Лекция 11: Hallucinations и RAG

Grounding LLMs в реални данни

Цели на лекцията

- Какво са hallucinations и защо се появяват
- Категории и последици на халюцинациите
- Embeddings и semantic search
- Vector databases — как работят
- RAG pipeline — от документи до отговор
- Практически strategies за quality

Част 1: Проблемът с Hallucinations

Какво е Hallucination?

Определение: Confident генериране на невярна информация

```
User: Кой написа "Война и мир"?
Model: "Война и мир" е написана от Лев Толстой през 1869. ✓

User: Кой написа "Отвъд хоризонта на звездите"?
Model: "Отвъд хоризонта на звездите" е роман от
       българския автор Петър Караиванов, публикуван
       през 1987 г. ✗
       (книга и автор не съществуват)
```

Категории hallucinations

Тип	Пример
Factual errors	Грешни дати, имена, числа
Fabricated citations	Измислени papers, книги
Impossible claims	Физически невъзможни неща
Outdated info	Остаряла информация
Confident nonsense	Убедително звучащи глупости

Защо LLMs халюцинират?

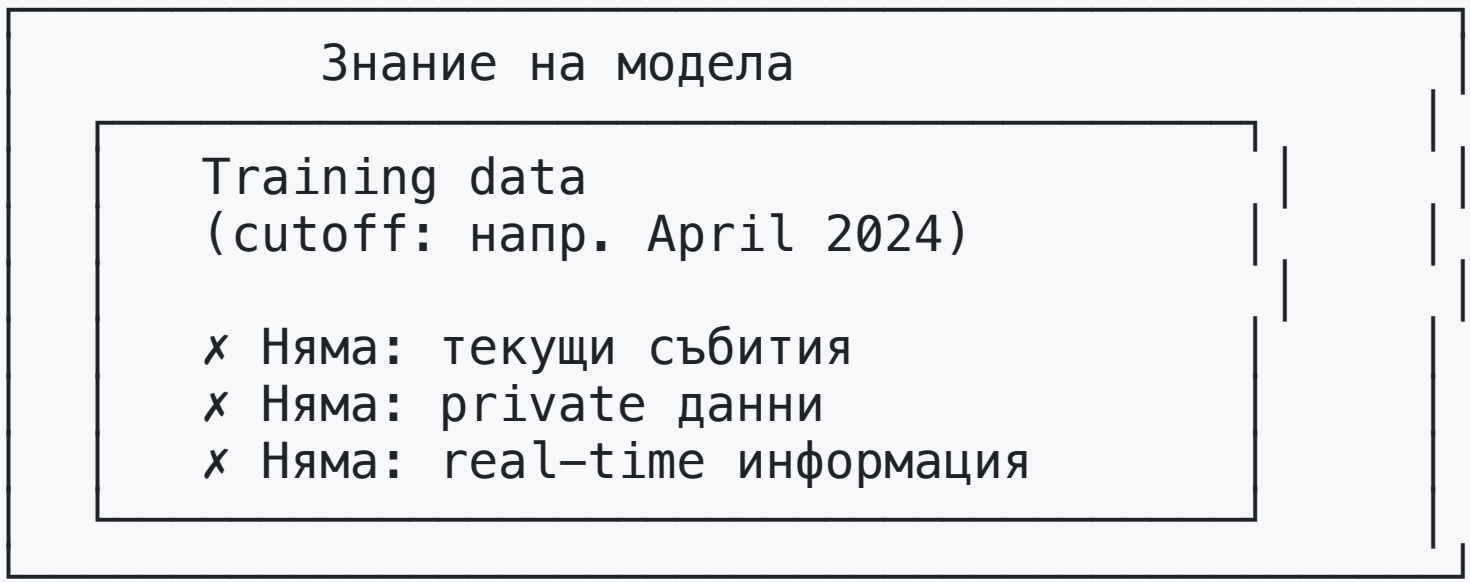
```
Training objective: P(next_token | context)

NOT: P(true_statement | question)
```

Моделът е обучен да **предвижда вероятен текст**,

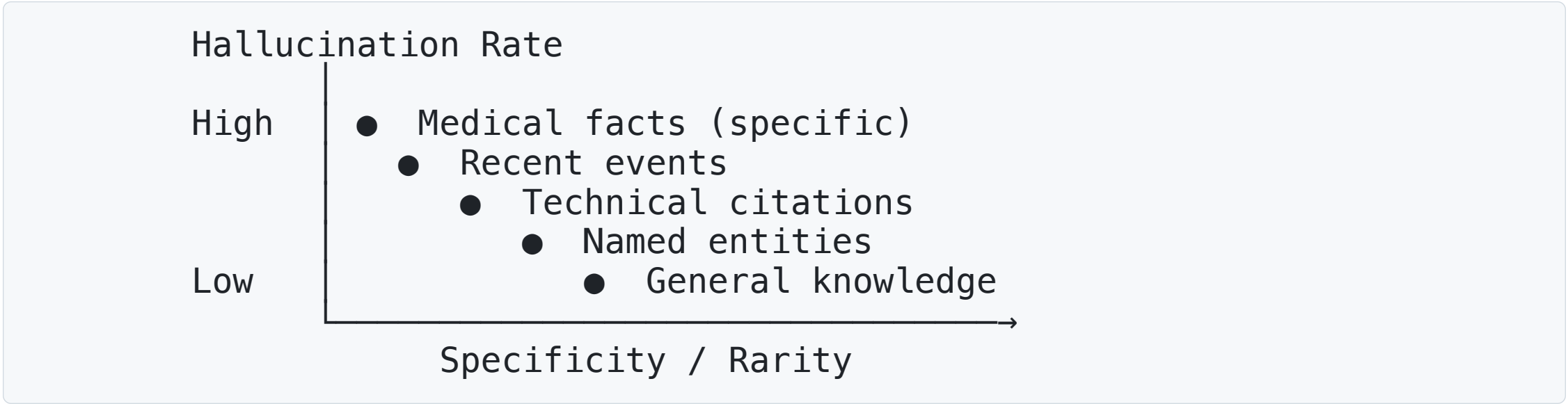
не да **верифицира истинност**

```
"Столицата на България е" → "София" ✓ (вероятно)
"Автор на несъществуваща книга" → [нещо убедително] ✗
```

