From Concept to Code: Understanding and Building Retrieval-Augmented Generation (RAG) Systems

Powering LLMs with Tailored Knowledge

Shubham Chatterjee

Missouri University of Science and Technology, USA

April 23, 2025

Before we start.... 🚀

Setup Steps:

1. Visit console.mistral.ai to create a free account.
2. Generate your API key in the dashboard.
3. Install the Python client package.
4. Choose your model:
   - mistral-tiny
   - mistral-small
   - mistral-medium
   - mistral-large
5. Follow the tutorial in our Colab Notebook or GitHub repo for a hands-on guide.

Tutorial Resources:

- Open the tutorial in Google Colab
- View the tutorial code on GitHub

These resources walk you through setup and basic usage of the Mistral AI Python client, ideal for building RAG applications.

Free Tier Access

Start with 100 free credits! Perfect for prototyping your RAG application 💡

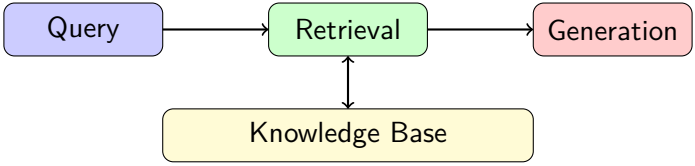
Retrieval-Augmented Generation

- RAG supercharges LLMs by fetching relevant information before generating answers 🚀
- Think of it as giving your LLM a personalized research assistant!
- Reduces hallucinations by grounding responses in factual information 📈
- No need to retrain models - just connect them to your knowledge sources 🔌

Question

Have you ever caught an LLM making up facts? RAG helps solve that problem!

The Magic Behind RAG ✨



1. User asks a question 🙋
2. System finds relevant information in a knowledge base 🔎
3. LLM crafts an answer using that specific information 💬

Analogy

It's like the difference between taking a test with or without your notes!

What We'll Learn Today 📚

By the end of this tutorial, you'll be able to

- Build a complete RAG pipeline from scratch 🛠️
- Customize RAG for different domains (science, tech support, education) 🔧
- Improve retrieval quality with bi-encoders and cross-encoders 📈
- Craft domain-specific prompts for better answers 📝
- Evaluate your RAG system with the right metrics ⚖️

Ready?

Ready to supercharge your LLMs with domain knowledge? Let's dive in! 🏊

Why We Need RAG: The Hallucination Problem 👻

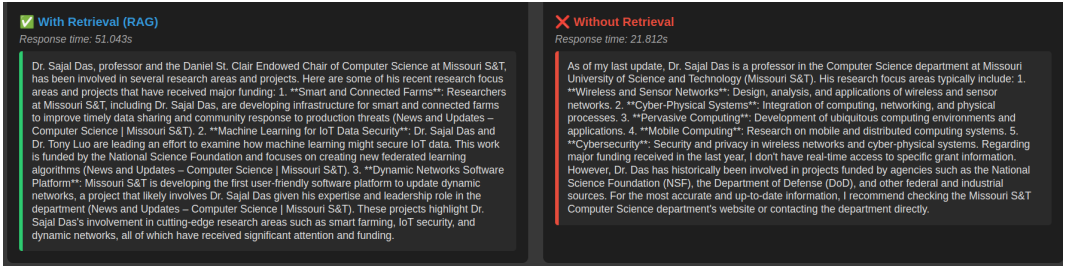
- LLMs have a tendency to 'hallucinate" - generate text that sounds plausible but is factually incorrect 🥴
- This happens because they:
  - Were trained on outdated information 📅
  - Lack access to specialized or recent information 🔎
  - Sometimes "make up" details to complete patterns 🧩
- For applications requiring factual accuracy, this is a major problem! ⚠️
- RAG addresses this by grounding LLM outputs in trustworthy documents 📚

Real Example

Let's look at a real example of RAG vs. no retrieval! 👀

RAG vs. No Retrieval: A Real Example 🔎

What are the current research focus areas of Sajal Das of Missouri S&T's CS department, and which projects have received major funding in the last year?



