



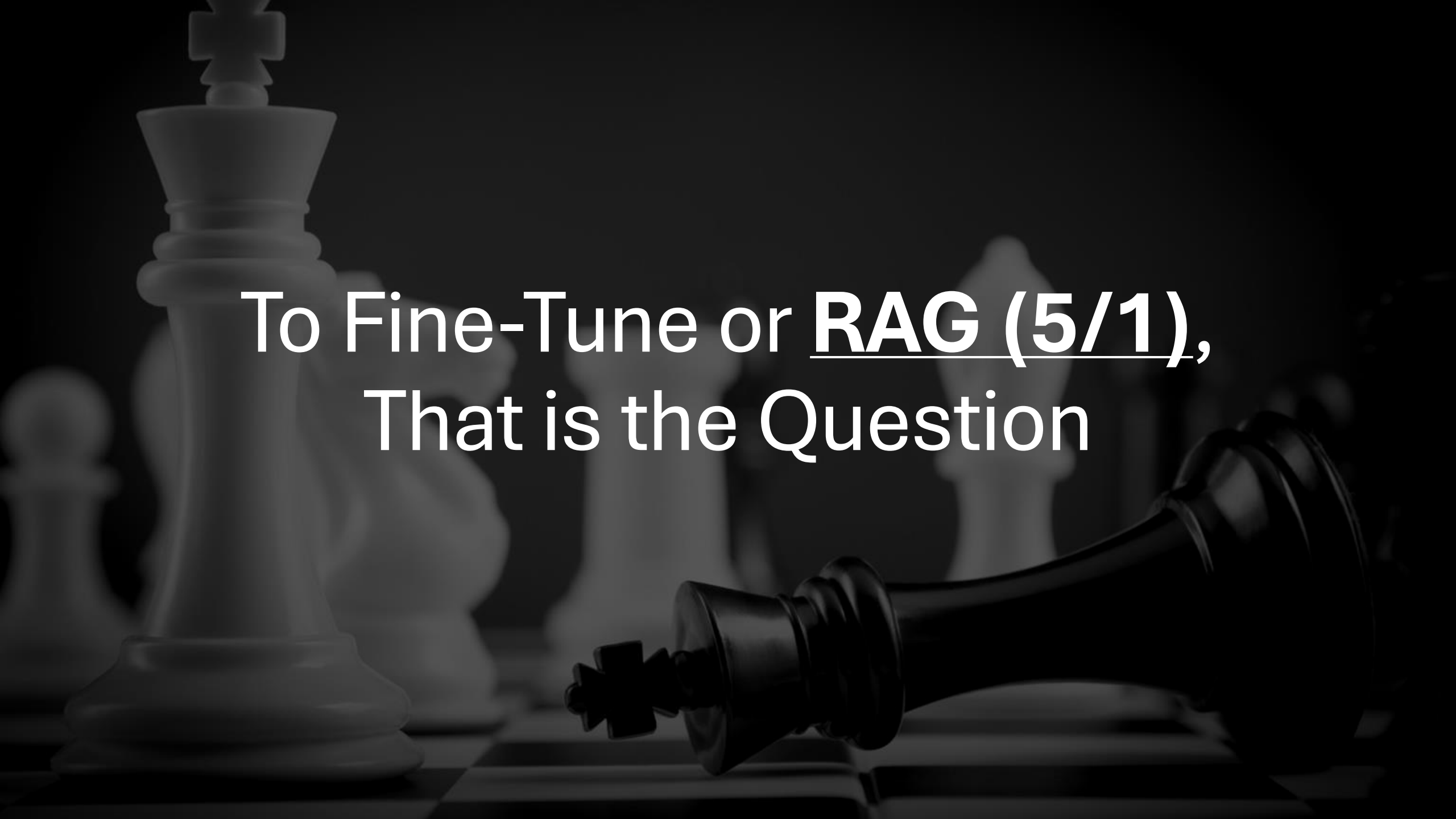
# Topics in LLMs

**RAG and Fine-Tuning**

*Joel Kowalewski, PhD*





A grayscale photograph of a chessboard. In the foreground on the left, a white king piece stands upright. To its right, a black king piece lies on its side, having been captured. The background shows other chess pieces out of focus.

