Loading and splitting code files

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Meri Nova
Machine Learning Engineer

More document loaders...

````
# 🦜️🔗 LangChain

⚡ Build context-aware reasoning applications ⚡

[![Release Notes](https://img.shields.io/github/release/langchain-ai/langchain?style=flat-square)](https://github.com/langchain-ai/langchain/rele
[![CI](https://github.com/langchain-ai/langchain/actions/workflows/check_diffs.yml/badge.svg)](https://github.com/langchain-ai/langchain/actions/
[![PyPI - License](https://img.shields.io/pypi/l/langchain-core?style=flat-square)](https://opensource.org/licenses/MIT)
[![PyPI - Downloads](https://img.shields.io/pypi/dm/langchain-core?style=flat-square)](https://pypistats.org/packages/langchain-core)
[![GitHub star chart](https://img.shields.io/github/stars/langchain-ai/langchain?style=flat-square)](https://star-history.com/#langchain-ai/langc
[![Open Issues](https://img.shields.io/github/issues-raw/langchain-ai/langchain?style=flat-square)](https://github.com/langchain-ai/langchain/iss
[![Open in Dev Containers](https://img.shields.io/static/v1?label=Dev%20Containers&message=Open&color=blue&logo=visualstudiocode&style=flat-squar
remote.remote-containers/cloneInVolume?url=https://github.com/langchain-ai/langchain)
[![Open in GitHub Codespaces](https://github.com/codespaces/badge.svg)](https://codespaces.new/langchain-ai/langchain)
[![Twitter](https://img.shields.io/twitter/url/https/twitter.com/langchainai.svg?style=social&label=Follow%20%40LangChainAI)](https://twitter.com/

Looking for the JS/TS library? Check out [LangChain.js](https://github.com/langchain-ai/langchainjs).

To help you ship LangChain apps to production faster, check out [LangSmith](https://smith.langchain.com).
[LangSmith](https://smith.langchain.com) is a unified developer platform for building, testing, and monitoring LLM applications.
Fill out [this form](https://www.langchain.com/contact-sales) to speak with our sales team.

## Quick Install

With pip:
```bash
pip install langchain
```
````

🦜🔗 LangChain

⚡ Build context-aware reasoning applications ⚡

release langchain-core==0.2.41 | CI passing | license MIT | downloads 19M/month | stars 93k | open issues 598

Dev Containers Open | Open in GitHub Codespaces | Follow @LangChainAI

Looking for the JS/TS library? Check out LangChain.js.

To help you ship LangChain apps to production faster, check out LangSmith. LangSmith is a unified developer platform for building, testing, and monitoring LLM applications. Fill out this form to speak with our sales team.

Quick Install

With pip:

```
pip install langchain
```

Loading Markdown files (.md)

```python
from langchain_community.document_loaders import UnstructuredMarkdownLoader


loader = UnstructuredMarkdownLoader("README.md")
markdown_content = loader.load()
print(markdown_content[0])
```

```
Document(page_content='# Discord Text Classification ![Python Version](https...'
         metadata={'source': 'README.md'})
```

Loading Python files (.py)

```python
from abc import ABC, abstractmethod

class LLM(ABC):
  @abstractmethod
  def complete_sentence(self, prompt):
    pass

...
```

- Integrated into RAG applications for writing or fixing code, creating docs, etc.
- Imports, classes, functions, etc.

```python
from langchain_community.document_loaders \
    import PythonLoader

loader = PythonLoader('chatbot.py')

python_data = loader.load()
print(python_data[0])
```

```
Document(page_content='from abc import ABC, ..

class LLM(ABC):
  @abstractmethod
...',
metadata={'source': 'chatbot.py'})
```

Splitting code files

```
python_splitter = RecursiveCharacterTextSplitter(
    chunk_size=150, chunk_overlap=10
)

chunks = python_splitter.split_documents(python_data)
for i, chunk in enumerate(chunks[:3]):
    print(f"Chunk {i+1}:\n{chunk.page_content}\n")
```

```
Chunk 1:
from abc import ABC, abstractmethod

class LLM(ABC):
  @abstractmethod
  def complete_sentence(self, prompt):
    pass

Chunk 2:
class OpenAI(LLM):
  def complete_sentence(self, prompt):
    return prompt + " ... OpenAI end of sentence."

class Anthropic(LLM):

Chunk 3:
def complete_sentence(self, prompt):
    return prompt + " ... Anthropic end of sentence."
```