Фундаменталното ограничение



Hallucination rates



По-специфични въпроси = повече hallucinations

Част 2: Когато Hallucinations са проблем

Spectrum на риска

Low stakes:

```
User: Дай ми рецепта за шоколадова торта
Model: [грешна стъпка] → Торта не става перфектна, но OK
```

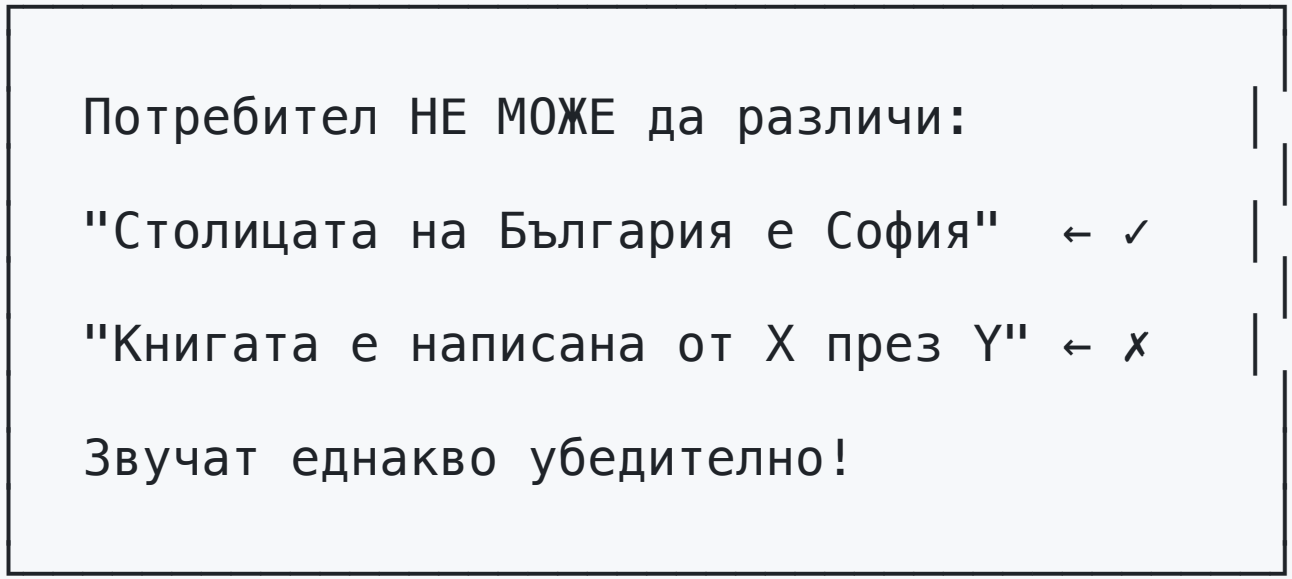
High stakes:

```
User: Каква е дозата на лекарство X за дете?
Model: [грешен отговор] → Потенциално фатално
```

Проблемни домейни

Домейн	Риск	Защо
Медицина	Критичен	Живот и здраве
Право	Висок	Неверни закони/случаи
Финанси	Висок	Финансови загуби
Новини	Среден	Дезинформация
Образование	Среден	Научава грешни неща

Trust Problem



"Бъди внимателен" не работи

```
Prompt: Answer carefully and only if you're sure.

Model: I'm confident that... [hallucination]
```