To Fine-Tune or RAG (5/1),  
That is the Question

# Case Study

```
def create_llm_prompt(system_prompt, rag_context, user_history, user_query):
    # Format the user's chat history as a string
    history_text = "\n".join(f"Chat {idx + 1}: {entry}" for idx, entry in enumerate(user_history))

    # Assemble the complete prompt using formatted strings
    prompt = (f"System Prompt: {system_prompt}\n\n"
              f"RAG Retrieved Context: {rag_context}\n\n"
              f"User Chat History:\n{history_text}\n\n"
              f"User's Current Query: {user_query}\n")

    return prompt

# Example usage
system_prompt = "Please assist the user with their medical questions."
rag_context = "Disease X is a rare genetic disorder characterized by the absence of symptom Y"
user_history = [
    """{"role": "User", "message": "What is disease Z?"},
    {"role": "Assistant", "message": "Disease Z is a rare genetic disorder that affects 1/1,000,000 people."}
    """
]
user_query = "What is disease X?"

# Generate the LLM prompt
prompt = create_llm_prompt(system_prompt, rag_context, user_history, user_query)
print(prompt)
```

System Prompt: Please assist the user with their medical questions.

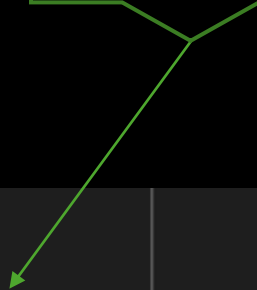
RAG Retrieved Context: Disease X is a rare genetic disorder characterized by the absence of symptom Y

User Chat History:

Chat 1: {"role": "User", "message": "What is disease Z?"},  
{"role": "Assistant", "message": "Disease Z is a rare genetic disorder that affects 1/1,000,000 people."}

User's Current Query: What is disease X?

How did we  
retrieve this  
information?



```
# Example usage
system_prompt = "Please assist the user with their medical questions."
rag_context = "Disease X is a rare genetic disorder characterized by the absence of symptom Y"
user_history = [
    """{"role": "User", "message": "What is disease Z?"},
    {"role": "Assistant", "message": "Disease Z is a rare genetic disorder that affects 1/1,000,000 people."}
    """
]
user_query = "What is disease X?"
```





# RAG

---

- RAG involves the use of a vector database (indexed embeddings) to contextualize user prompts.
- A RAG retriever is used to compare the vectors (the normalized dot product) of the documents in the database with the user's query, returning the top k documents.
- This database may be readily updated with live information, making this amended prompt more relevant than the corpus used to pretrain the LLM .

# What are Embeddings?

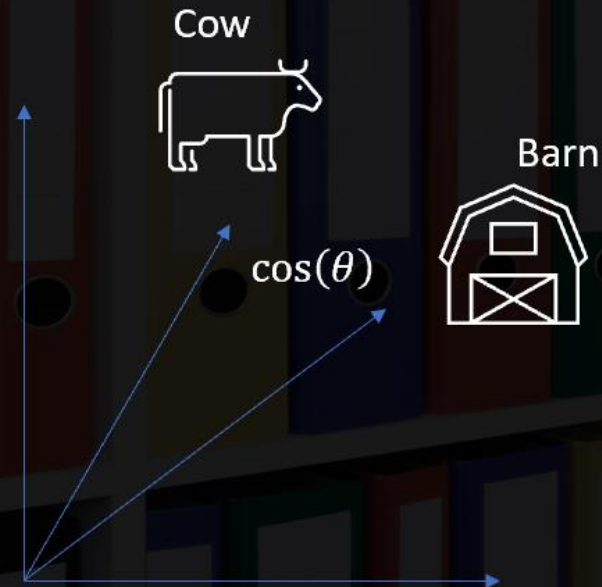
```
array([ 0.09253988,  0.01249253,  0.07163866, -0.2743904 , -0.11499565,  
       -0.08302431, -0.11840086,  0.16413152,  0.01885854, -0.02985609,  
       0.12597775, -0.00793147,  0.06259963, -0.15034045,  0.18575828,  
       -0.06957697,  0.08653288,  0.00627072, -0.34941605,  0.11818368,  
       0.13578415, -0.11451971, -0.03621563,  0.149329 ,  0.03569125,  
       -0.09157975,  0.10489449,  0.28316483, -0.00125286,  0.0339428 ,  
       0.02545493,  0.08704362,  0.12570098, -0.04304032,  0.10261655,  
       -0.17717567,  0.07606252, -0.0185764 , -0.07972988,  0.00443853,  
       0.1494237 ,  0.05579368, -0.05541351,  0.06199389,  0.20522949,  
       -0.22456484,  0.01813756, -0.02627641, -0.01573355,  0.00099138,  
       0.06590673, -0.2637863 , -0.01715699,  0.27637798,  0.04561216,  
       -0.02390107,  0.02922816, -0.16145538,  0.1465739 , -0.11907259,  
       0.00830688, -0.09886483, -0.06092348,  0.17352898], dtype=float32)
```