Without Retrieval:

- Generic information 📖
- Based on outdated training data 📅
- Hedging with "typically include" ⚠️
- Ends with a disclaimer referring elsewhere 🤷

With Retrieval:

- Specific, current projects 📊
- Names of collaborators 👤👤
- Mentions of actual funding sources 💰
- Cites specific information sources 📝

The Real Cost of Hallucinations 💸

- **Business Impact:**
  - Incorrect product information leads to customer mistrust 😡
  - Wrong technical advice increases support costs ☎️
  - Legal liability from providing false information ⚖️
- **Critical Domains:**
  - **Healthcare:** Incorrect medical information could harm patients 🏥
  - **Finance:** False market information leads to poor decisions 📉
  - **Education:** Misleading facts undermine learning objectives 🎓
- **User Trust:**
  - One wrong answer can destroy confidence in the entire system 💔
  - Users can't always detect hallucinations, creating hidden risks 🕵️

Essential Technology

RAG isn't just a nice-to-have - it's essential for reliable AI systems! 🛡️

Why One-Size RAG Doesn't Fit All 👕👔👗

Different domains need different approaches:

- **Scientific Research** 🧪
  - Needs: Citations, uncertainty statements, evidence quality
  - Example: "What's the evidence for aerosol COVID transmission?"
- **Technical Support** 💻
  - Needs: Step-by-step instructions, clear solutions, actionable advice
  - Example: "How do I fix Android connectivity issues?"
- **Education** 🎓
  - Needs: Appropriate learning level, clear explanations, structured content
  - Example: "How do photosynthesis and respiration relate?"

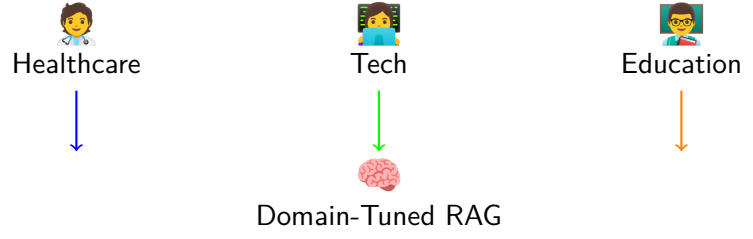
Think About It

Would you use the same language with a doctor, a mechanic, and a 5th grader?

Customizing Your RAG Pipeline 🧩

Customization Points

- **Data Sources** 📚
- **Embedding Models** 🧮
- **Retrieval Strategy** 🔎
- **Prompting Templates** 📝
- **Evaluation Metrics** 📊



Your Domain

How would you tune RAG for your specific use case?

What We'll Need 🧰

Tools & Libraries:

- 📦 `sentence-transformers`
- 🔎 `FAISS`
- 📚 `ir_datasets`
- 🤖 Mistral AI API
- 🤗 Hugging Face Hub

Computing Resources:

- 💾 RAM: 8GB+ recommended
- 🖥️ GPU: Nice to have!
- 💽 Storage: 2GB for indexes
- ☁️ Google Colab works well

Pro Tip

Use a free Colab instance with T4 GPU for this tutorial!

RAG in 60 Seconds ⏱️

Let's see a minimal example:

```
# 1. Create a tiny document collection
mini_docs = ["RAG stands for Retrieval-Augmented Generation in AI.", ...]

# 2. Create embeddings for documents
doc_embeddings = model.encode(mini_docs)

# 3. Index the embeddings with FAISS
index = faiss.IndexFlatL2(dimension)
index.add(doc_embeddings)

# 4. Convert query to a vector and search
query_vector = model.encode([query])
D, I = index.search(query_vector, k=2)

# 5. Generate answer using retrieved documents
```

Exploring Real-World RAG Applications 🌎

This tutorial covers 6 practical use cases:

1. **Scientific Research** 🧪
   - Dataset: `beir/trec-covid`
   - "What treatments work for COVID-19?"
2. **Technical Support** 💻
   - Dataset: `beir/cqadupstack/android`
   - "How do I fix network connectivity issues?"
3. **Education & Library** 🎓
   - Dataset: `beir/natural-questions`
   - "What teaching methods improve engagement?"

Choose Your Path

Which domain would you like to explore first?