Проблем: Моделът не знае какво не знае

Self-reported confidence ≠ actual accuracy

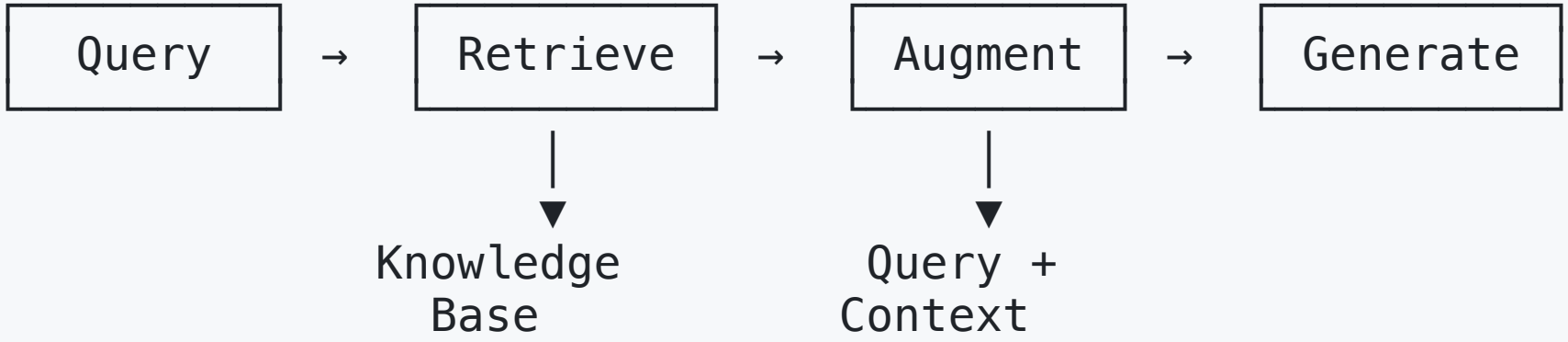
Решението: External Knowledge

```
Вместо:
  Question → Model → [може да халюцинира]

По-добре:
  Question → Retrieve relevant docs → Model + docs → Answer
                        ↑
               Grounded в реални данни
```

Част 3: RAG Overview

Какво е RAG?

Retrieval-**A**ugmented **G**eneration



RAG Pipeline

```
1. User Query: "Какви са vacation policies?"
                        |
                        ▼
2. Embed query → [0.12, -0.34, 0.56, ...]
                        |
                        ▼
3. Search vector DB → Top 3 relevant chunks
                        |
                        ▼
4. Augment prompt:
   "Based on these documents: [chunks]
    Answer: Какви са vacation policies?"
                        |
                        ▼
5. LLM generates grounded answer
```

Защо RAG намалява hallucinations?

Без RAG:

```
Model → Training data → Може да няма info → Hallucinate
```

С RAG:

```
Model → Retrieved docs → Explicit source → Grounded answer
```

- Модел работи с **конкретен контекст**
- Може да каже "According to document X..."
- По-лесно да refuse ако няма info

RAG Trade-offs

Предимство	Недостатък
По-малко hallucinations	Добавена latency
Актуална информация	Нужна infrastructure
Цитируеми източници	Retrieval може да fail
Private data support	Качество зависи от docs

Част 4: Text Embeddings

От думи към вектори

```
"котка"     → [0.21, −0.45, 0.67, ...]  (dim=384–1536)
"куче"      → [0.19, −0.42, 0.71, ...]  ← близо до котка
"автомобил" → [−0.31, 0.55, −0.22, ...] ← далеч
```

Embedding: Dense vector representation на текст

Семантично подобни → Геометрично близки

Защо не keyword search?

```
Query: "Как да подам оплакване за работодателя?"

Document: "Процедура за жалба срещу компанията"

Keyword match: 0 общи думи!
Semantic match: Високо сходство ✓
```

Embeddings catch:

- Синоними
- Парафрази
- Conceptual similarity

Cosine Similarity

$$\text{similarity}(A, B) = \frac{A \cdot B}{||A|| \cdot ||B||}$$

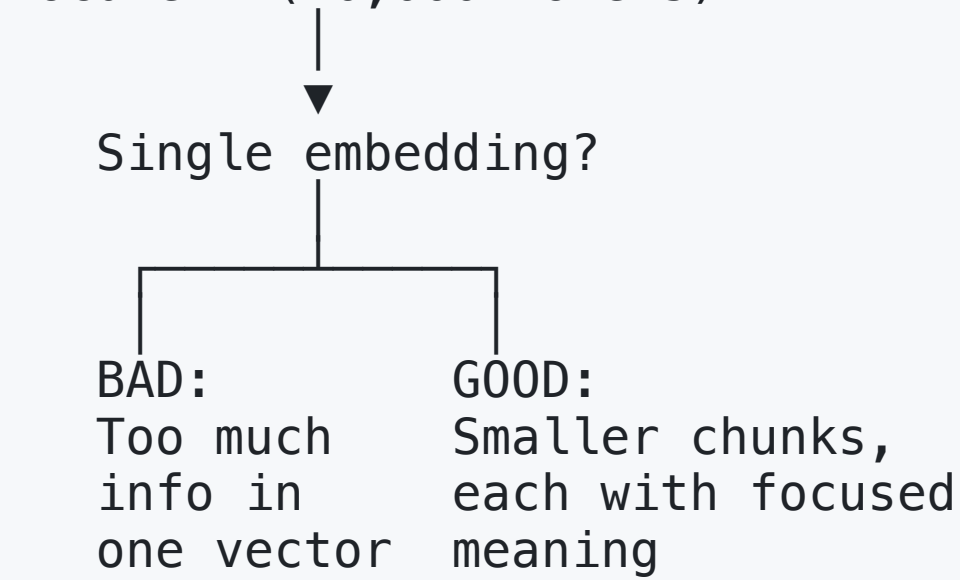
```
similarity = 1.0   → Identical meaning
similarity = 0.8+  → Very similar
similarity = 0.5   → Somewhat related
similarity = 0.0   → Unrelated
similarity = -1.0  → Opposite meaning
```