Embeddings are learned, numerical representations of text.

They incorporate context and semantics so that the numerical representation mirrors human language.

Importantly, this numerical representations therefore allows us to turn language into mathematics (*see image, right*)

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$



**Limitations:** Same word, same embedding.

Next, we'll discuss how modern neural network architectures address this



# What are Vector Databases Then?

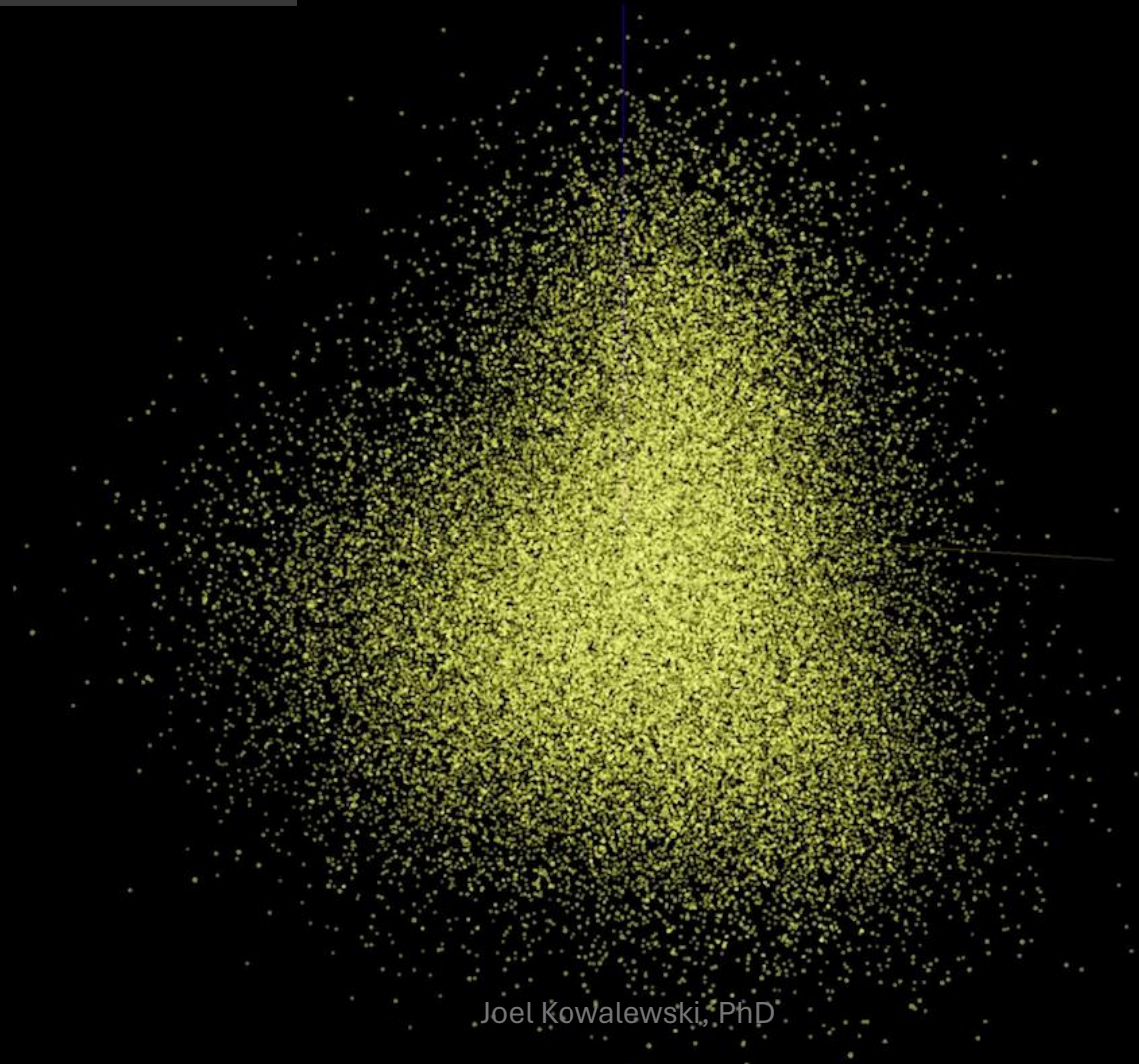
---





```
array([ 0.09253988,  0.01249253,  0.07163866, -0.2743904 , -0.11499565,  
       -0.08302431, -0.11840086,  0.16413152,  0.01885854, -0.02985609,  
        0.12597775, -0.00793147,  0.06259963, -0.15034045,  0.18575828,  
       -0.06957697,  0.08653288,  0.00627072, -0.34941605,  0.11818368,  
        0.13578415, -0.11451971, -0.03621563,  0.149329  ,  0.03569125,  
       -0.09157975,  0.10489449,  0.28316483, -0.00125286,  0.0339428 ,  
        0.02545493,  0.08704362,  0.12570098, -0.04304032,  0.10261655,  
       -0.17717567,  0.07606252, -0.0185764 , -0.07972988,  0.00443853,  
        0.1494237 ,  0.05579368, -0.05541351,  0.06199389,  0.20522949,  
       -0.22456484,  0.01813756, -0.02627641, -0.01573355,  0.00099138,  
        0.06590673, -0.2637863 , -0.01715699,  0.27637798,  0.04561216,  
       -0.02390107,  0.02922816, -0.16145538,  0.1465739 , -0.11907259,  
        0.00830688, -0.09886483, -0.06092348,  0.17352898], dtype=float32)
```