Splitting by language

- `separators`
  - `["\n\n", "\n", " ", ""]`
  - `["\nclass ", "\ndef ", "\n\tdef ", "\n\n", " ", ""]`

```
from langchain_text_splitters import RecursiveCharacterTextSplitter, Language

python_splitter = RecursiveCharacterTextSplitter.from_language(
    language=Language.PYTHON, chunk_size=150, chunk_overlap=10
)

chunks = python_splitter.split_documents(data)

for i, chunk in enumerate(chunks[:3]):
    print(f"Chunk {i+1}:\n{chunk.page_content}\n")
```

```
Chunk 1:
from abc import ABC, abstractmethod

Chunk 2:
class LLM(ABC):
  @abstractmethod
  def complete_sentence(self, prompt):
    pass

Chunk 3:
class OpenAI(LLM):
  def complete_sentence(self, prompt):
```

Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Advanced splitting methods

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Meri Nova
Machine Learning Engineer

Limitations of our current splitting strategies

1. Splits are naive (not context-aware) → `SemanticChunker`
    - Ignores context of surrounding text
2. Splits are made using characters vs. **tokens** → `TokenTextSplitter`
    - Tokens are processed by models
    - Risk exceeding the **context window**

Splitting on tokens

How is text split into tokens?

RecursiveCharacter
TextSplitter

chunk_size=5
chunk_overlap=2

['How', 'is', 'text', 'spli', 'lit', 'into', 'toke', 'kens?']

Splitting on tokens

How is text split into tokens?

RecursiveCharacter TextSplitter

['How', 'is', 'text', 'spli', 'lit', 'into', 'toke', 'kens?']

