

# Retrieval-Augmented Generation (RAG) Systems

Understanding Modern Information Retrieval and Generation

RAG Academic Assistant Team



## **Retrieval-Augmented Generation (RAG) Systems**

### **Understanding Modern Information Retrieval and Generation**

#### **Table of Contents**

1. What is RAG and Why Do We Need It?
2. Understanding Chunking: The Foundation of RAG
3. The Retrieval Process: From Query to Results
4. Vector Databases and ChromaDB
5. Hybrid Search: Combining Semantic and Keyword Search
6. Language Model Generation and Parameters
7. Real-World Applications and Examples
8. System Architecture Overview
9. Practical Implementation Considerations
10. Future of RAG Systems

# 1. What is RAG and Why Do We Need It?

## Project Implementation

This presentation is based on our RAG Academic Assistant project, which demonstrates these concepts in practice. You can explore the complete implementation at:

**GitHub Repository:** [github.com/naholav/rag-academic-assistant](https://github.com/naholav/rag-academic-assistant)

The project includes: - Full RAG pipeline implementation with ChromaDB - Support for multiple PDF documents - Dual-mode chat interface (RAG and Direct) - Turkish and English language support - Streamlit-based user interface

## The Problem with Traditional Language Models

Large Language Models (LLMs) like GPT, Claude, and Llama have revolutionized natural language processing, but they face critical limitations:

### Knowledge Limitations

- **Static Knowledge:** Models only know what they learned during training
- **No Real-Time Updates:** Cannot access information after their training cutoff date
- **Hallucination:** May generate plausible-sounding but incorrect information
- **No Source Attribution:** Cannot cite where information comes from

### The Solution: RAG

**Retrieval-Augmented Generation (RAG)** combines the power of: 1. **Information Retrieval:** Finding relevant information from a knowledge base 2. **Language Generation:** Using LLMs to create natural, coherent responses

Think of RAG as giving an AI assistant access to a library. Instead of relying only on memorized information, it can look up specific books and pages to answer your questions accurately.

## How RAG Transforms AI Applications

### Without RAG (Traditional LLM)

User: "What's our company's vacation policy?"

LLM: "I don't have access to your company's specific policies."

## **With RAG**

User: "What's our company's vacation policy?"

RAG System: "According to the Employee Handbook (Page 23), employees receive 15 vacation days in their first year, increasing to 20 days after three years of service."

## 2. Understanding Chunking - The Foundation of RAG

### What is a Chunk?

A **chunk** is a small, manageable piece of text extracted from a larger document. Think of it like breaking a book into paragraphs or pages that can be individually searched and retrieved.

### Why Do We Need Chunking?

1. **Context Window Limitations:** LLMs can only process a limited amount of text at once
2. **Efficiency:** Processing entire documents for every query would be slow and expensive
3. **Relevance:** We only need the specific parts that answer the user's question
4. **Cost:** Processing less text means lower computational costs

### Chunk Size: Finding the Sweet Spot

#### What is Chunk Size?

**Chunk size** is the number of tokens (words/subwords) in each text segment. Common sizes range from 200 to 1500 tokens.

#### Optimal Chunk Sizes and Trade-offs

Table 1: Chunk Size Comparison

Chunk Size	Tokens	Use Case	Advantages	Disadvantages
Small	200-400	Precise facts, definitions	High precision, specific answers	May lose context
Medium	500-800	General use, balanced	Good context, manageable	Balanced trade-offs
Large	1000-1500	Complex topics	Full context, complete ideas	Less precise, higher costs

#### Our Choice: 800 Tokens

We use 800 tokens because: - **Complete Ideas:** Enough space for 2-3 full paragraphs - **Sufficient Context:** Captures relationships between concepts -

**Efficient Processing:** Reasonable computational load - **Good Retrieval:** Not too broad to dilute relevance

## Chunk Overlap: Maintaining Context

### What is Chunk Overlap?

**Chunk overlap** is the number of tokens shared between consecutive chunks. It ensures important information at chunk boundaries isn't lost.

### Visual Example of Overlapping

Document: "The sky was blue. Birds were singing. It was a perfect day.  
Children played in the park. Their laughter filled the air."

Without Overlap:

Chunk 1: "The sky was blue. Birds were singing."

Chunk 2: "It was a perfect day. Children played in the park."

Problem: Lost connection between weather and children playing

With 150 Token Overlap:

Chunk 1: "The sky was blue. Birds were singing. It was a perfect day."

Chunk 2: "It was a perfect day. Children played in the park. Their laughter..."

Benefit: Maintains context between chunks

### Why 150 Token Overlap?

- **Context Preservation:** ~20% overlap maintains narrative flow
- **Boundary Information:** Captures complete sentences at edges
- **Retrieval Improvement:** Related information appears in multiple chunks
- **Storage Balance:** Reasonable increase in storage (not doubling data)

## 3. The Retrieval Process

### Candidates to Retrieve vs. Final Chunks Used

#### Stage 1: Candidates to Retrieve (Casting a Wide Net)

**Candidates to retrieve** refers to the initial number of potentially relevant chunks the system fetches from the database.

