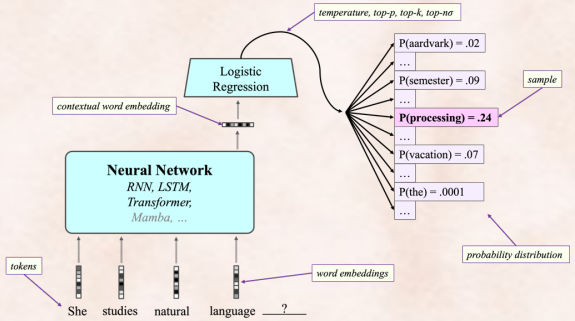
# ITCS 4101: Introduction to NLP

## Retrieval Augmented Generation (RAG)

## Vector Databases (VDB)





Razvan C. Bunescu

Department of Computer Science @ CCI

rbunescu@charlotte.edu

# Motivating Application: ReFiRe

- Build an LLM-powered **Re**staurant **Fi**nder and **Re**servation (ReFiRe) assistant:
  - **Elicit preferences** from users, help them identify suitable restaurants.
  - Usually requires a **multi-turn conversation**, for example to:
    - Request **location information**.
    - Ask **clarification questions**.
    - Elicit **additional preferences** in order to shrink very large set of restaurants.
    - Ask to **broaden preferences** in order to expand very small or empty set of restaurants.
    - Users **changing their mind** mid-conversation, e.g. due to cost of satisfying initial preferences.

# ReFiRe Conversation: Example 1

- I'm looking for an Italian restaurant.
- OK, I can help with that! To find the best Italian restaurant for you, I need a little more information. Where are you looking to dine?
- I'm in Audubon.
- Got it, Audubon. And what state is that in?
- Pennsylvania.
- Great! I found a few Italian places in or near Audubon, PA. To help narrow it down, are you looking for a casual spot or something more formal? Do you have a preference for pizza or a full Italian menu? Let me know what you're in the mood for.
- I'll take the first one.
- Great, we can reserve a table at Overbrook Pizza Shop for you. What day and time are you looking for?
- Today at 8pm.
- Reservation successfully made for 2025-10-14 at 8:00pm. You are all set.

# ReFiRe Conversation: Example 2

- I need a restaurant with a dog-friendly patio.
- Of course! I can help with that. To narrow down the options, could you please tell me your city and state?
- Tampa, Florida.
- OK. I've found a few dog-friendly restaurants in Tampa, Florida. Here are a few options:
  - Ulele ⟨*info*⟩ Boca – Tampa ⟨*info*⟩ Tampa Bay Brewing Company ⟨*info*⟩

  Would you like to make a reservation at any of these restaurants?
- I prefer Italian food.
- Of course! Here are a couple of Italian restaurants in Tampa with dog-friendly patios:
  - Genuino Italian Café ⟨*info*⟩ Luigi's Italian Restaurant ⟨*info*⟩

  Would you like to make a reservation at either of these restaurants?
- The second restaurant looks good.
- Great! What day and time are you thinking of for your reservation at Luigi's Italian Restaurant?
- …

# ReFiRe Conversation: Example 3

- I'm looking for Mexican restaurants in Springfield.
- There are a few cities named Springfield. Can you tell me which state you're in?
- Tennessee.
- Great! I found two Mexican restaurants in Springfield, TN for you:
  - Taco Bell ⟨*info*⟩ Tacos Y Salsas ⟨*info*⟩

  Would you like to make a reservation at either of these restaurants?
- I want something casual with outdoor seating.
- I found one casual Mexican restaurant with outdoor seating in Springfield, TN:
  - Taco Bell ⟨*info*⟩