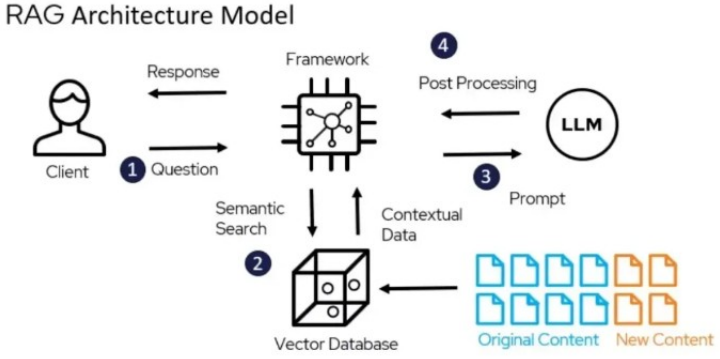
More Practical Use Cases 🔄

4. **Fact Verification** ✅
   - Dataset: `beir/fever`
   - "Do vaccines cause autism?"
5. **Healthcare Information** 🏥
   - Dataset: `beir/nfcorpus`
   - "What are the side effects of statins?"
6. **Campus Info** 🏫
   - Dataset: `custom_mst_site`
   - "Where is the ISSS office located?"

Today's Focus

For this tutorial, we'll select one domain and build a complete RAG solution for it!

Our RAG Architecture 🏗️



1. Convert query to embedding vector with Bi-Encoder 🔄
2. Retrieve top-k documents with FAISS (fast!) ⚡
3. Rerank results with Cross-Encoder (more accurate) 🎯
4. Generate answer with Mistral LLM using domain-specific prompting 🧠

Two-Stage Retrieval: Better Results 🎯

Why Two-Stage Retrieval?

- **Bi-encoders** (first stage)
  - Fast! 🏎️ Can search millions of documents quickly
  - But less precise than direct comparison
- **Cross-encoders** (reranking)
  - More accurate! 🎯 Compare query+document together
  - But too slow to run on entire corpus
- **Best of both worlds:** 🏆
  - Bi-encoder narrows down to 100 candidates
  - Cross-encoder carefully ranks the final top 5-10

Analogy

It's like using a metal detector to find a search area, then a fine-tooth comb to find the treasure! 🏝️

Retrieve-then-Rerank

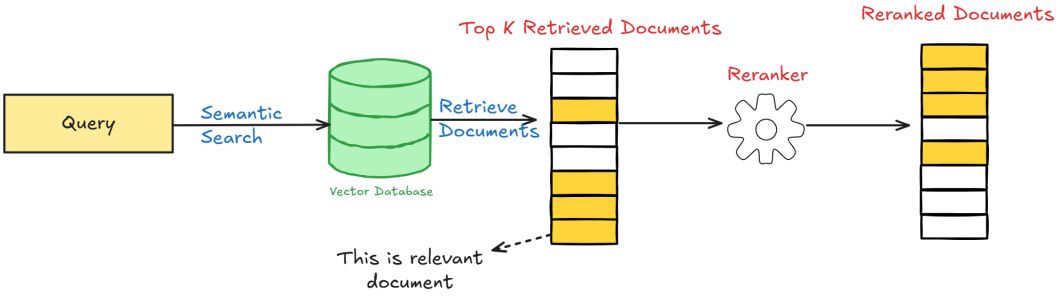


Figure: This two-stage pipeline begins with a retrieval phase, where a bi-encoder model or lexical search engine (e.g., Elasticsearch) retrieves a broad set of candidate documents based on the input query. Subsequently, in the reranking phase, a more sophisticated cross-encoder model evaluates the relevance of these candidates in the context of the query, producing a refined, ranked list of results. This approach balances retrieval efficiency with ranking precision, enhancing the quality of information retrieval systems.