Embedding Models

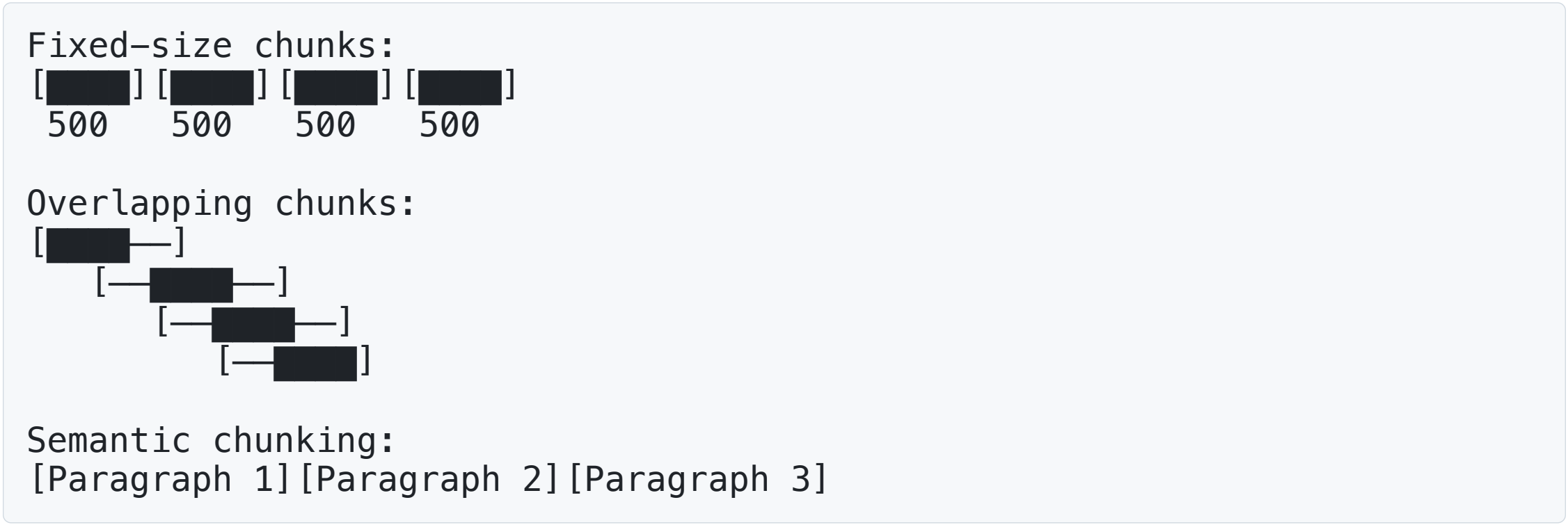
Model	Dimensions	Quality	Speed
OpenAI text-embedding-3-small	1536	High	Fast
OpenAI text-embedding-3-large	3072	Higher	Slower
all-MiniLM-L6-v2	384	Good	Very fast
BGE-large	1024	High	Medium
E5-large	1024	High	Medium

Избор: Баланс между качество и inference cost

Chunking: Защо?



Chunking Strategies



Chunking Parameters

Parameter	Typical Value	Trade-off
Chunk size	200-1000 tokens	Smaller = precise, larger = context
Overlap	10-20%	More = redundancy, less = gaps
Strategy	Semantic > Fixed	Semantic = better, harder

Practical default: 500 tokens, 50 token overlap

Част 5: Vector Databases

Проблемът

```
Имаш: 1,000,000 документа embeddings
Query embedding: [0.12, -0.34, ...]

Naive search:
  Compare with ALL 1M vectors
  O(N × D) operations
  Too slow!
```

Решение: Approximate Nearest Neighbor (ANN)

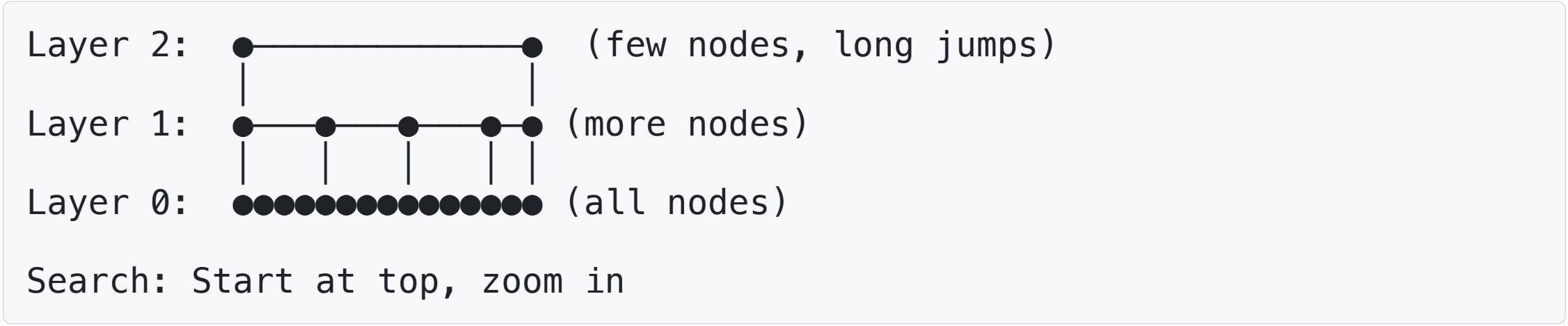
Exact vs Approximate

Approach	Complexity	Accuracy	Use Case
Exact (brute force)	O(N)	100%	Small datasets
Approximate (ANN)	O(log N)	~95-99%	Production