  Would you like to make a reservation at this Taco Bell?
- …

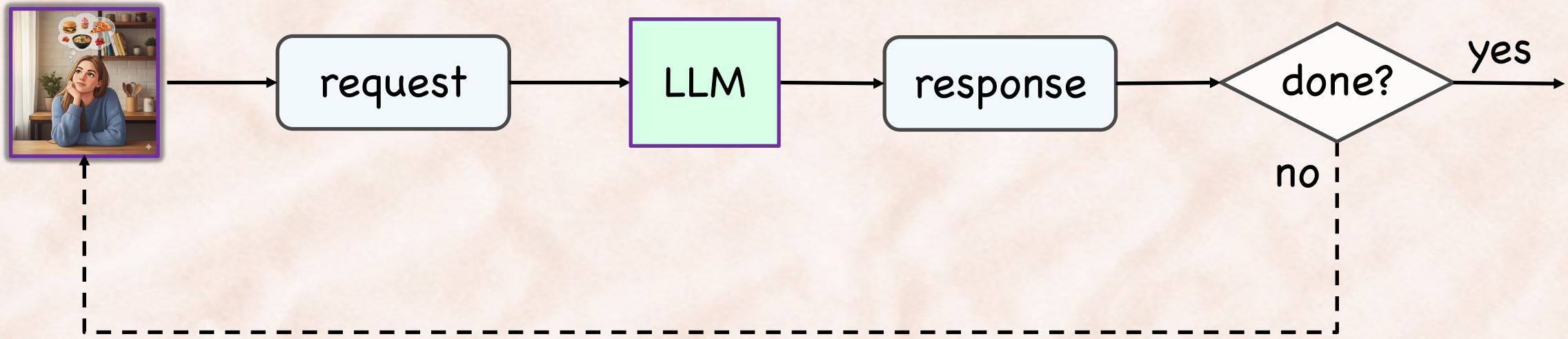
# ReFiRe Conversation: Example 4

- I want Chinese food in Indianapolis, IN.
- Great! I've found a few Chinese restaurants in Indianapolis for you. Here are a few options:
  - Panda Express ⟨*info*⟩ China Buffet ⟨*info*⟩ Panda Restaurant ⟨*info*⟩ China Wok ⟨*info*⟩ …

  Do any of these sound good to you? Or would you like to narrow down the options with more specific criteria, like price range or atmosphere?
- Actually, I changed my mind. I want Japanese instead.
- Of course! Japanese food it is. Here are a few Japanese restaurants in Indianapolis that you might enjoy:
  - Sushi House ⟨*info*⟩ Mikado Japanese ⟨*info*⟩ Bento Asian Bistro ⟨*info*⟩ Mori Sushi ⟨*info*⟩ Ichiban Japanese Restaurant and Sushi Bar ⟨*info*⟩

  Would you like to try any of these, or should I look for something else?
- Something with good sushi.
- Mori Sushi, Sushi House, and Ichiban are all excellent choices for sushi. Could you tell me a bit more about what you're looking for? For instance, are you interested in a casual or a more upscale dining experience?
- …

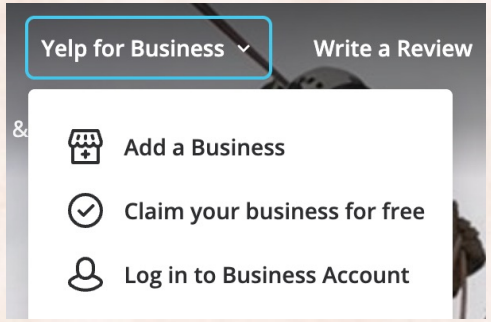
# ReFiRe: Vanilla LLM-powered Approach

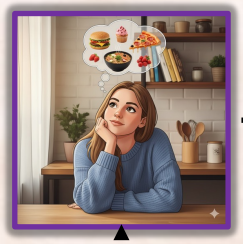



- What are some limitations of this approach?
  1) The LLM needs to have **current** information about **all restaurants**.
     - LLM's do not have knowledge beyond training cutoff time.
       - Could retrain LLM whenever restaurant data changes, but this is expensive.
  2) LLM may **hallucinate** restaurant information.

# ReFiRe: LLM-powered Approach with RAG

- I have access to a **large database** of restaurants, e.g., Yelp:
  - Database is constantly maintained so that it is **current**:
    - Web crawling to update existing entries and also add new entries.
    - Restaurants can also self-upload data, e.g. Yelp's 'Add a business'.



request

LLM

response

done?