- **Typical Range:** 10-100 chunks
- **Our Setting:** 20 candidates
- **Purpose:** Ensure we don't miss relevant information

#### Stage 2: Final Chunks Used (Precision Selection)

**Final chunks used** are the top-ranked chunks actually sent to the LLM for answer generation.

- **Typical Range:** 3-10 chunks
- **Our Setting:** 5 chunks
- **Purpose:** Provide focused, relevant context without overwhelming the model

### The Filtering Process

User Query: "What are the performance metrics?"

↓

Step 1: Retrieve 20 candidate chunks (broad search)

- Chunk A: Performance metrics... (Score: 0.95)
- Chunk B: Benchmark results... (Score: 0.92)
- Chunk C: System performance... (Score: 0.89)
- ... (17 more chunks with lower scores)

↓

Step 2: Rerank using cross-encoder

↓

Step 3: Select top 5 chunks for generation

↓

Final Context: Only the 5 most relevant chunks

### Why This Two-Stage Approach?

1. **Recall vs. Precision:** First maximize recall (find everything), then maximize precision (keep only the best)
2. **Computational Efficiency:** Initial search is fast, detailed reranking only on subset

3. **Quality Assurance:** Multiple evaluation stages reduce chance of missing information
4. **Context Optimization:** LLM receives focused, high-quality information

## 4. Vector Databases and ChromaDB

### What is a Vector Database?

A **vector database** is a specialized database designed to store and search numerical representations (embeddings) of text, images, or other data.

### Traditional Database vs. Vector Database

Table 2: Database Comparison

Traditional Database	Vector Database
Stores exact data (text, numbers)	Stores numerical vectors (embeddings)
Searches by exact match or keywords	Searches by semantic similarity
"Find documents with 'car'"	"Find documents similar in meaning to 'automobile'"
Returns exact matches only	Returns conceptually related results

### ChromaDB: Our Vector Storage Solution

**ChromaDB** is an open-source vector database optimized for AI applications.

### Key Features of ChromaDB

1. **Persistent Storage:** Data survives system restarts
2. **Metadata Support:** Store additional information with vectors
3. **Efficient Indexing:** Fast similarity search using HNSW algorithm
4. **Local or Cloud:** Can run on your machine or in the cloud
5. **Simple API:** Easy integration with Python applications

### How ChromaDB Works in RAG

1. Document Processing:  
Text: "Machine learning revolutionizes data analysis"  
↓
2. Create Embedding:  
Vector: [0.23, -0.45, 0.67, 0.12, ...] (768 dimensions)  
↓
3. Store in ChromaDB:  
ID: chunk\_001  
Vector: [0.23, -0.45, 0.67, ...]

Metadata: {page: 5, source: "AI\_Guide.pdf"}

↓

4. Query Time:

User: "How does AI analyze data?"

Query Vector: [0.21, -0.43, 0.65, ...]

↓

5. Similarity Search:

ChromaDB finds vectors closest to query vector

Returns: chunk\_001 (similarity: 0.92)

## Why Vector Databases Matter

- **Semantic Understanding:** Find information by meaning, not just keywords
- **Multilingual:** Works across languages (similar concepts have similar vectors)
- **Scalability:** Efficiently search millions of documents
- **Flexibility:** Works with any type of content that can be embedded

## 5. Hybrid Search - The Best of Both Worlds

### Semantic Search Weight (70%)

**Semantic search** uses embeddings to find content based on meaning and context.

#### How Semantic Search Works

Query: "vehicle performance"

Semantic Search Finds:

- "car speed and efficiency" (different words, same meaning)
- "automobile capabilities" (synonyms understood)
- "transportation metrics" (related concepts)

#### Why 70% Weight?

- **Conceptual Understanding:** Captures the intent behind queries
- **Synonym Recognition:** Finds related terms automatically
- **Context Awareness:** Understands relationships between concepts
- **Language Flexibility:** Works across different phrasings

### BM25 Keyword Weight (30%)

**BM25 (Best Matching 25)** is a probabilistic ranking function for keyword-based search.

#### How BM25 Works

Query: "GPT-4 temperature parameter"

BM25 Finds:

- Documents with exact term "GPT-4"
- Documents with exact phrase "temperature parameter"
- Weights by term frequency and document length

#### Why 30% Weight?

- **Precision for Technical Terms:** Exact matches for acronyms, model names
- **Specific Terminology:** Medical terms, legal phrases, product names
- **Complementary Coverage:** Catches what semantic search might miss
- **Proven Reliability:** Decades of information retrieval research

## **The Power of Combination**

### **Example: Hybrid Search in Action**

Query: "OCR-Qwen-32B performance benchmarks"

Semantic Search (70%) finds:

- "model evaluation results"
- "accuracy measurements"
- "testing outcomes"

BM25 (30%) finds:

- Exact matches for "OCR-Qwen-32B"
- Exact matches for "benchmarks"

Combined Result: Documents that contain the specific model name  
AND discuss performance conceptually

## 6. Language Model Generation and Parameters

### Understanding LLM Parameters

#### Temperature (0.3)

**Temperature** controls the randomness/creativity of the model's output.

Temperature Scale:



#### Why 0.3 for RAG?

- **Factual Accuracy:** Low temperature ensures consistent, reliable answers
- **Reduced Hallucination:** Less likely to generate creative but incorrect information
- **Reproducibility:** Similar queries produce similar answers
- **Professional Tone:** Maintains formal, informative style

#### Examples at Different Temperatures

Query: "What is machine learning?"