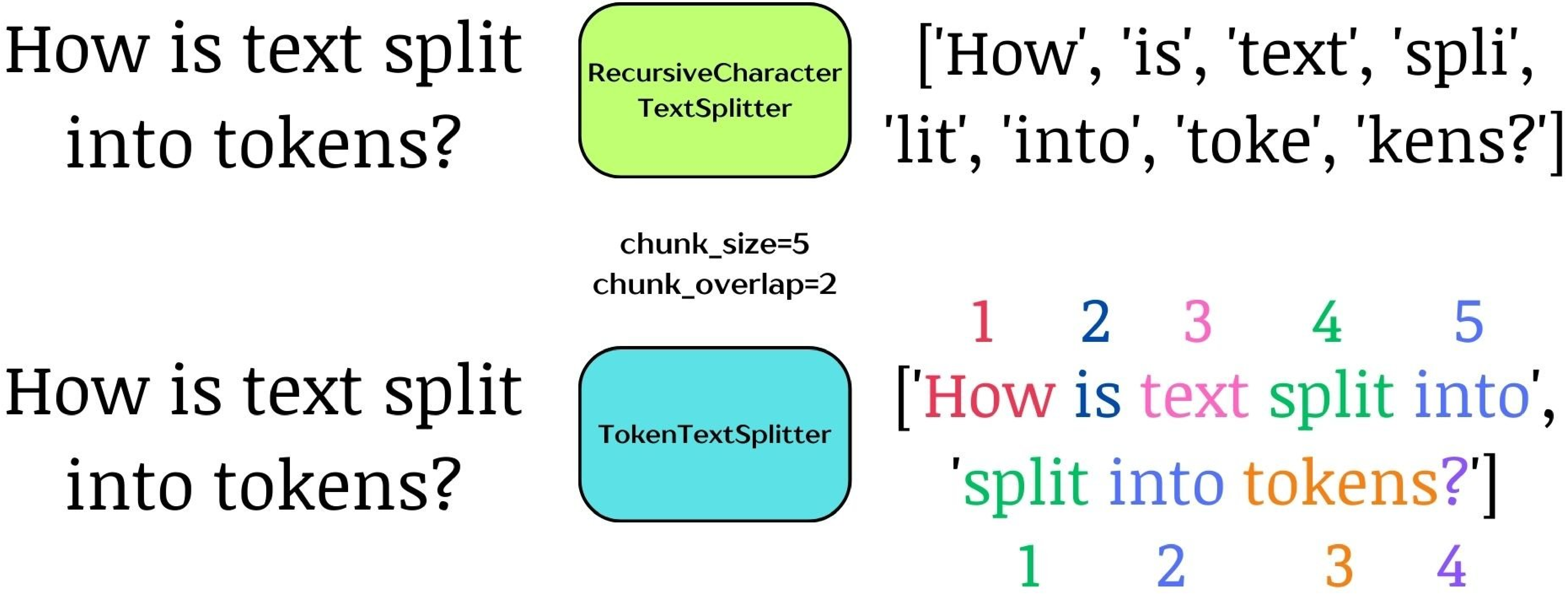
chunk_size=5
chunk_overlap=2

How is text split into tokens?

TokenTextSplitter

['How is text split into', 'split into tokens?']

Splitting on tokens



Splitting on tokens

```python
import tiktoken
from langchain_text_splitters import TokenTextSplitter
example_string = "Mary had a little lamb, it's fleece was white as snow."

encoding = tiktoken.encoding_for_model('gpt-4o-mini')
splitter = TokenTextSplitter(encoding_name=encoding.name,
                             chunk_size=10,
                             chunk_overlap=2)

chunks = splitter.split_text(example_string)

for i, chunk in enumerate(chunks):
    print(f"Chunk {i+1}:\n{chunk}\n")
```

Splitting on tokens

```
Chunk 1:
Mary had a little lamb, it's fleece


Chunk 2:
 fleece was white as snow.
```

Splitting on tokens

```
for i, chunk in enumerate(chunks):
    print(f"Chunk {i+1}:\nNo. tokens: {len(encoding.encode(chunk))}\n{chunk}\n")
```

```
Chunk 1:
No. tokens: 10
Mary had a little lamb, it's fleece was


Chunk 2:
No. tokens: 6
 fleece was white as snow.
```

Semantic splitting

RAG applications has numerous use cases, but are most frequently deployed in chatbots. Domestic dogs are descendants from an extinct population of Pleistocene wolves over 14,000 years ago.

Semantic splitting

RAG applications has numerous use cases,
but are most frequently deployed in chatbots.
Domestic dogs are descendants from an
extinct population of Pleistocene wolves over
14,000 years ago.

Semantic splitting

RAG applications has numerous use cases, but are most frequently deployed in chatbots.

Domestic dogs are descendants from an extinct population of Pleistocene wolves over 14,000 years ago.

Semantic splitting

```
from langchain_openai import OpenAIEmbeddings
from langchain_experimental.text_splitter import SemanticChunker

embeddings = OpenAIEmbeddings(api_key="...", model='text-embedding-3-small')

semantic_splitter = SemanticChunker(
    embeddings=embeddings,
    breakpoint_threshold_type="gradient",
    breakpoint_threshold_amount=0.8
)
```

[1] https://api.python.langchain.com/en/latest/text_splitter/langchain_experimental.text_splitter.SemanticChunker.html

Semantic splitting

```
chunks = semantic_splitter.split_documents(data)
print(chunks[0])
```

```
page_content='Retrieval-Augmented Generation for\nKnowledge-Intensive NLP Tasks\ Patrick Lewis,
Ethan Perez,\nAleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich
Küttler,\nMike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, Douwe Kiela\nFacebook AI
Research; University College London;New York University;\nplewis@fb.com\nAbstract\nLarge
pre-trained language models have been shown to store factual knowledge\nin their parameters,
and achieve state-of-the-art results when ?ne-tuned on down-\nstream NLP tasks. However, their
ability to access and precisely manipulate knowl-\nedge is still limited, and hence on
knowledge-intensive tasks, their performance\nlags behind task-specific architectures.'
metadata={'source': 'rag_paper.pdf', 'page': 0}
```

Let's practice!