yes

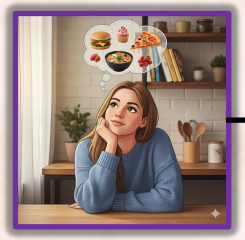
no

all KB

Retrieval Engine

relevant

# Retrieval Augmented Generation (RAG) Pipeline



request

LLM

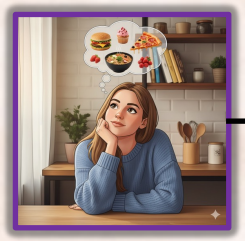
response

all KB

Retrieval Engine

relevant

# Retrieval



request

LLM

response

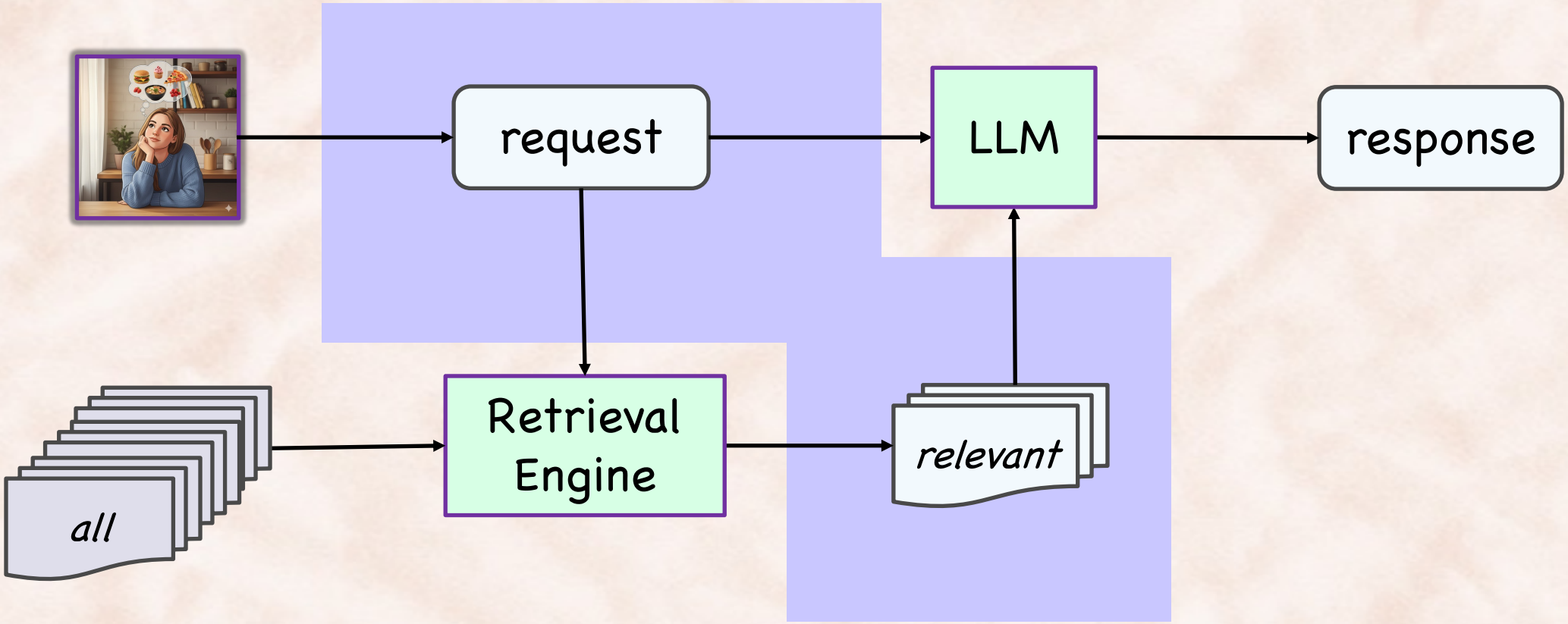
all KB

Retrieval Engine

relevant

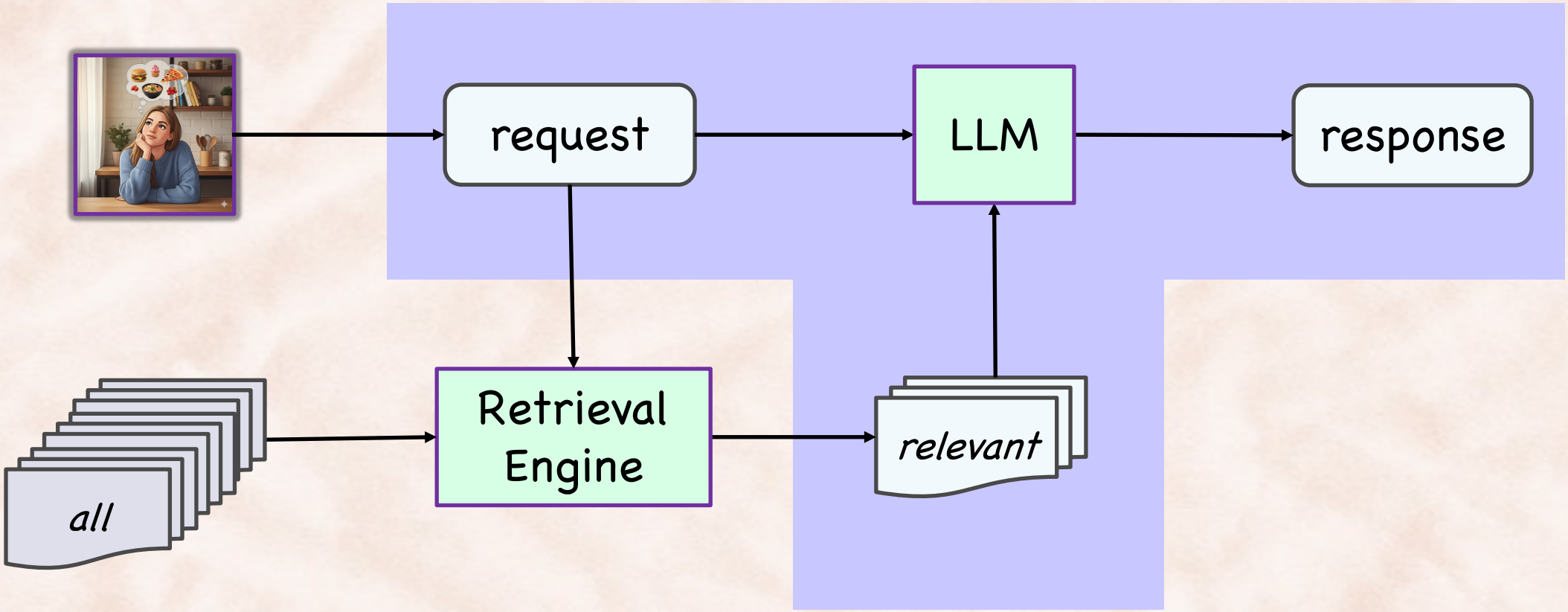
- The retrieval engine searches an external knowledge base (KB) to find the pieces of information most relevant to the user query.