Temperature 0.1 (Very Factual):

"Machine learning is a subset of artificial intelligence that enables systems to learn from data without explicit programming."

Temperature 0.3 (Balanced):

"Machine learning is a branch of AI where computers learn patterns from data to make decisions without being explicitly programmed for each specific task."

Temperature 0.8 (Creative):

"Machine learning is like teaching a computer to think by showing it examples, similar to how children learn by observing patterns in the world around them."

#### Top-p (0.95)

**Top-p (nucleus sampling)** limits the model to considering only the most probable tokens that sum to probability p.

Top-p Visualization:

All possible next words sorted by probability:

[the: 0.3] [a: 0.25] [an: 0.2] [this: 0.15] [that: 0.05] ...

↑        ↑        ↑        ↑  
Cumulative: 0.95 (stop here)  
Only consider these words

### Why 0.95?

- **Quality Control:** Excludes very unlikely (potentially wrong) options
- **Diversity:** Still allows variety in word choice
- **Natural Language:** Produces fluent, human-like text
- **Safety:** Reduces chance of generating inappropriate content

### Max Tokens (2048)

**Max tokens** sets the maximum length of the generated response.

Token Examples:

"Hello" = 1 token

"Machine learning" = 2 tokens

"The quick brown fox" = 4 tokens

Average: ~1.3 tokens per word

### Why 2048 Tokens?

- **Comprehensive Answers:** ~1500 words, enough for detailed explanations
- **Context Preservation:** Sufficient space for multi-part answers
- **User Experience:** Not overwhelming, readable in one sitting
- **Cost Efficiency:** Balances completeness with computational resources

## 7. Real-World Applications and Examples

### Example 1: Document-Based Question Answering

**Soru:** “OpenCodeReasoning’de en iyi model hangisi?”

**RAG System Process:**

1. **Query Embedding:** Convert Turkish query to vector
2. **Retrieval:** Find chunks about model performance
3. **Context Assembly:** Gather information about OCR-Qwen-32B
4. **Generation:** Produce answer with citations

**Cevap:** “OPENCODEREASONING’de en iyi model, OCR-Qwen-32B-Instruct olarak belirlenmiştir. Bu model, LiveCodeBench’te pass@1 oranı 61.7’dir (ortalama @64), bu değer DeepSeek-R1’in 65.9’una çok yakın olup, diğer açık kaynak modellerden üstündür. (Kaynak: Sayfa 2, 7)”

### Example 2: Out-of-Context Query Handling

**Soru:** “Selam”

When queries are unrelated to the document content, RAG systems have two options:

#### Option 1: Fallback to General LLM

No relevant chunks found → Use LLM's general knowledge

**Cevap:** “Merhaba! Size nasıl yardımcı olabilirim? Eğer belge içeriği hakkında sorularınız varsa, detaylı bilgi verebilirim.”

#### Option 2: Context-Aware Response

System recognizes greeting → Appropriate response without retrieval

This demonstrates RAG’s flexibility in handling both document-specific and general queries.

## Interactive Parameter Exploration with GitHub Implementation

To see these concepts in action and experiment with RAG parameters, explore our implementation:

**GitHub Repository:** [github.com/naholav/rag-academic-assistant](https://github.com/naholav/rag-academic-assistant)

The implementation provides an interactive interface where users can adjust parameters in real-time to understand how each affects the system:

## Retrieval Parameters

- **Candidates to Retrieve (10-100):** Start with more candidates for comprehensive coverage
  - Set to 10: Fast but might miss relevant info
  - Set to 50: Balanced coverage
  - Set to 100: Maximum recall, slower processing
- **Final Chunks Used (1-20):** How many chunks actually go to the LLM
  - Set to 1: Very focused, risk of incomplete context
  - Set to 5: Our default - balanced context
  - Set to 20: Maximum context, but may include noise

**Hybrid Retrieval Weights** Users can adjust the balance between search methods: - **Semantic Search Weight (0.0-1.0):** Understanding meaning and concepts - 0.0: No semantic search - 0.7: Our default - prioritizes meaning - 1.0: Only semantic, might miss exact terms

- **BM25 Keyword Weight (0.0-1.0):** Exact term matching
  - 0.0: No keyword search
  - 0.3: Our default - catches specific terms
  - 1.0: Only keywords, misses synonyms