# A library of embeddings



```

import faiss
import numpy as np
from transformers import AutoTokenizer, AutoModel

# Initialize tokenizer and model
tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')
model = AutoModel.from_pretrained('bert-base-uncased')

# Create a toy document about "Disease X"
documents = ["Disease X is a rare genetic disorder characterized by the absence of symptom Y. It affects 1 in every 100,000 people.",
             "The hippocampus is an area of the brain that is needed to form new memories." ]

# Convert documents to embeddings
embeddings = []
for doc in documents:
    inputs = tokenizer(doc, return_tensors="pt", padding=True, truncation=True, max_length=512)
    outputs = model(**inputs)
    # Use mean pooling to get a single vector representation for each document
    embeddings.append(outputs.last_hidden_state.mean(dim=1).detach().numpy())

# Create a FAISS index
dimension = embeddings[0].shape[1] # Dimension of embeddings
index = faiss.IndexFlatL2(dimension) # Flat index using L2 distance
# index = faiss.IndexFlatIP(dimension) # This index computes the inner product

# Add embeddings to index
for emb in embeddings:
    index.add(emb)

```

RAG options include FAISS (Facebook AI Similarity Search), Annoy, or Elasticsearch or something that can support efficient vector search.



We Need to Define a Retriever  
for our RAG database

```
def retrieve_document(query):  
    # Tokenize and embed the query  
    query_inputs = tokenizer(query, return_tensors="pt", padding=True, truncation=True, max_length=512)  
    query_outputs = model(**query_inputs)  
    query_embedding = query_outputs.last_hidden_state.mean(dim=1).detach().numpy()  
  
    # Search the FAISS index  
    D, I = index.search(query_embedding, k=1) # Search for the top 1 nearest document  
    return documents[I[0][0]]
```

## Example RAG Usage

```
query = "What is Disease X?"  
retrieved_doc = retrieve_document(query)  
print(f"Retrieved document: {retrieved_doc}")
```

Retrieved document: Disease X is a rare genetic disorder characterized by the absence of symptom Y. It affects 1 in every 100,000 people.

```
query = "How are new memories formed in the brain?"  
retrieved_doc = retrieve_document(query)  
print(f"Retrieved document: {retrieved_doc}")
```

Retrieved document: The hippocampus is an area of the brain that is needed to form new memories.



```
# Example usage
system_prompt = "Please assist the user with their medical questions."
rag_context = "Disease X is a rare genetic disorder characterized by the absence of symptom Y"
```

How did we  
retrieve this  
information?