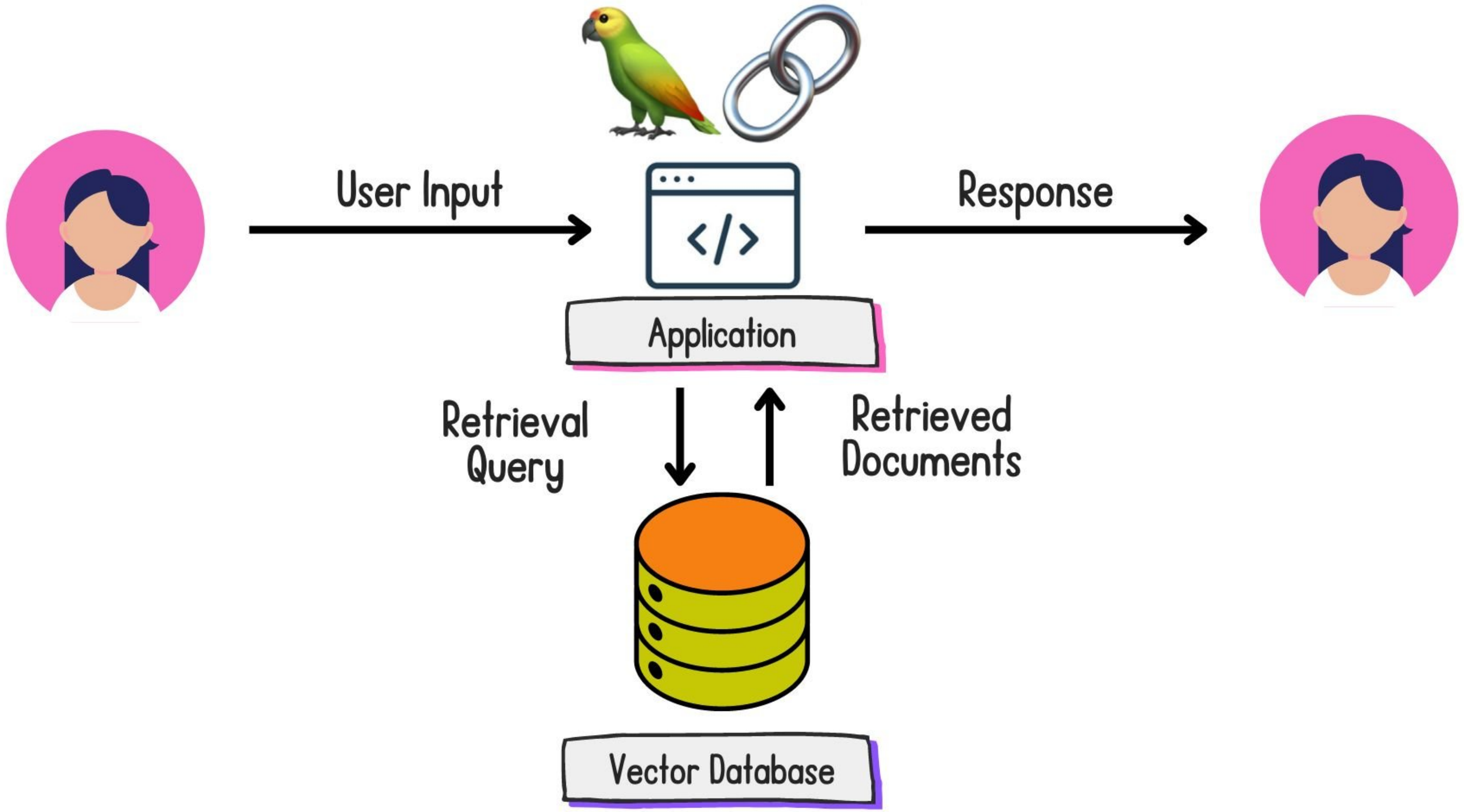
RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Optimizing document retrieval

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

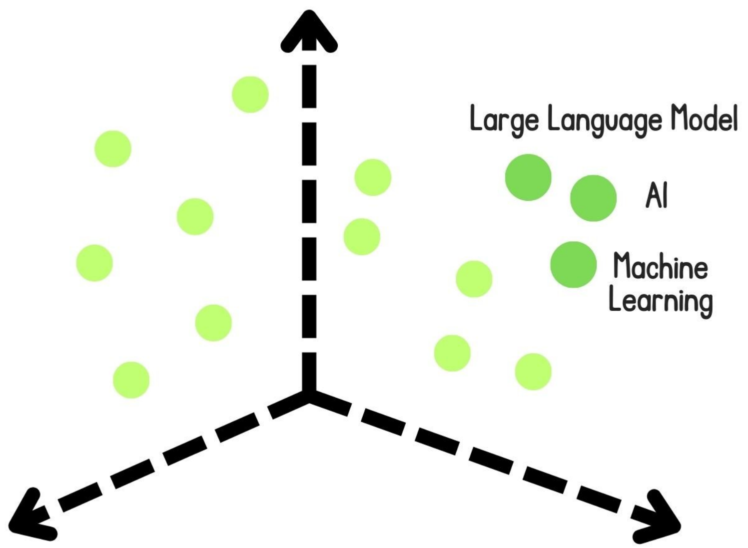
Meri Nova
Machine Learning Engineer

Putting the R in RAG...