**Generation Settings** Fine-tune how the LLM generates responses: - **Temperature (0.10-1.00):** Controls randomness/creativity - 0.10: Very deterministic, same answer every time - 0.30: Our default - factual but natural - 1.00: Creative, varied responses

- **Max Response Tokens (256-4096):** Response length limit
  - 256: Short, concise answers
  - 2048: Our default - comprehensive responses
  - 4096: Maximum detail possible
- **Top P (0.10-1.00):** Nucleus sampling for quality control
  - 0.10: Only highest probability tokens
  - 0.95: Our default - natural variation
  - 1.00: All possibilities considered

## Learning by Experimentation

By adjusting these parameters, users can observe: - How semantic vs keyword weights affect retrieval quality - The impact of temperature on response consistency - Trade-offs between speed and comprehensiveness - How different settings work better for different query types

This hands-on approach helps users internalize RAG concepts through direct experimentation rather than theoretical understanding alone.

## **When to Use RAG vs. Pure LLM**

Table 3: RAG vs Pure LLM Use Cases

<b>Use RAG When:</b>	<b>Use Pure LLM When:</b>
Need specific, factual information	General conversation
Require source citations	Creative writing
Working with proprietary data	Common knowledge questions
Information changes frequently	Philosophical discussions
Accuracy is critical	Brainstorming ideas

## 8. System Architecture Overview

### Complete RAG Pipeline

#### Input Processing Layer

RAG systems can handle various input formats, not just PDFs:

- **Documents:** PDF, Word, PowerPoint, Excel
- **Web Content:** HTML pages, blog posts, wikis
- **Databases:** SQL databases, APIs, knowledge graphs
- **Multimedia:** Transcribed audio, video captions, image descriptions

## The Universal RAG Workflow

1. Data Ingestion (Any Source)
  - ├─ PDF documents
  - ├─ Web scraping
  - ├─ Database exports
  - └─ API responses

↓
2. Preprocessing
  - ├─ Text extraction
  - ├─ Cleaning & formatting
  - ├─ Language detection
  - └─ Metadata extraction

↓
3. Chunking Strategy
  - ├─ Size: 800 tokens (optimal for most use cases)
  - ├─ Overlap: 150 tokens (maintains context)
  - └─ Boundary detection (sentence/paragraph aware)

↓
4. Embedding Generation
  - ├─ Model: all-mpnet-base-v2 (768 dimensions)
  - ├─ Batch processing for efficiency
  - └─ Cache embeddings for reuse

↓
5. Vector Storage (ChromaDB)
  - ├─ Persistent storage
  - ├─ Metadata indexing
  - └─ HNSW index for fast search

↓
6. Query Processing
  - ├─ User query → embedding
  - ├─ Hybrid search (70% semantic + 30% BM25)
  - ├─ Retrieve 20 candidates
  - └─ Rerank to top 5

↓
7. Generation (Qwen3-4B)
  - ├─ Temperature: 0.3 (factual)
  - ├─ Top-p: 0.95 (quality control)
  - ├─ Max tokens: 2048 (comprehensive)
  - └─ Streaming for real-time feedback

↓
8. Response Delivery
  - ├─ Answer with citations
  - ├─ Source attribution
  - └─ Confidence scoring

## Performance Metrics and Expectations

### System Performance Benchmarks

Table 4: Performance Metrics

Metric	Value	Explanation
Initial Setup	30-60 sec	One-time processing of documents
Query Response	1-3 sec	From question to answer
Accuracy	85-95%	Depends on document quality
Chunk Retrieval	<100ms	Vector similarity search
Generation Time	1-2 sec	LLM response generation

### Quality Indicators

1. **Retrieval Quality:** Measured by relevance of retrieved chunks
2. **Answer Accuracy:** Factual correctness compared to source
3. **Citation Precision:** Correct attribution to source documents
4. **Response Coherence:** Natural, well-structured answers

## 9. Practical Implementation Considerations

### Choosing the Right Parameters

#### Document Type Considerations

Table 5: Parameter Selection by Document Type

Document Type	Chunk Size	Overlap	Retrieval Count
Technical Manuals	1000-1200	200	15-20
Research Papers	800-1000	150	10-15
Legal Documents	600-800	150	20-25
General Content	500-800	100	10-15

#### Language Model Selection

Table 6: Model Size Comparison

Model Size	Use Case	Advantages	Trade-offs
4B Parameters	General use	Speed, efficiency	Less nuanced
7B Parameters	Balanced	Good quality	Moderate resources
13B+ Parameters	Complex	Highest quality	Slower, expensive

### Common Challenges and Solutions

#### Challenge 1: Information Not in Documents

**Scenario:** User asks about something not in the knowledge base

**Solution:** Implement fallback mechanisms: - Clearly state when information isn't available - Offer to search related topics - Provide general knowledge if appropriate

#### Challenge 2: Conflicting Information

**Scenario:** Different documents contain contradictory information

**Solution:** - Show both perspectives with sources - Highlight the discrepancy - Let user evaluate based on source credibility