The Core Retrieval Function 🔎

```python
# 1. First-stage retrieval with biencoder + FAISS
# Encode the query using the bi-encoder model
query_embedding = bi_encoder.encode(query, show_progress_bar=False, convert_to_numpy=True)
query_embedding = np.array([query_embedding], dtype=np.float32)
faiss.normalize_L2(query_embedding)  # Normalize for cosine similarity

# Search the FAISS index
scores, indices = index.search(query_embedding, k=min(top_k_first_stage, index.ntotal))
```

```python
# 2. Reranking with cross-encoder
cross_encoder_candidates = []
for idx in indices[0]:  # indices comes as a 2D array
    doc_id = doc_ids[idx]
    doc_info = corpus[doc_id]
    title = doc_info.get('title', '')
    text = doc_info['text']
    combined_text = f"{title}. {text}" if title else text
    cross_encoder_candidates.append([query, combined_text])

# Score with cross-encoder
cross_scores = cross_encoder.predict(cross_encoder_candidates)

# Sort by cross-encoder scores
cross_results = []
for i, idx in enumerate(indices[0]):
    doc_id = doc_ids[idx]
    biencoder_score = float(scores[0][i])  # Convert from numpy float to Python float
    cross_score = float(cross_scores[i])   # Ensure Python float
    cross_results.append((doc_id, biencoder_score, cross_score))

# Sort by cross-encoder score
cross_results = sorted(cross_results, key=lambda x: x[2], reverse=True)[:top_k_reranked]
```

Figure: **Left:** First-stage retrieval using bi-encoder and FAISS. The query is encoded into an embedding vector, normalized, and used to search the FAISS index for initial candidates based on cosine similarity. **Right:** Second-stage re-ranking with cross-encoder. Retrieved documents are paired with the query, scored by the cross-encoder model, and sorted to return the most relevant results.

Essential Point

The quality of your retrieval makes or breaks your RAG system. It's all about finding those golden nuggets of information! 💎

The Power of Domain Prompts 💪

Scientific Research Prompt

You are a scientific research assistant. Provide a clear, accurate, and evidence-based answer using only the retrieved documents. Cite supporting information, and include scientific context, limitations, and caveats where applicable.

Technical Support Prompt

You are a technical support specialist. Based on the retrieved documents, provide a concise and practical solution to the user's problem. Structure your answer step-by-step, cite relevant technical details, and ensure instructions are clear and executable.

Healthcare Information Prompt

You are a healthcare information assistant. Provide an accurate, evidence-based summary addressing the user's question. Do not offer medical advice. Focus on established research findings, clinical guidelines, and clearly state risks, limitations, or uncertainties.

Generating Domain-Appropriate Answers 🎯

```python
# Mistral API endpoint
api_url = "https://api.mistral.ai/v1/chat/completions"

# Create the messages list
messages = []

# Add system message with domain-specific prompt if provided
if domain_prompt:
    messages.append({
        "role": "system",
        "content": domain_prompt.strip()
    })

# Add user message with context and query
messages.append({
    "role": "user",
    "content": f"Context:\n{context.strip()}\n\nQuestion: {query.strip()}"
})

# Prepare the API request payload
payload = {
    "model": "mistral-large-latest",  # You can change to other Mistral models as needed
    "messages": messages,
    "max_tokens": max_length,
    "temperature": 0.7,
    "top_p": 0.9,
}

# Set up headers with API key
headers = {
    "Content-Type": "application/json",
    "Authorization": f"Bearer {api_key}"
}
```

```python
try:
    # Make the API request
    response = requests.post(api_url, headers=headers, json=payload)
    response.raise_for_status()  # Raise an exception for HTTP errors

    # Parse the JSON response
    response_data = response.json()

    # Extract the generated text
    generated_text = response_data["choices"][0]["message"]["content"]

    return generated_text

except Exception as e:
    print(f"Error calling Mistral API: {e}")
    # Fallback to a simple response if API call fails
    return f"I couldn't generate a response using the Mistral API due to an error: {str(e)}"
```

Figure: **Left:** Setting up the Mistral API call. We configure the API endpoint, create the messages list with system prompt and user query (including context from retrieval), and prepare the payload with model parameters. **Right:** Making the API request and handling the response. We send the request, handle potential errors, and extract the generated text from the JSON response.

Effective Prompting for RAG Systems 🪄

Essential Point 💡

The retrieved context combined with your query creates a powerful prompt for the LLM to generate accurate, contextually relevant responses.

Power of Prompting

The right prompt turns good retrieval into great answers! What domain-specific instructions would you add to your prompt?