Dense

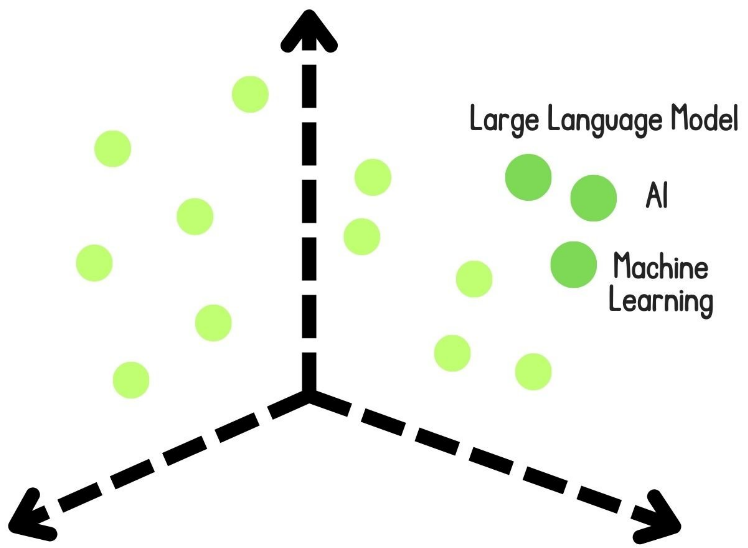
*Encode chunks as a single vector with **non-zero** components*



- **Pros:** Capturing semantic meaning
- **Cons:** Computationally expensive

Dense

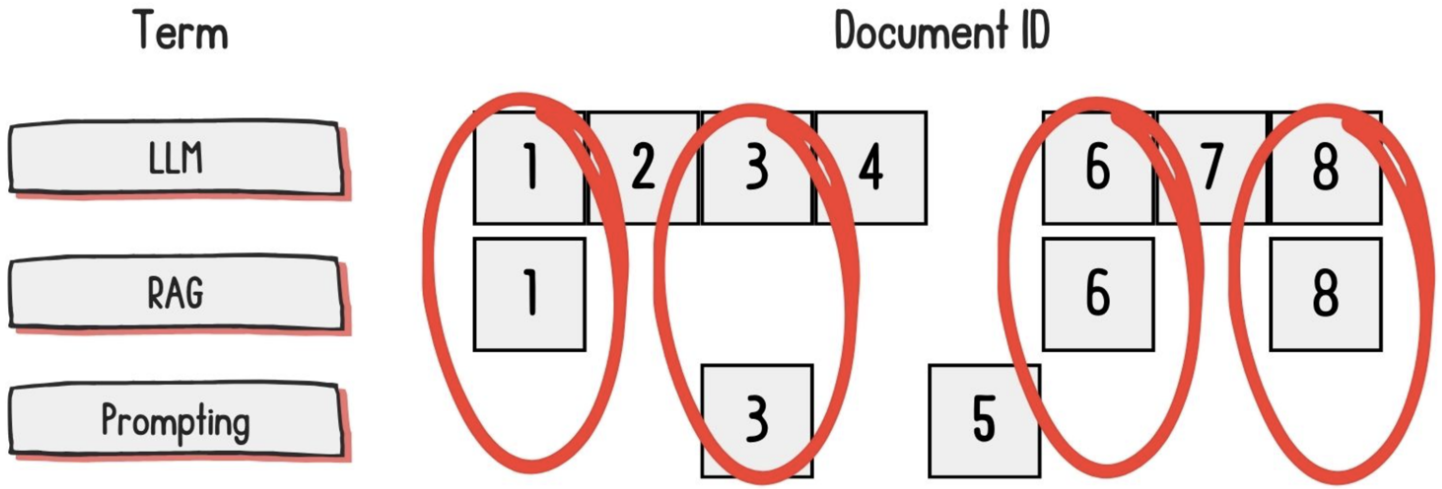
*Encode chunks as a single vector with **non-zero** components*



