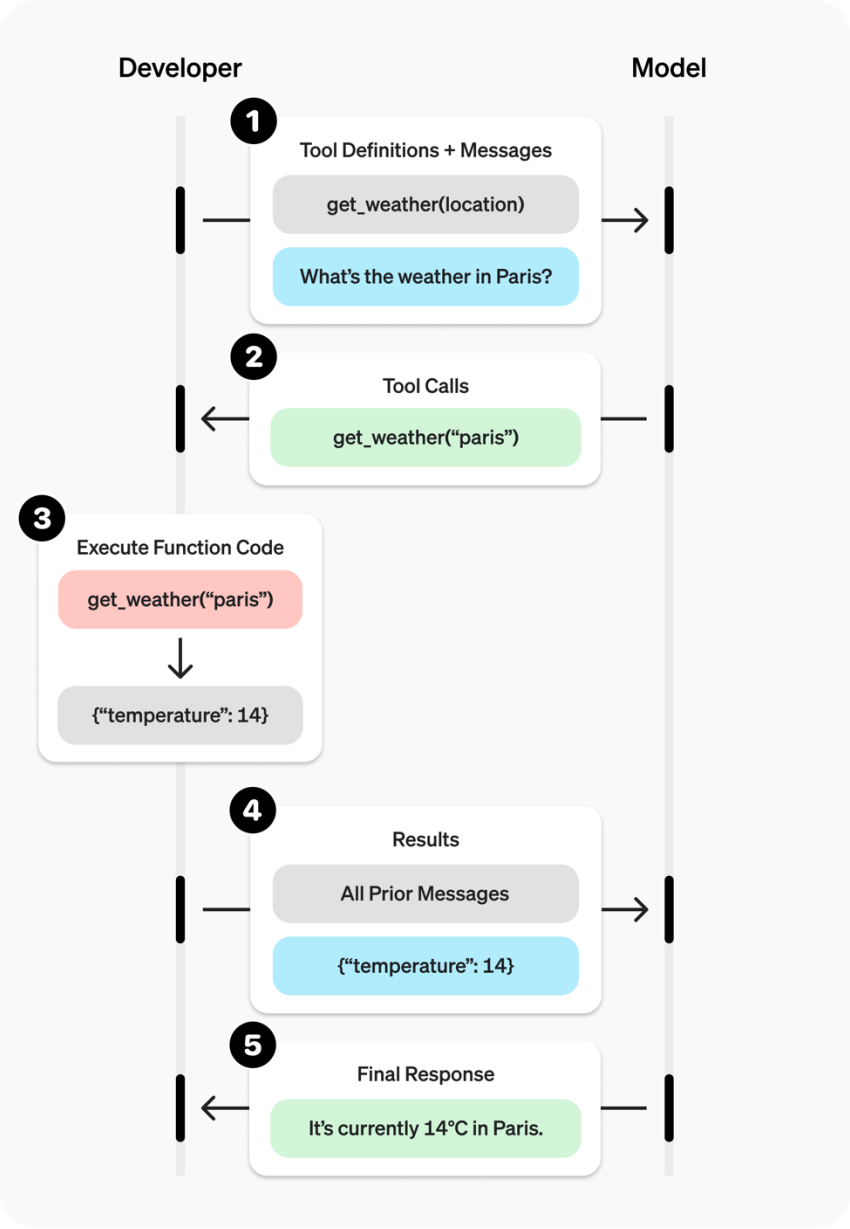
Thought 3: Summarize results.

Final Answer: John Wick 4 — Netflix, Hulu; Mission Impossible 7 — Paramount+; Fast X — Peacock.

Function Calling (OpenAI)

- At *inference*

```
{
    "type": "function",
    "name": "get_weather",
    "description": "Retrieves current weather for the given location.",
    "parameters": {
        "type": "object",
        "properties": {
            "location": {
                "type": "string",
                "description": "City and country e.g. Bogotá, Colombia"
            },
            "units": {
                "type": "string",
                "enum": ["celsius", "fahrenheit"],
                "description": "Units the temperature will be returned in."
            }
        },
        "required": ["location", "units"],
        "additionalProperties": false
    },
    "strict": true
}
```



Tool-Use Papers in 2025

- Iterative Tool Retrieval (ToolReAGT)
- Tool Retrieval is hard (ToolRet)
- RL for tools? (ToolRL)

Open Research Problems

- End-to-end RAG
  - Encoders -> Relevance
  - Generators -> Likelihood
- Retrieval + ChatBot
- Long-horizon memory
- Reward design for tool trace beyond final answers
- Tool retrieval at scale

Extra References

[1] https://medium.com/@mayssamayel4/building-a-rag-system-with-gpt-4-a-step-by-step-guide-291711342f0d

[2] https://phontron.com/class/anlp2024/lectures/