## SLIDE 10 (Created Embeddings → Built Database)

ANSWER

```
import faiss
import numpy as np
from transformers import AutoTokenizer, AutoModel

# Initialize tokenizer and model
tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')
model = AutoModel.from_pretrained('bert-base-uncased')

# Create a toy document about "Disease X"
documents = ["Disease X is a rare genetic disorder characterized by the absence of symptom Y. It affects 1 in every 100,000 people.",
             "The hippocampus is an area of the brain that is needed to form new memories." ]

# Convert documents to embeddings
embeddings = []
for doc in documents:
    inputs = tokenizer(doc, return_tensors="pt", padding=True, truncation=True, max_length=512)
    outputs = model(**inputs)
    # Use mean pooling to get a single vector representation for each document
    embeddings.append(outputs.last_hidden_state.mean(dim=1).detach().numpy())

# Create a FAISS index
dimension = embeddings[0].shape[1] # Dimension of embeddings
index = faiss.IndexFlatL2(dimension) # Flat index using L2 distance
# index = faiss.IndexFlatIP(dimension) # This index computes the inner product

# Add embeddings to index
for emb in embeddings:
    index.add(emb)
```

```
# Example usage
system_prompt = "Please assist the user with their medical questions."
rag_context = "Disease X is a rare genetic disorder characterized by the absence of symptom Y"
```

How did we  
retrieve this  
information?

## SLIDE 11 ( Then we Defined A Retrieval Strategy)

We Need to Define a Retriever  
for our RAG database

```
def retrieve_document(query):
    # Tokenize and embed the query
    query_inputs = tokenizer(query, return_tensors="pt", padding=True, truncation=True, max_length=512)
    query_outputs = model(**query_inputs)
    query_embedding = query_outputs.last_hidden_state.mean(dim=1).detach().numpy()

    # Search the FAISS index
    D, I = index.search(query_embedding, k=1) # Search for the top 1 nearest document
    return documents[I[0][0]]
```

**ANSWER**



## Summary Table: RAG (5/1/24) versus Fine-Tuning (Next time)

Feature	Fine-tuning LLM	RAG
Customization	<b>Pros:</b> Highly customizable to specific tasks and datasets. <b>Cons:</b> Requires extensive computational resources for training.	<b>Pros:</b> Adapts dynamically by retrieving from a knowledge base. <b>Cons:</b> Retrieval errors can propagate to generation errors.
Data Dependency	<b>Pros:</b> Performs well with large, diverse datasets. <b>Cons:</b> Performance depends heavily on the quality of training data.	<b>Pros:</b> Less dependent on specific training datasets. <b>Cons:</b> Reliant on the quality and relevance of external data.
Computational Cost	<b>Pros:</b> High upfront computational cost for training, but lower ongoing costs. <b>Cons:</b> High initial investment in resources.	<b>Pros:</b> Lower training costs since it uses pre-trained models. <b>Cons:</b> Higher query-time costs due to retrieval process.
Adaptability	<b>Pros:</b> Highly tailored once trained. <b>Cons:</b> Not easy to adapt to new data quickly; retraining required.	<b>Pros:</b> Can adapt to new information if the knowledge source is updated. <b>Cons:</b> May struggle with very recent or niche topics not in the database.
Inference Time	<b>Pros:</b> Faster performance at inference time. <b>Cons:</b> Eventually, retraining is needed since new data/information is generated at a high rate. This issue leads back to the computational cost dilemma; fine-tuning may require repeated investments in hardware or cloud compute services.	<b>Pros:</b> Can incorporate real-time data if connected to a live database. <b>Cons:</b> Slower response times due to retrieval step.
Scalability	<b>Pros:</b> Scales well once trained, with consistent performance. <b>Cons:</b> Limited by the cost and feasibility of retraining.	<b>Pros:</b> Can scale by expanding the external data source. <b>Cons:</b> Affected by the efficiency of the retrieval system.