Trade-off: Малка загуба в recall за огромна скорост

HNSW Algorithm

Hierarchical **N**avigable **S**mall **W**orld

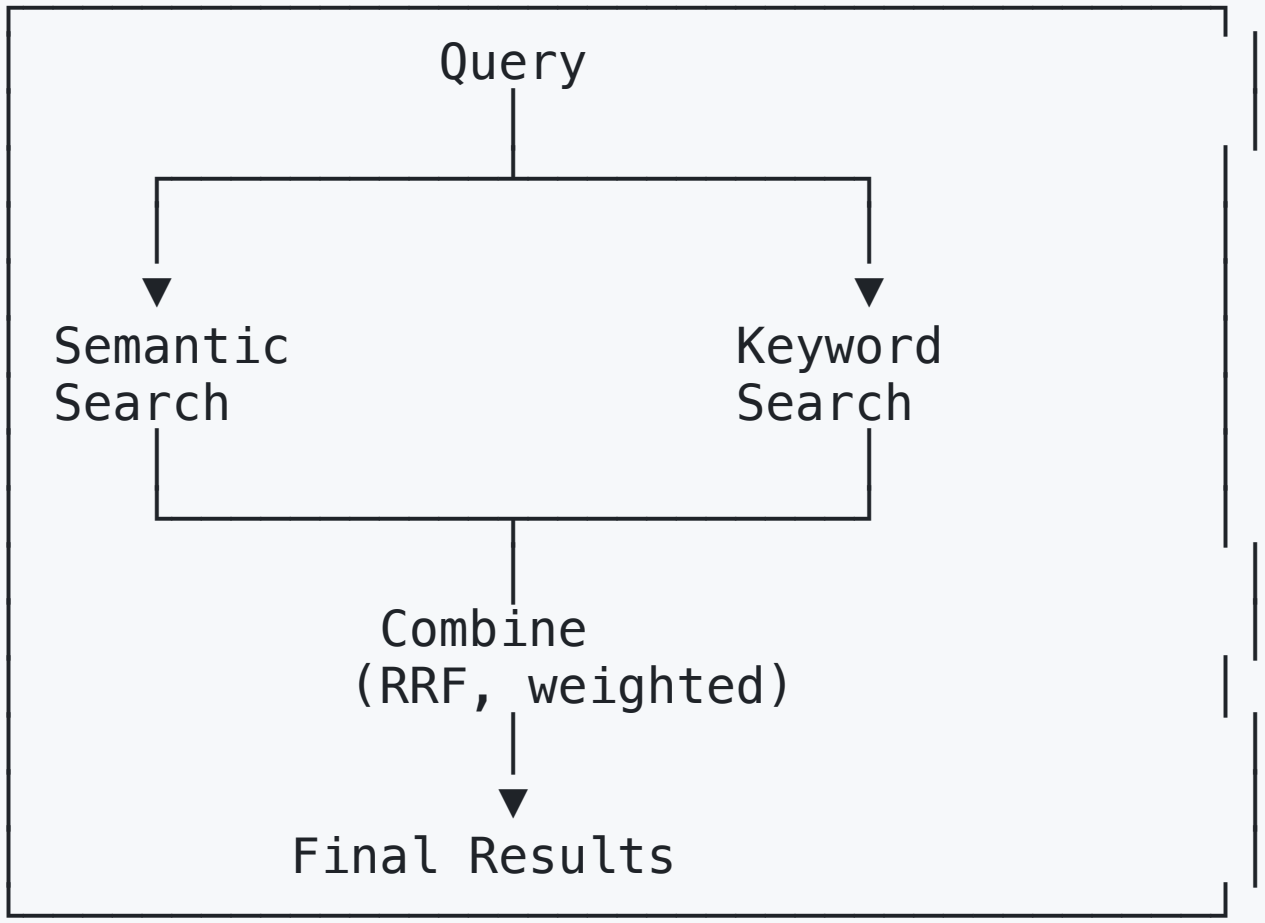


Most common in production (Pinecone, Weaviate, etc.)

Vector Database Options

Database	Type	Best For
Chroma	Embedded	Prototyping, small scale
Pinecone	Managed	Production, no ops
Weaviate	Self-hosted	Full control
pgvector	Postgres ext.	Already use Postgres
Milvus	Self-hosted	Large scale

Hybrid Search



Best of both: Semantic ловит keyword miss, но exact matches

Част 6: Building RAG Pipeline

Step 1: Document Ingestion

```
# Load documents
docs = load_documents("./knowledge_base/")

# Parse different formats
for doc in docs:
    if doc.type == "pdf":
        text = parse_pdf(doc)
    elif doc.type == "docx":
        text = parse_docx(doc)
    # ...

# Clean and normalize
text = clean_text(text)
```

Step 2: Chunking

```
# Simple fixed-size chunking with overlap
def chunk_text(text, chunk_size=500, overlap=50):
    chunks = []
    start = 0
    while start < len(text):
        end = start + chunk_size
        chunk = text[start:end]
        chunks.append(chunk)
        start = end - overlap
    return chunks
```

Step 3: Embedding & Indexing

```
# Generate embeddings
embeddings = embedding_model.encode(chunks)

# Store in vector database
for i, (chunk, embedding) in enumerate(zip(chunks, embeddings)):
    vector_db.insert(
        id=f"doc_{i}",
        embedding=embedding,
        metadata={"text": chunk, "source": doc_name}
    )
```