- **Format Control:** Instruct the model on output structure (JSON, markdown, etc.)
- **Reasoning Chain:** Ask the model to explain its thought process step-by-step
- **Citation Style:** Request citations to specific retrieved documents
- **Confidence Signals:** Have the model indicate uncertainty levels for its answers
- **Role Definition:** Assign a specific expert persona to increase domain relevance

Measuring Success: Domain by Domain 📊

Different domains need different evaluation approaches

Scientific Research 🧪

- Citation accuracy
- Evidence quality
- Uncertainty handling

Technical Support 💻

- Step completeness
- Actionability
- Safety considerations

Education 🎓

- Grade-level appropriateness
- Concept scaffolding
- Engagement factors

Healthcare 🏥

- Clinical accuracy
- Completeness of warnings
- Information clarity

Domain-Specific Success

A good answer in one domain might be terrible in another! How do you judge success?

Custom Scientific Evaluation Example 🔬

```python
def evaluate_scientific_rag(query, retrieved_docs, generated_answer, ground_truth=None):
    """Evaluate a scientific RAG response with domain-specific metrics"""

    # 1. Check for citations in the answer
    citation_pattern = r'\[(\d+)\]|\(([A-Za-z\s]+,\s*\d{4})\)'
    has_citations = bool(re.search(citation_pattern, generated_answer))

    # 2. Check if answer mentions limitations or uncertainty
    uncertainty_terms = ['may', 'might', 'could', 'possibly', 'suggests',
                         'limited evidence', 'more research', 'not conclusive']
    has_uncertainty = any(term in generated_answer.lower() for term in uncertainty_terms)

    # 3. Check relevance of retrieved documents (simplified)
    # In practice, this might use a trained classifier or human evaluation
    relevance_score = sum(doc['cross_score'] for doc in retrieved_docs) / len(retrieved_docs)

    # 4. Calculate a combined scientific quality score (example only)
    scientific_quality = (
        (2 if has_citations else 0) +
        (1 if has_uncertainty else 0) +
        min(2, relevance_score / 3)  # Scale to 0-2 range
    ) / 5  # Normalize to 0-1

    # Return evaluation results
    return {
        'has_citations': has_citations,
        'acknowledges_uncertainty': has_uncertainty,
        'avg_relevance_score': relevance_score,
        'scientific_quality_score': scientific_quality,
    }
```

This evaluator checks:

- 📚 Are there citations to sources? `[1]` or `(Author, Year)`
- 🤔 Does it acknowledge uncertainty? “may,” “suggests,” “limited evidence”
- 📊 How relevant are the retrieved documents? (Cross-encoder scores)
- 🧪 Overall scientific quality score (combination of factors)

Custom Metrics

What metrics would you add for your specific domain? 🤔

IR-Based Evaluation Metrics 📏

Domain-Specific Metrics

- **Scientific Research:** nDCG@10, R@10, RR
  - *Prioritizes comprehensive literature review*
- **Technical Support:** RR, P@1, P@3
  - *Prioritizes finding the exact solution quickly*
- **Education:** nDCG@10, RBP(p=0.8)
  - *Models student patience when browsing results*
- **Healthcare:** P@1, P@3, RR
  - *Critical to get accurate information at the top*

User-Centered Metrics

Choose metrics that match your users' actual information needs! 🎯

Putting It All Together

```python
def run_rag_pipeline(query, query_id=None, top_k_first_stage=100, top_k_reranked=5, domain_prompt="", show_timings=False):
    """Run the full RAG pipeline with Mistral API for generation"""
    # Timing dictionary
    timings = {}

    # 1. Retrieve and rerank documents
    start_time = time.time()
    retrieved_docs = retrieve_documents(query, top_k_first_stage, top_k_reranked)
    timings['retrieval'] = time.time() - start_time

    # 2. Format context for LLM
    context = ""
    for i, doc in enumerate(retrieved_docs):
        context += f"Document {i+1} [Score: {doc['cross_score']:.3f}]:\n{doc['text']}\n\n"

    # 3. Generate answer with Mistral API
    start_time = time.time()
    answer = generate_answer(query, context, domain_prompt)
    timings['generation'] = time.time() - start_time

    # Total time
    timings['total'] = timings['retrieval'] + timings['generation']

    # Create run (for ir_measures evaluation)
    run = {}
    if query_id is not None:
        run[query_id] = {doc['doc_id']: float(doc['cross_score']) for doc in retrieved_docs}

    return retrieved_docs, answer, timings, run
```

Action Time

Now let's see it in action with a real query!