### **Challenge 3: Language Mixing**

**Scenario:** Documents in English, queries in Turkish

**Solution:** - Use multilingual embeddings - Implement translation layer - Maintain language consistency in responses

## 10. Future of RAG Systems

### Emerging Trends

#### 1. Multimodal RAG

- Combining text, images, tables, and graphs
- Understanding relationships across media types
- Generating rich, multimedia responses

#### 2. Adaptive Chunking

- Dynamic chunk sizes based on content type
- Intelligent boundary detection
- Context-aware segmentation

#### 3. Personalized Retrieval

- Learning user preferences over time
- Adjusting weights based on query patterns
- Custom ranking algorithms

#### 4. Real-Time Knowledge Updates

- Continuous document ingestion
- Incremental index updates
- Version control for information

### Key Takeaways

1. **RAG bridges the gap** between static LLM knowledge and dynamic information needs
2. **Chunking is crucial**: 800 tokens with 150 overlap provides optimal balance
3. **Hybrid search** (70% semantic + 30% keyword) outperforms single approaches
4. **ChromaDB** enables efficient vector storage and retrieval
5. **Parameter tuning** (temperature=0.3, top-p=0.95, max\_tokens=2048) ensures quality outputs
6. **Two-stage retrieval** (20 candidates → 5 final) balances comprehensiveness and precision

7. **RAG is flexible:** Works with any document type, not just PDFs
8. **Context matters:** System can handle both document-specific and general queries appropriately

## Conclusion

RAG systems represent a paradigm shift in how AI applications access and utilize information. By combining the retrieval capabilities of modern search with the generation power of large language models, RAG enables:

- **Accurate, sourced responses** to complex queries
- **Dynamic knowledge bases** that can be updated without retraining
- **Cost-effective scaling** compared to fine-tuning models
- **Transparent AI** that shows its reasoning and sources

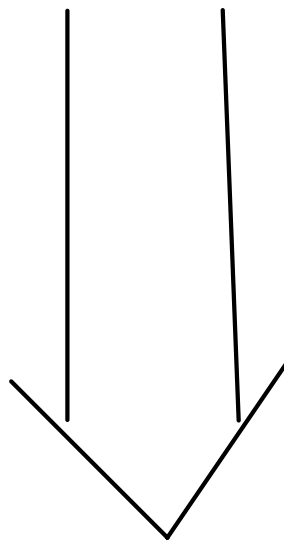
Whether applied to academic research, enterprise knowledge management, or customer support, RAG systems provide a robust framework for intelligent information retrieval and generation.

The key to successful RAG implementation lies in understanding and optimizing each component - from chunk size and overlap to retrieval weights and generation parameters. With proper configuration, RAG systems can transform how organizations leverage their information assets.

---

*Thank you for your attention!*

**Questions?**



# RAG Academic Assistant

Powered by Qwen3-4B-Instruct and ChromaDB

## Chat

merhaba

Merhaba, sorunuza cevap veremedim çünkü sorduğunuz "merhaba" ifadesi bir soru ya da istek olarak yorumlanamaz. Lütfen daha net ve bilgi içeren bir soru soralım. Örneğin, "Efficient reasoning in LLMs nasıl çalışır?" veya "Large language models'in kendi açıklamalarını yapma yeteneği nedir?" gibi bir soru ile ilgili detaylı bilgi sunabilirim.

Ask a question about the documents...

## Retrieved Sources

&gt; FaithLM - Page 10

&gt; StopOverthinking - Page 19

&gt; StopOverthinking - Page 20

&gt; FaithLM - Page 9

&gt; StopOverthinking - Page 30

&gt; StopOverthinking - Page 28

&gt; StopOverthinking - Page 8

&gt; FaithLM - Page 10

&gt; FaithLM - Page 9

&gt; FaithLM - Page 15

## Documents

OpenCodeReasoning.pdf

StopOverthinking.pdf

FaithLM.pdf

## Chat Mode

☒ Enable RAG  
(Document Search)RAG Mode: Searching  
documents

## Settings

Retrieval

Merhaba sorusuna context dışı  
olduğu için bu şekilde yanıt  
veriyor.

Rag modelinde:

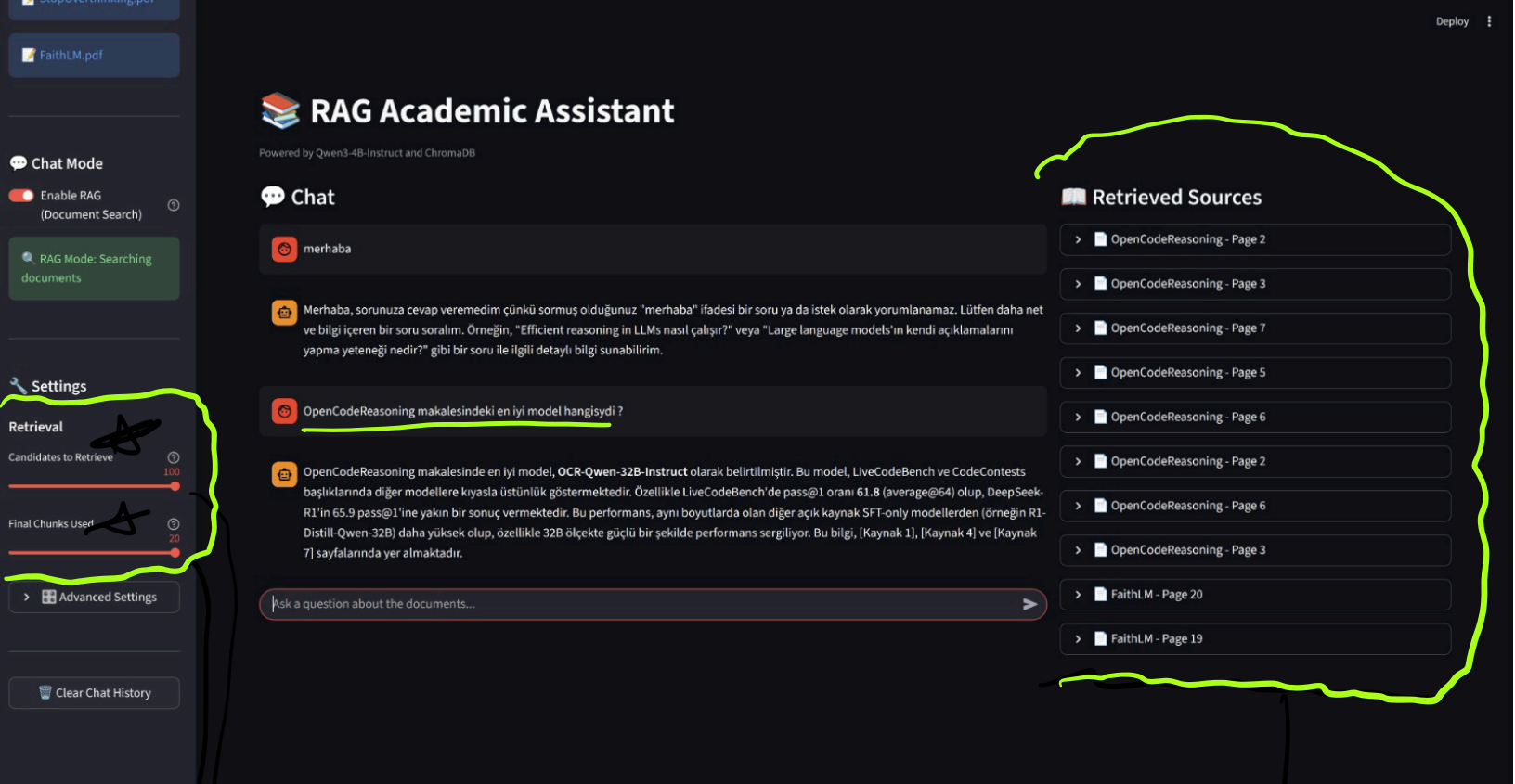
3 adet pdf var.

1) Open Code Reasoning

2) Faith LM

3) Stop Overthinking

} Makale



Her iki değerde çok yüksek

Verdiğimiz  
parametreye  
göre değişiyor

**Chat Mode**

Enable RAG (Document Search)

RAG Mode: Searching documents

**Settings**

**Retrieval**

Candidates to Retrieve: 10

Final Chunks Used: 1

> Advanced Settings

Clear Chat History

**Chat**

merhaba

Merhaba, sorunuza cevap veremedim çünkü sorduğunuz "merhaba" ifadesi bir soru ya da istek olarak yorumlanamaz. Lütfen daha net ve bilgi içeren bir soru soralım. Örneğin, "Efficient reasoning in LLMs nasıl çalışır?" veya "Large language models'in kendi açıklamalarını yapma yeteneği nedir?" gibi bir soru ile ilgili detaylı bilgi sunabilirim.

OpenCodeReasoning makalesindeki en iyi model hangisidi ?

OpenCodeReasoning makalesinde en iyi model, **OCR-Qwen-32B-Instruct** olarak belirtilmiştir. Bu model, LiveCodeBench ve CodeContests başlıklarında diğer modellere kıyasla üstünlük göstermektedir. Özellikle LiveCodeBench'de pass@1 oranı **61.8** (average@64) olup, DeepSeek-R1'in 65.9 pass@1'ine yakın bir sonuç vermektedir. Bu performans, aynı boyutlarda olan diğer açık kaynak SFT-only modellerden (örneğin R1-Distill-Qwen-32B) daha yüksek olup, özellikle 32B ölçekte güçlü bir şekilde performans sergiliyor. Bu bilgi, [Kaynak 1], [Kaynak 4] ve [Kaynak 7] sayfalarında yer almaktadır.

OpenCodeReasoning makalesindeki en iyi model hangisidi ?