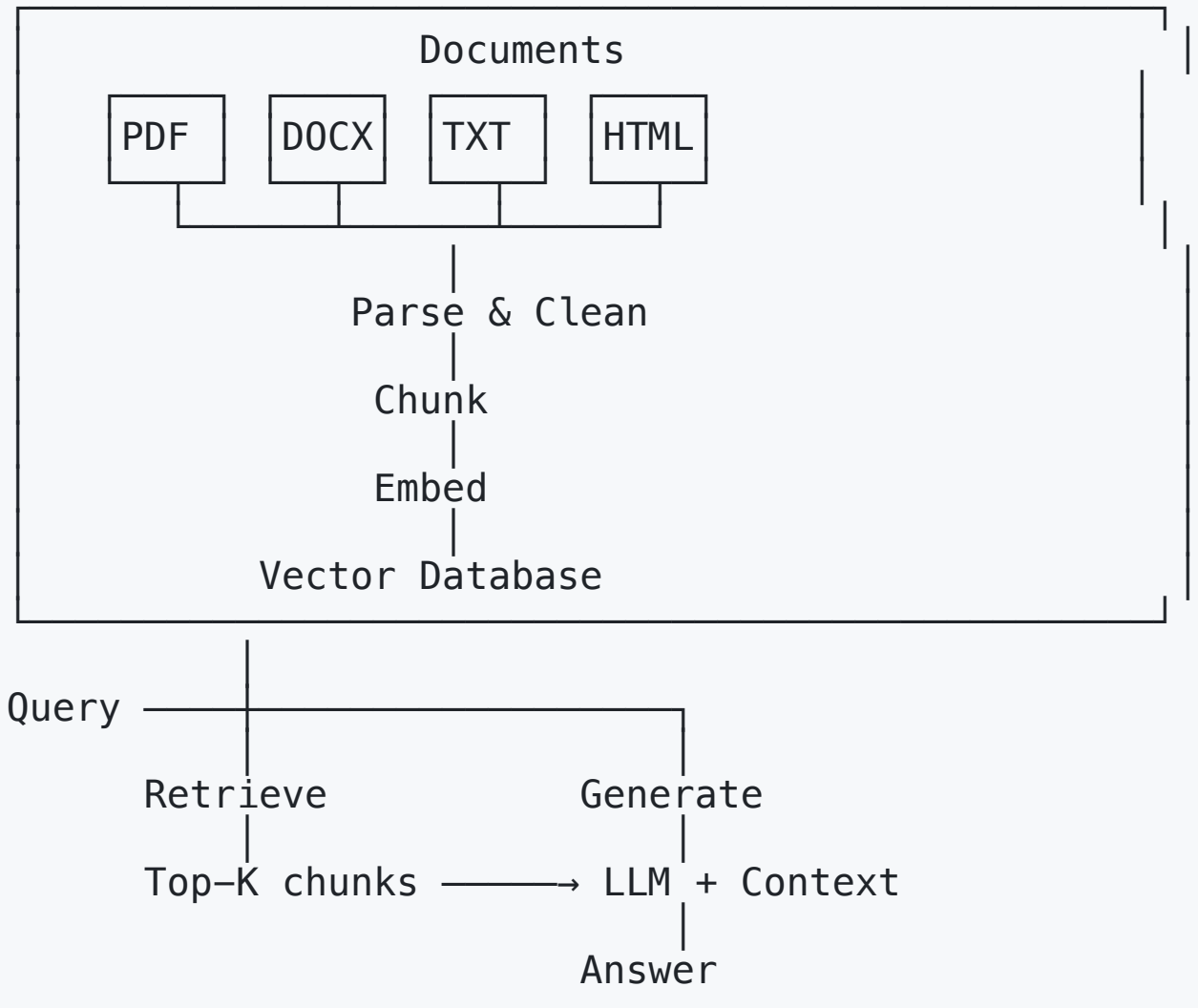
Step 4: Retrieval

```
def retrieve(query, k=3):
    # Embed query
    query_embedding = embedding_model.encode(query)

    # Search vector DB
    results = vector_db.search(
        embedding=query_embedding,
        top_k=k
    )

    return [r.metadata["text"] for r in results]
```

Step 5: Augment & Generate

```
def rag_answer(query):
    # Retrieve relevant chunks
    chunks = retrieve(query, k=3)

    # Build augmented prompt
    context = "\n\n".join(chunks)
    prompt = f"""Based on the following context, answer the question.

Context:
{context}

Question: {query}

Answer:"""

    # Generate
    return llm.generate(prompt)
```