Example: COVID-19 Treatment Research Query

What evidence supports the effectiveness of Remdesivir in treating COVID-19 patients?

Retrieved Documents (Top 2)

Document 1 [Score: 8.94]:

A randomized controlled trial of Remdesivir showed **shortened recovery time in hospitalized COVID-19 patients compared to placebo. The median recovery time was 11 days with Remdesivir vs. 15 days with placebo.** However, **no significant mortality benefit was observed.**

Document 2 [Score: 7.65]:

The FDA granted **Emergency Use Authorization for Remdesivir based on preliminary data showing potential benefits in severe COVID-19 cases.** Some studies suggest it may **reduce hospital stay duration when administered early** in the disease course.

Generated Answer

Based on the retrieved information, Remdesivir has shown mixed evidence for effectiveness in COVID-19 treatment. **Clinical trials indicate it may shorten recovery time (11 days vs. 15 days with placebo), but no significant mortality benefit has been demonstrated**[1].

The FDA granted Emergency Use Authorization based on preliminary data, with **some studies suggesting it could reduce hospital stay when given early** in the disease course[2].

However, more research is needed to fully establish its clinical benefits across different patient populations and disease stages.

Analysis: What Makes This Answer Good? 🔎

- **Evidence-Based:** 📚 Draws directly from scientific studies
- **Balanced:** ⚖️ Presents both positive findings and limitations
- **Citations:** 📝 References sources with [1], [2]
- **Uncertainty:** 🤔 “mixed evidence,” “may shorten,” “more research needed”
- **Specific Details:** 📊 “11 days vs. 15 days”
- **Domain-Appropriate:** 🧪 Scientific tone without being overly technical

Domain Alignment

This answer succeeds because it follows the scientific domain prompt guidance! 🎯

Domain-Specific Best Practices 🏆

Scientific Research 🧪

- Use domain-specific embeddings
- Include publication date ranking
- Implement citation tracking
- Use subject-specific vocabulary

Technical Support 💻

- Optimize for high precision
- Structure answers as procedures
- Use hybrid retrieval approaches
- Add feedback mechanisms

Education 🎓

- Organize by difficulty level
- Build learning path generation
- Support multiple learning styles
- Use readability scores

Healthcare 🏥

- Assess evidence quality
- Prioritize recency factors
- Use medical taxonomies
- Include appropriate disclaimers

Your Turn

What specific optimizations would benefit your use case most? 🤔

Common RAG Problems & Solutions 🛠️

Troubleshooting RAG Systems

- **Problem:** Irrelevant documents retrieved 🤨
  - **Solutions:** Try domain-specific embeddings, adjust chunking strategy, add reranking
- **Problem:** Missing information (can't find what you know exists) 🤷
  - **Solutions:** Increase k value, try hybrid retrieval, check document preprocessing
- **Problem:** Hallucinations despite RAG 👻
  - **Solutions:** Strengthen prompt constraints, implement fact-checking, try different LLM
- **Problem:** Poor integration of retrieved content 🔄
  - **Solutions:** Improve context formatting, adjust prompt, try chain-of-thought

Share Your Experience

Debugging RAG is an art form - what problems have you encountered? 🎨

Taking RAG to Production 🚀

1. **Scale Your Data Pipeline** 📈
   - Build efficient document ingestion flows
   - Set up regular knowledge updates
   - Consider hybrid storage solutions
2. **Optimize Performance** ⚡
   - Benchmark vector search at scale
   - Implement caching strategies
   - Consider quantized embedding models
3. **Add Advanced Techniques** 🧠
   - Query rewriting/decomposition
   - Multi-stage retrieval architectures
   - Hybrid search methods
4. **Implement Monitoring** 👀
   - Track quality metrics over time
   - Watch for knowledge gaps
   - Create user feedback loops

Real-World RAG: Query Routing 🧭

Beyond Single Indexes

In production, you'll often need multiple knowledge sources:

- 📃 Internal documentation (Confluence, GitHub)
- 🌐 Public product FAQs and marketing
- 💬 Customer support tickets or chat logs
- 📜 Regulatory PDFs, whitepapers, guidelines
- 🔎 Web content or external APIs

Query Routing decides which sources to check for each question!