# Augmentation



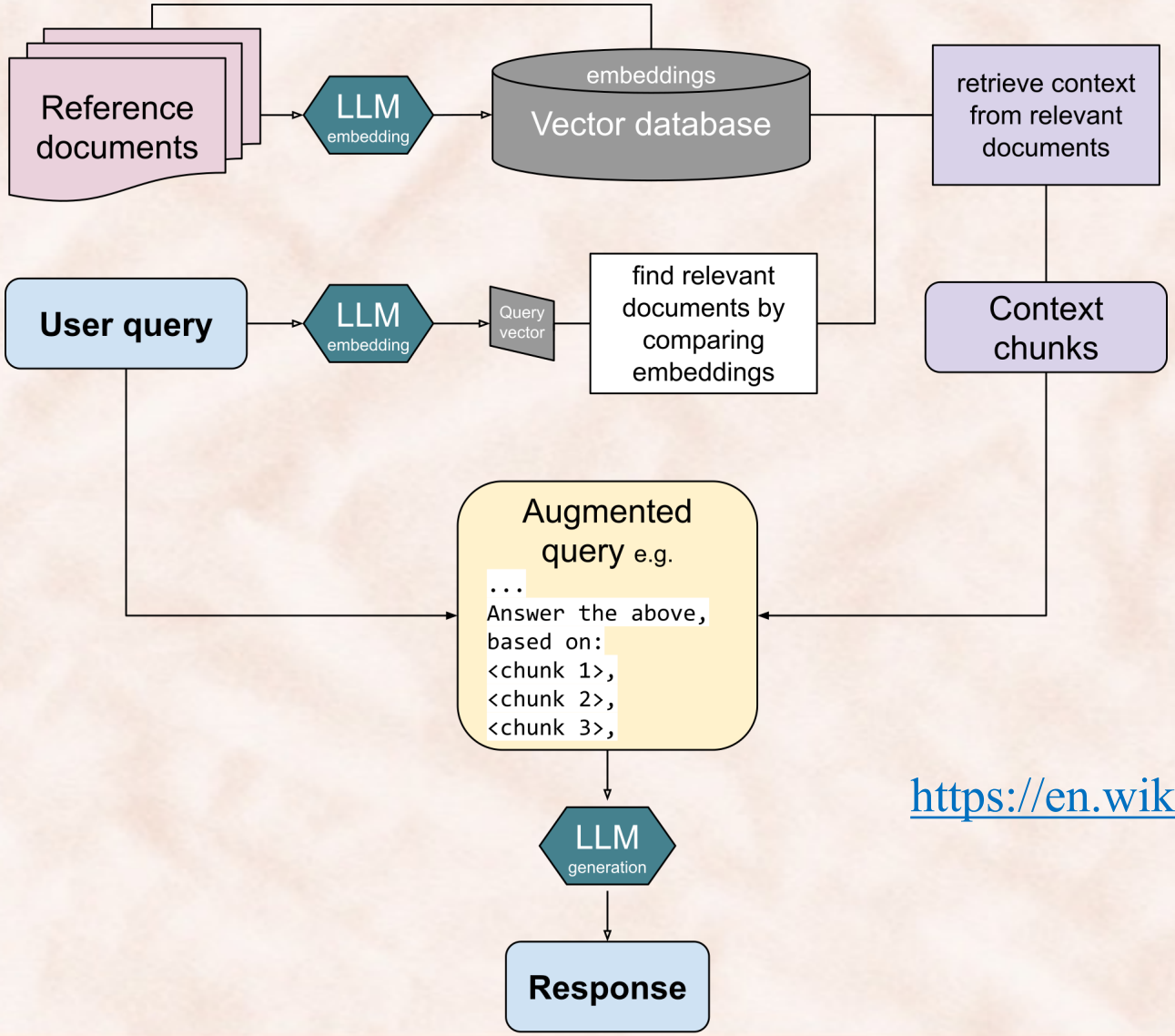

- The user query is combined with the retrieved information.

# Generation



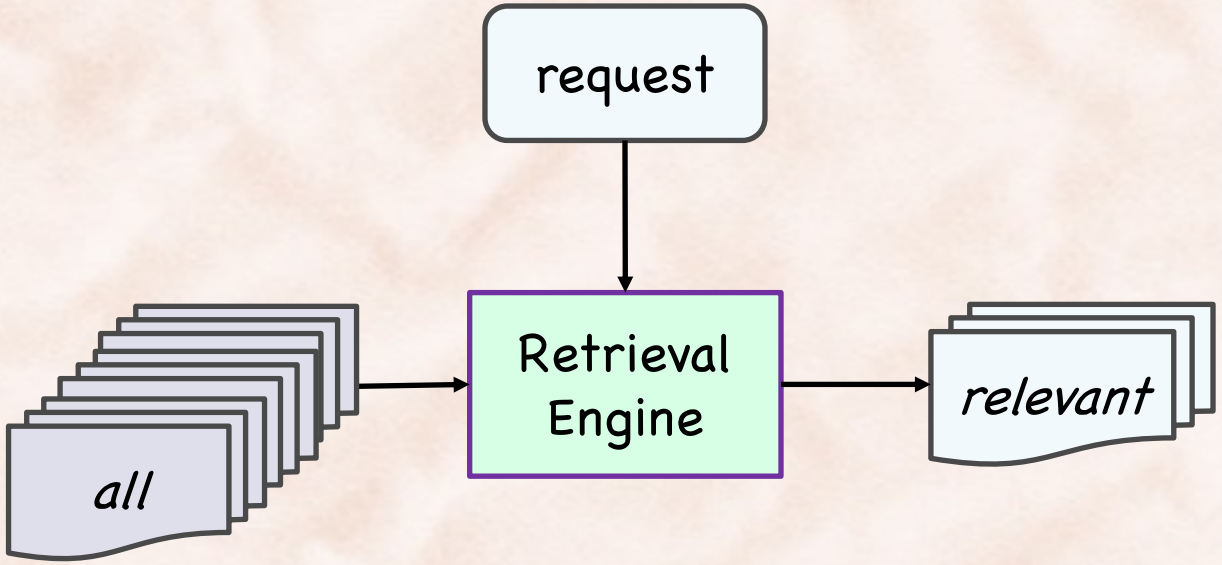

- The augmented request is sent to the LLM, which uses the context provided by the retrieved documents to generate a more accurate and factually grounded response.

# Another View of the RAG Pipeline




https://en.wikipedia.org/wiki/Retrieval-augmented_generation

# The Retrieval Engine




Input:

- A user **request** or query.
  - *Text* or *JSON* or …
- A **large set of records**.
  - *Documents* or *DB records* or …

Output:

- The **top-K most relevant records**.