Full Pipeline Visualization



Част 7: Advanced RAG Techniques

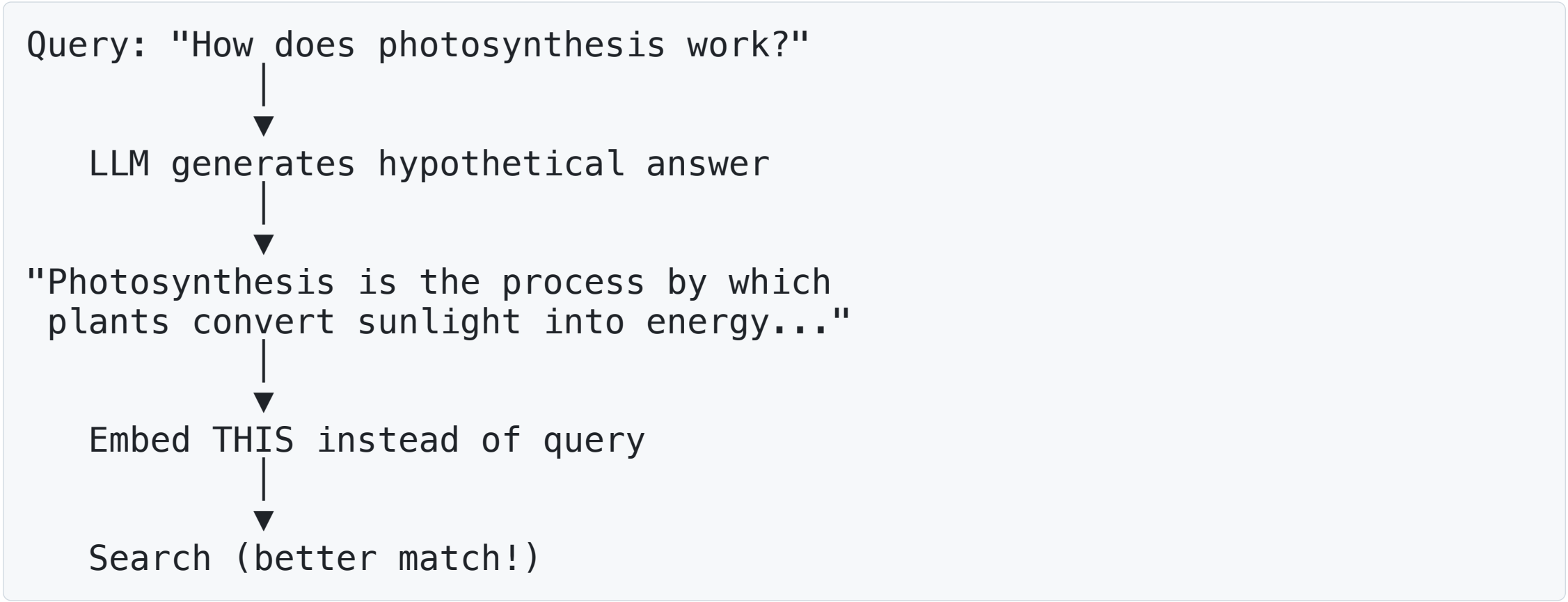
Query Reformulation

Problem: User query може да не match document phrasing

```
Original: "vacation days"
Reformulated: "vacation days, paid time off, PTO, annual leave"
```

Technique: LLM expands query to synonyms/related terms

HyDE (Hypothetical Document Embeddings)

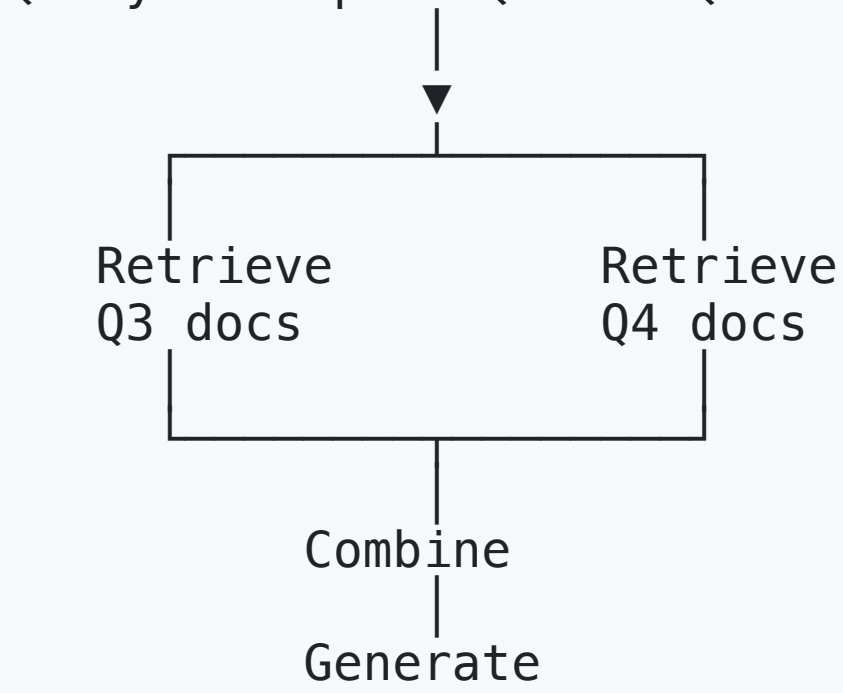


Re-ranking

```
Query → Retrieve top 20 → Re-ranker → Top 5

Initial retrieval: Fast, approximate
Re-ranking: Slow, precise (cross-encoder)
```

```
Retriever (bi-encoder):
  Query embedding vs Doc embedding

Re-ranker (cross-encoder):
  [Query + Doc] → Score
```