Verilen bağlamda, OpenCodeReasoning makalesinde çözüm kodu üretimi adımı için kullanılan en iyi veya ana model, **DeepSeek-R1** olarak belirtilmiştir. Bu bilgi, metnin "2.2 Solution Code Generation" bölümüne göre verilmiştir. Burada, DeepSeek-R1'in Python ve C++ dillerinde çözümler üretmesi, özellikle daha zor olan IOI benchmark'ı test etmek amacıyla C++ kullanılarak yapılan deneylerde de vurgulanmıştır. Ayrıca, çözüm üretimi sırasında Nucleus Sampling ile sıcaklık (temperature) 0.6, top-p 0.95 değerleri ve "örnek" etiketi ("injecting 'thinking' tag") kullanıldığı da belirtilmektedir.

Modelin seçildiği nedeniyle, bu modelin özellikle akıl yürütme izleri oluşturmaya (reasoning traces) gerektiren görevlere uygun olduğu ifade edilmektedir. Ancak, bu süreçte çözüm üretimi için kullanılan modelin "en iyi" olup olmadığına dair doğrudan bir değerlendirme yapılmamıştır. Yani, diğer modellerle karşılaştırıldığında "en iyi" olma durumu doğrudan belirtilememektedir.

Sonuç olarak, OpenCodeReasoning çalışmasında çözüm üretimi için kullanılan ana model **DeepSeek-R1**'dir. Bu bilgi, kaynaktaki yer alan 2.2 bölümünde (sayfa 3) yer almaktadır.

Cevap: DeepSeek-R1 modeli. (Kaynak: Sayfa 3)

Aynı soru ama değerler en düştükte.  
Ürettiği metinler arasındaki kalite  
farkına bkn2.

stopoverthinking.pdf

FaithLM.pdf

Chat Mode

Enable RAG (Document Search)

RAG Mode: Searching documents

Settings

Retrieval

Candidates to Retrieve 100

Final Chunks Used 20

Advanced Settings

Clear Chat History

Chat

stop overthinking makalesi ne anlatıyor, özetler misin?

"Stop Overthinking" makalesi, Büyük Dilsel Modellerinde (LLM'lerde) verimli düşünme (efficient reasoning) konusunu sistematik bir şekilde araştırarak yapılandırılmış bir özetlemiştir. Bu çalışma, uzun ve gereksiz olan "düşünme aşırılığı" (overthinking) fenomeninin, özellikle Chain-of-Thought (CoT) düşünme sürecinde, gereksiz ve tekrar eden çıktılar üretmesi nedeniyle maliyet artışı ve yavaş yanıt süreleri ile sonuçlandığını belirtir.

Makale, verimli düşünmeyi üç ana yönde sınıflandırır:

- Model temelli verimli düşünme:** Tam uzunlukta düşünme modellerinin daha kısa ve sade hâle getirilmesi veya doğrudan verimli düşünme modellerinin eğitilmesi.
- Düşünme çıktısı temelli verimli düşünme:** Çalışma sırasında düşüncelerin adım sayısının ve uzunluğunun dinamik olarak azaltılması.
- Girdi uyarıları temelli verimli düşünme:** Düşünme verimliliğinin, giriş uyarısının zorluğu ya da uzunluğu gibi özelliklerine göre artırılması.

Makale ayrıca verimli verilerin kullanımına, küçük dilsel modellerin düşünme kapasitesine, değerlendirme yöntemlerine ve standartlara dair tartışmalar içerir. Örneğin, [4] sayfasında "overthinking" fenomeni, analiz paralizisi, kötü davranışlar ve erken çözmeye geçiş gibi desenlerle incelenmiş ve bu durumların model başarımını düşürdüğü gösterilmiştir. Ayrıca, [143] ve [145] kaynaklarında, düşünce zinciri uzunluğu ile model çıktısının etkinliği arasında güçlü ilişki olduğu ve güvenilirlik ile ilgili kritik noktaların tespit edildiği belirtilmektedir.

Bu çalışmanın önemli bir bölümü, verimli düşünmenin uygulama alanlarına katkılarını açıklar: otomatik sürüş sistemlerinde karar alma süreçlerinin iyileştirilmesi ([4]), sağlık alanında doktorlar için hastane kayıtlarının hızlı analiz edilmesi ([5]), ve önerme sistemlerinde kullanıcı tercihlerine göre daha kişiselleştirilmiş önerilerin oluşturulması ([5]).

Sonuç olarak, verimli düşünme, yalnızca işlem maliyetini azaltmakla kalmaz; aynı zamanda güvenlik ve verimlilik arasındaki gerilimi de ele alır. Örneğin, güvenlik önlemleri uzun düşünceler gerektirirken, verimlilik odaklı yaklaşım, modelin kendi cevabını kontrol etme veya karşı saldırıya direnç kazanma yeteneğini azaltabilir ([5]). Bu nedenle, güvenliğin verimlilikten ayrıştırılmasının olası olduğunu vurgular.