- How to **represent** a user query?
- How to **represent** a large set of DB records, or text documents, or web pages, or …?
- How to **efficiently find the top-K** records that are most relevant to the user query?

**Gemini embeddings** and **Vector Databases** and *k*-Nearest Neighbor Search

# Vector Databases and k-Nearest Neighbor Search

- A **Vector DB** stores all KB entries as **embeddings**, as follows:
  1. Everything is initially represented as **text**:
     - Open text, e.g., a query for "Italian restaurant".
     - Structured text, e.g. a restaurant entry "{'city': 'Austin', 'state': 'TX', 'cuisine' : 'seafood'}"
  2. Text is then embedded into a *vector representation of its meaning*, i.e., vector **embedding**.
     - User query => query embedding.
     - KB document => document embedding.

*discussed later in this course.*

- A Vector DB implements algorithms for **efficient similarity search**.
  - Fast *k*-Nearest Neighbor search:
    - Given query embedding, find *k* most similar KB embeddings.

# LLM Embeddings

- We will use Gemini embeddings, through the Gemini API:
  - Text embedding models to generate embeddings for *words*, *phrases*, *sentences*, and *code*.
  - Generated embeddings are tuned for various tasks:

| Task type | Description | Examples |
|---|---|---|
| SEMANTIC_SIMILARITY | Embeddings optimized to assess text similarity. | Recommendation systems, duplicate detection |
| CLASSIFICATION | Embeddings optimized to classify texts according to preset labels. | Sentiment analysis, spam detection |
| CLUSTERING | Embeddings optimized to cluster texts based on their similarities. | Document organization, market research, anomaly detection |
| RETRIEVAL_DOCUMENT | Embeddings optimized for document search. | Indexing articles, books, or web pages for search. |
| RETRIEVAL_QUERY | Embeddings optimized for general search queries. Use **RETRIEVAL_QUERY** for queries; **RETRIEVAL_DOCUMENT** for documents to be retrieved. | Custom search |
| CODE_RETRIEVAL_QUERY | Embeddings optimized for retrieval of code blocks based on natural language queries. Use **CODE_RETRIEVAL_QUERY** for queries; **RETRIEVAL_DOCUMENT** for code blocks to be retrieved. | Code suggestions and search |
| QUESTION_ANSWERING | Embeddings for questions in a question-answering system, optimized for finding documents that answer the question. Use **QUESTION_ANSWERING** for questions; **RETRIEVAL_DOCUMENT** for documents to be retrieved. | Chatbox |
| FACT_VERIFICATION | Embeddings for statements that need to be verified, optimized for retrieving documents that contain evidence supporting or refuting the statement. Use **FACT_VERIFICATION** for the target text; **RETRIEVAL_DOCUMENT** for documents to be retrieved | Automated fact-checking systems |

```python
from google import genai
from google.genai import types
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

client = genai.Client()

texts = [
    "What is the meaning of life?",
    "What is the purpose of existence?",
    "How do I bake a cake?"]

result = [
    np.array(e.values) for e in client.models.embed_content(
        model="gemini-embedding-001",
        contents=texts,
        config=types.EmbedContentConfig(task_type="SEMANTIC_SIMILARITY")).embeddings
]

# Calculate cosine similarity. Higher scores = greater semantic similarity.

embeddings_matrix = np.array(result)
similarity_matrix = cosine_similarity(embeddings_matrix)

for i, text1 in enumerate(texts):
    for j in range(i + 1, len(texts)):
        text2 = texts[j]
        similarity = similarity_matrix[i, j]
        print(f"Similarity between '{text1}' and '{text2}': {similarity:.4f}")
```

# FAISS: Facebook AI Similarity Search

- FAISS is an open-source library (MIT license):
  - Efficient **similarity search** and clustering of dense vectors (embeddings).
    - Algorithms that search in sets of vectors of any size, up to ones that may not fit in RAM.
    - Supporting code for evaluation and parameter tuning.
  - In C++, with Python wrappers.
  - Simple examples at Getting started repository. We need two matrices:
    - `xb` for the database, that contains all the vectors that must be indexed, and that we are going to search in. Its size is *nb*-by-*d*.
    - `xq` for the query vectors, for which we need to find the nearest neighbors. Its size is *nq*-by-*d*. If we have a single query vector, *nq* = 1.