Multi-Step Retrieval



Handling Long Retrieved Context

Problem: Много chunks = надхвърляне на context window

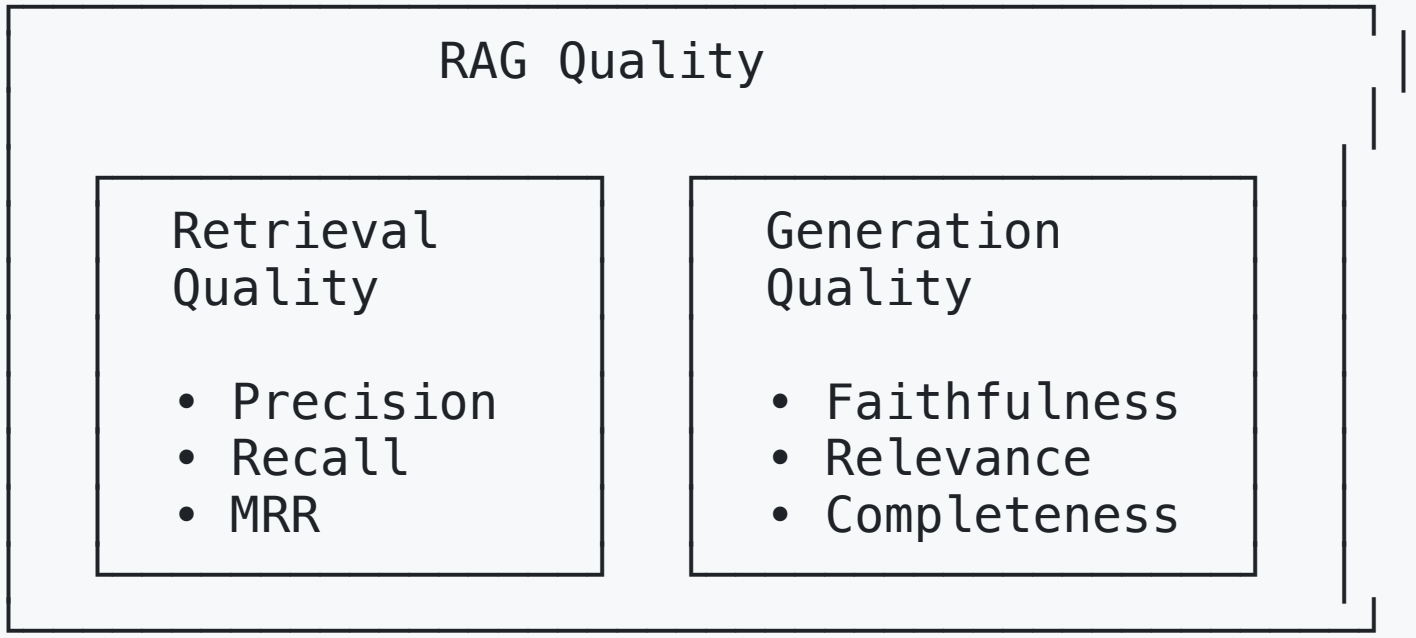
Solutions:

1. **Summarize** chunks преди да подадеш
2. **Map-Reduce:** Process chunks separately, combine
3. **Select best** N chunks по relevance score

```
20 chunks × 500 tokens = 10,000 tokens

Summarize → 3,000 tokens
или
Keep top 5 → 2,500 tokens
```

Част 8: Evaluation & Debugging

RAG Quality Dimensions



Retrieval Metrics

Precision@K: От K retrieved, колко са relevant?

$$\text{Precision@K} = \frac{\text{Relevant in top K}}{K}$$

Recall@K: От всички relevant, колко са retrieved?

$$\text{Recall@K} = \frac{\text{Relevant in top K}}{\text{Total Relevant}}$$

MRR: Средна позиция на първия relevant резултат

Common Failure Modes

Problem	Symptom	Solution
Bad chunking	Cuts mid-sentence	Semantic chunks
Wrong K	Missing info OR noise	Tune K
Embedding mismatch	Good docs not retrieved	Better model
Poor prompting	Ignores context	Better prompt

Debugging Strategy

```
1. Check Retrieval first
   └─ Query → Retrieved chunks
      └─ Are they relevant? If no → tune retrieval

2. Check Generation
   └─ Good chunks → Bad answer?
      └─ Tune prompt, add instructions

3. Check Sources
   └─ Is the info IN the chunks?
      └─ If no → index more docs
```

RAG vs Fine-tuning

RAG	Fine-tuning
Динамично knowledge	Static knowledge
Лесен update	Нужен retrain
Explicit sources	No citations
По-бавно inference	По-бързо inference
No training needed	Нужни данни и compute

Use RAG when: Knowledge changes, citations needed, quick setup

Обобщение

Ключови идеи

1. **Hallucinations** идват от prediction ≠ verification
2. **RAG** grounds модела в external knowledge
3. **Embeddings** capture semantic meaning
4. **Vector databases** enable efficient similarity search
5. **Pipeline:** Ingest → Chunk → Embed → Index → Retrieve → Generate
6. **Quality:** Measure both retrieval AND generation

RAG Decision Framework

```
Нужна актуална информация?      → RAG
Нужни citations?                → RAG
Private/domain data?            → RAG
Knowledge changes often?        → RAG
Static general knowledge only? → Maybe fine-tune
Latency критична?               → Consider caching
```

Следваща лекция

Лекция 12: AI Agents и Tools

- От text generation към actions
- Tool use и function calling
- Agent архитектури
- ReAct, планиране, memory

Ресурси

Papers:

- Lewis et al. (2020) — RAG
- Karpukhin et al. (2020) — Dense Passage Retrieval
- Gao et al. (2023) — HyDE
- Barnett et al. (2024) — Seven Failure Points of RAG

Tools:

- LangChain, LlamaIndex — RAG frameworks
- Chroma, Pinecone — Vector databases

Въпроси?