- **LLM-Based Routing:** 🧠 Use an LLM to classify query intent
- **Semantic Routing:** 📊 Compare to representative vectors
- **Keyword Routing:** 🔤 Set up triggers or patterns
- **Hybrid Approaches:** 🔄 Combine multiple strategies

Specialized Experts

Think of it like having specialized experts for different questions! 🧑‍💻👩‍💻👩‍🔬

Core Principles for Domain-Specific RAG 🧠

1. **Know Your Domain** 🔎
   - Understand specific information needs
   - Identify domain language and terminology
   - Recognize evidence standards and formats
2. **Customize Each Component** 🔧
   - Data: Select domain-appropriate content
   - Embedding models: Choose specialized ones
   - Prompting: Design domain-specific instructions
   - Evaluation: Measure what matters to users
3. **Iterate with Feedback** 🔄
   - Collect domain expert evaluation
   - Analyze failures to identify patterns
   - Update knowledge as domains evolve

Continuous Improvement

Great RAG systems aren't built in a day - they improve with each iteration! 📈

Why Domain-Specific RAG Matters 💯

Tailoring your RAG system brings amazing benefits:

- **Specialized Knowledge** 🧠
  - Retrieves content perfectly aligned with domain needs
- **Better Contextual Understanding** 🔎
  - Domain-aware prompting = higher-quality answers
- **Targeted Evaluation** 📊
  - Metrics that match what users actually care about
- **Enhanced User Experience** 😄
  - Formatting and style that fits expectations
- **Higher Relevance** 🎯
  - More precise retrieval = better answers

True Value

The difference between a generic tool and an expert assistant! 🌟

Advanced RAG Architectures 🚀

Ready to level up? Try these advanced approaches:

- **Multi-hop RAG** 🦘
  - Performs multiple retrieval jumps for complex reasoning
  - "Who was the teacher of the person who discovered penicillin?"
- **Hypothetical Document Embeddings (HyDE)** 💭
  - Generates a fake "perfect" document first
  - Uses this to improve retrieval quality
- **Knowledge Graph RAG** 🕸️
  - Combines vector search with structured knowledge
  - Follows relationships between entities
- **Self-RAG** 🪞
  - System decides when to retrieve more information
  - Critiques its own answers and retrieval quality

Future Exploration

The RAG landscape is evolving rapidly - what will you build next? 🤔

Resources for Your RAG Journey 🗺️

Explore these resources to continue learning:

- **Datasets** 📚
  - ir_datasets: https://ir-datasets.com/
  - BEIR Benchmark: https://github.com/beir-cellar/beir
  - HuggingFace Datasets: https://huggingface.co/datasets?search=irds
- **Tools & Libraries** 🧰
  - Sentence Transformers: https://www.sbert.net/
  - FAISS: https://github.com/facebookresearch/faiss
  - LangChain: https://python.langchain.com/
  - LlamaIndex: https://docs.llamaindex.ai/
- **Research Papers** 📝
  - "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" (Lewis et al., 2020)
  - "Dense Passage Retrieval for Open-Domain Question Answering" (Karpukhin et al., 2020)

Fun Exercises to Try Next! 🎮

Challenge yourself with these exercises:

1. 🔄 **Switch domains!** Apply the same pipeline to a different use case
2. 🧪 **Experiment with models!** Try different encoders and LLMs
3. 🔎 **Test retrieval strategies!** Compare bi-encoder only vs. two-stage approach
4. 📝 **Craft better prompts!** See how prompt engineering affects answers
5. 🧠 **Build your own dataset!** Create a custom knowledge base for your interests
6. 🌐 **Make a web app!** Turn your RAG pipeline into an interactive interface

Learning By Doing

The best way to learn is by doing! Which exercise sounds most fun to you? 🤩

Thank You! 🙏

Questions?

Contact Information:
Shubham Chatterjee
Email: shubham.chatterjee@mst.edu
GitHub: github.com/shubham526

Final Thought

Remember: The best RAG system is one customized for YOUR specific needs! 🌟