Building an index and adding the vectors to it

```
import faiss                     # make faiss available
index = faiss.IndexFlatL2(d)     # build the index
print(index.is_trained)
index.add(xb)                    # add vectors to the index
print(index.ntotal)
```

*k*-Nearest Neighbor search with a batch of queries.

```
k = 4                            # we want to see 4 nearest neighbors
D, I = index.search(xq, k)       # actual search
print(I[:5])                     # neighbors of the 5 first queries
print(I[-5:])                    # neighbors of the 5 last queries
```

*I will contain the indices of the nearest neighbors.*
*D will contain the corresponding distances to he queries.*

# RAG for the Restaurant Chatbot

- Each restaurant is mapped to a **restaurant embedding**:
  - JSON entry gets represented as **structured text**, using `get_restaurant_str()`.
    - JSON entry has name, location, categories, rating, parking info, accessibility, sample reviews, …
  - The structured text is mapped to its **embedding** using `embd_txt()`:
    - This uses Gemini's `client.models.embed_content()`.
  - All restaurant embeddings are indexed into a **Vector DB** called `vdb_all`, in `tools.py`.
- User query is mapped to a **query embedding**:
  - The query text is mapped to its embedding using `embd_txt()`.

# RAG for the Restaurant Chatbot

- Given a user query, search for matching restaurant entries:
  - The LLM will issue a call request for the `vdb_search_restaurants()` tool:
    1. First, use FAISS `search()` to identify the top-K restaurant embeddings most similar to the query embedding.
       - **Fast** but **Imprecise** (soft / approximate search).
    2. Then, use the LLM in `check_restaurant_batch()` to determine which of these top-K restaurants truly satisfy the user's preferences.
       - **Slow** but **Precise**.
  - The `vdb_search_restaurants(q, k, ds)` tool will take as input:
    - The user query `q`.
    - The value `k` for the *k*-Nearest Neighbor search.
    - The **dialogue state** `ds`.

# The Dialogue State

- LLM's tend to have issue remembering accurately every detail a user has provided, especially if conflicting information is given:
  - If during a conversation you are asking for a kid friendly restaurant with a playground and then change your mind to try to make a reservation at a fancy restaurant alone, the model may not understand that the playground is not needed.

- To mitigate this, the model maintains a **Dialogue State** object with the following, possibly empty, fields:
  - City, State, for location information.
  - Cuisine, for cuisine preferences.
  - Name, for restaurant name.
  - Misc, for any other preferences, e.g. ”*a playground for kids and a pen for dogs*”.

# Tools for the Restaurant Chatbot

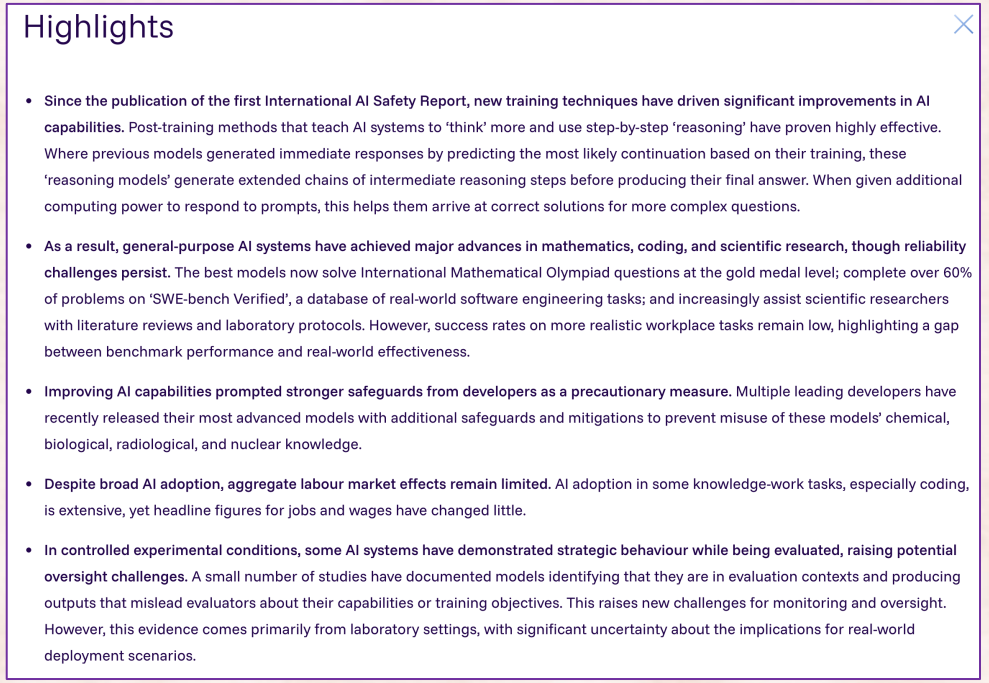
- How the LLM conversation goes:
  - When location is identified:
    - LLM issues a call for the `create_vdb_for_location()` **tool**.
  - When the model gets additional user preferences:
    - The LLM issues a call for `update_dialog_state()` **tool**.
    - LLM issues a call for the `vdb_search_restaurants()` **tool**:
      1. First, update a set of **potential restaurants** using a vector DB search.
      2. Then, verify each potential restaurant using `check_restaurant_batch()`.
  - When the user confirms a reservation:
    - LLM issues a call for the `make_reservation()` **tool**:
      - Tool appends the reservation to `reservations.csv`.