Bu makale, verimli düşünme üzerine yapılan çalışmaların gelişimini takip etmek amacıyla açık kaynaklı bir deposu da sunar ([1]).

Özet: Makale, büyük dil modellerinde verimli düşünmenin önemini vurgular, bunu üç ana kategoriye ayırarak inceleyerek, uzun düşünce zincirinin maliyeti ve başarımı üzerindeki etkisini, uygulama alanlarını ve güvenlik-verimlilik dengesini analiz eder.

Ask a question about the documents...

Retrieved Sources

StopOverthinking - Page 1

StopOverthinking - Page 23

StopOverthinking - Page 28

StopOverthinking - Page 18

StopOverthinking - Page 19

FaithLM - Page 21

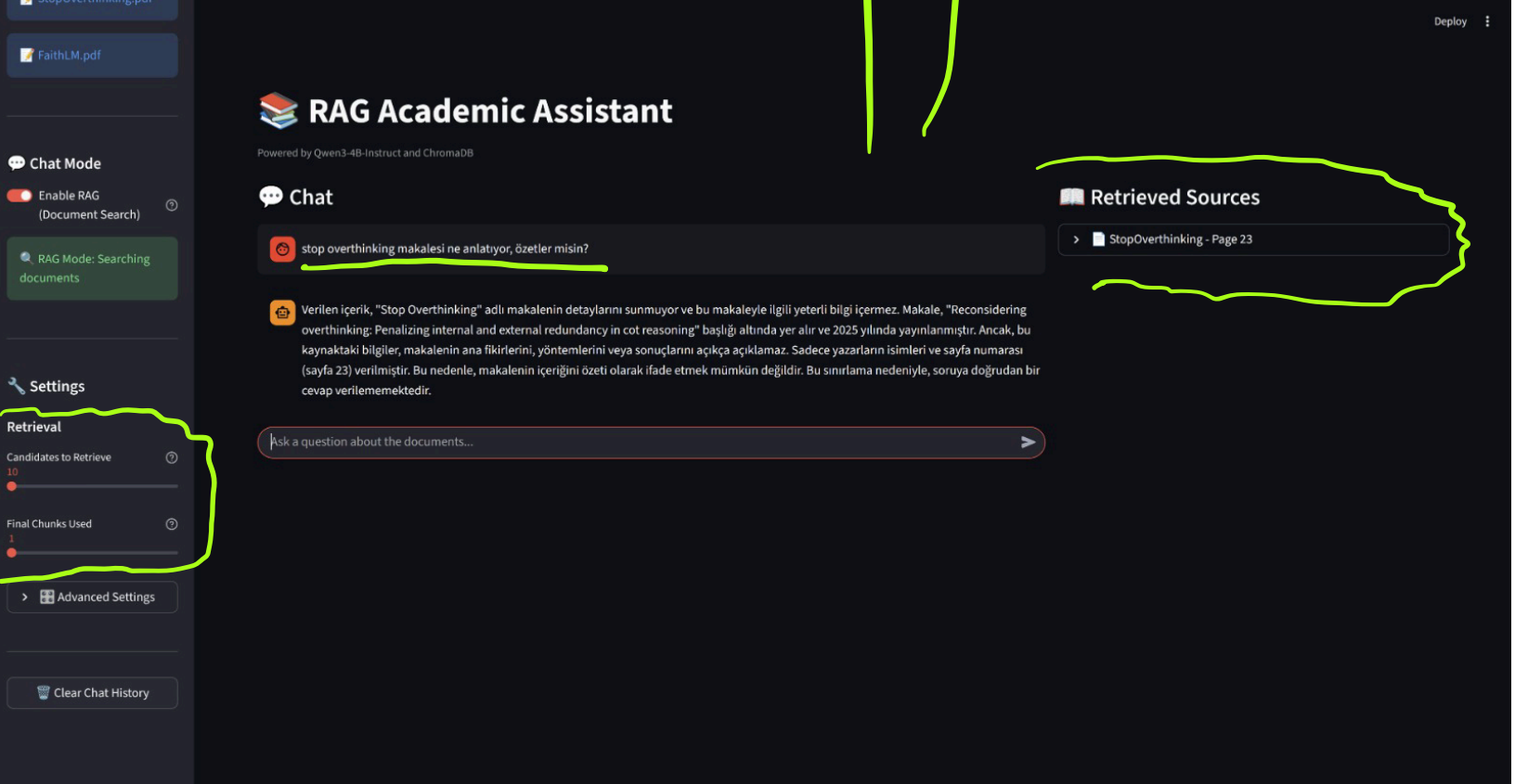
FaithLM - Page 8

FaithLM - Page 23

StopOverthinking - Page 26

StopOverthinking - Page 29

yukarıdakiyle aynı örnek, aynı mantık, sadece soru farklı.



Repo'nun tamamı github adresinde  
mevcut. NVIDIA ekran kartlarında daha  
stabil çalışacağını hatırlatmak isterim.  
(LLM → QWEN yüzünden)