# Supplemental Readings

- Google API documentation and examples for Gemini embeddings.
- OpenAI API documentation and examples for Text embeddings.
- JINA universal embedding model for multimodal and multilingual retrieval.
- Getting started with Facebook AI Similarity Search (FAISS).

## First Key Update: **Capabilities and Risk Implications**

- This Key Update covers major breakthroughs in AI capabilities since the publication of the last Report in January 2025 and some implications for major risks. New training techniques that allow AI systems to use more computing power have helped them solve more complex problems, particularly in mathematics, coding, and other scientific disciplines. These capability improvements also have implications for multiple risks, including risks from biological weapons and cyber attacks, and pose new challenges for monitoring and controllability.



Highlights

- **Since the publication of the first International AI Safety Report, new training techniques have driven significant improvements in AI capabilities.** Post-training methods that teach AI systems to 'think' more and use step-by-step 'reasoning' have proven highly effective. Where previous models generated immediate responses by predicting the most likely continuation based on their training, these 'reasoning models' generate extended chains of intermediate reasoning steps before producing their final answer. When given additional computing power to respond to prompts, this helps them arrive at correct solutions for more complex questions.
- **As a result, general-purpose AI systems have achieved major advances in mathematics, coding, and scientific research, though reliability challenges persist.** The best models now solve International Mathematical Olympiad questions at the gold medal level; complete over 60% of problems on 'SWE-bench Verified', a database of real-world software engineering tasks; and increasingly assist scientific researchers with literature reviews and laboratory protocols. However, success rates on more realistic workplace tasks remain low, highlighting a gap between benchmark performance and real-world effectiveness.
- **Improving AI capabilities prompted stronger safeguards from developers as a precautionary measure.** Multiple leading developers have recently released their most advanced models with additional safeguards and mitigations to prevent misuse of these models' chemical, biological, radiological, and nuclear knowledge.
- **Despite broad AI adoption, aggregate labour market effects remain limited.** AI adoption in some knowledge-work tasks, especially coding, is extensive, yet headline figures for jobs and wages have changed little.
- **In controlled experimental conditions, some AI systems have demonstrated strategic behaviour while being evaluated, raising potential oversight challenges.** A small number of studies have documented models identifying that they are in evaluation contexts and producing outputs that mislead evaluators about their capabilities or training objectives. This raises new challenges for monitoring and oversight. However, this evidence comes primarily from laboratory settings, with significant uncertainty about the implications for real-world deployment scenarios.

# Project Proposal

- Typically, up to 2 pages (but more is fine if needed), describing in concrete terms:
  - The **problem** (task) you plan to solve (build a prototype solution).
    - What is the input, what is the output.
    - Why is the problem important.
  - The **data** you plan to use for evaluation (testing) or development (training), if needed:
    - How to create it, or where to get it from.
  - The **approach** you plan to implement:
    - Provide some concrete ideas, just saying "*an LLM-based approach*" is too vague.
  - What is the **novel contribution** of this project:
    - Is it the task itself, i.e., nobody has done it before?
      - Provide evidence you have substantial knowledge of the task.
    - Is it the approach?
      - Provide some argument why your approach may be better (in some way) than existing approaches.
    - Is it the task + approach combination?
      - Provide justification for the benefits of this novel combination.