- **Pros:** Capturing semantic meaning
- **Cons:** Computationally expensive

Sparse

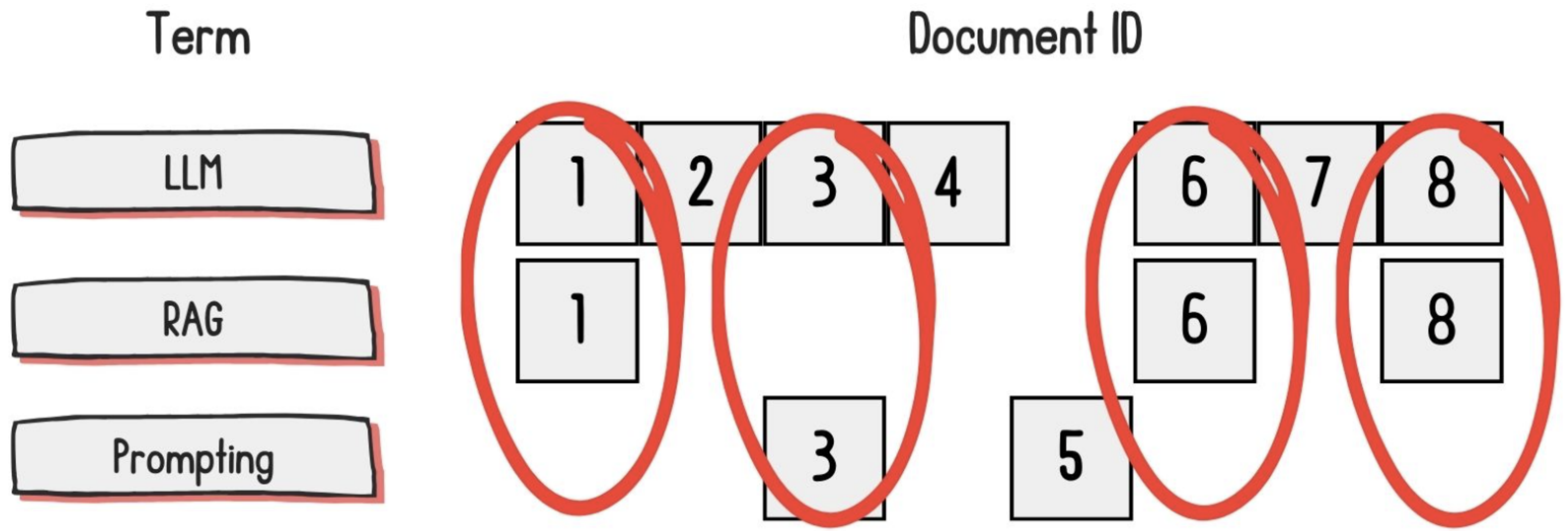
*Encode using **word matching** with mostly **zero** components*



- **Pros:** Precise, explainable, rare-word handling
- **Cons:** Generalizability

Sparse retrieval methods

TF-IDF: Encodes documents using the words that make the document unique



BM25: Helps mitigate high-frequency words from saturating the encoding

BM25 retrieval

```
from langchain_community.retrievers import BM25Retriever

chunks = [
    "Python was created by Guido van Rossum and released in 1991.",
    "Python is a popular language for machine learning (ML).",
    "The PyTorch library is a popular Python library for AI and ML."
]

bm25_retriever = BM25Retriever.from_texts(chunks, k=3)
```

BM25 retrieval

```
results = bm25_retriever.invoke("When was Python created?")
print("Most Relevant Document:")
print(results[0].page_content)
```

```
Most Relevant Document:
Python was created by Guido van Rossum and released in 1991.
```

- Python was created by Guido van Rossum and released in 1991."
- "Python is a popular language for machine learning (ML)."
- "The PyTorch library is a popular Python library for AI/ML."

BM25 in RAG

```
retriever = BM25Retriever.from_documents(
    documents=chunks,
    k=5
)

chain = ({"context": retriever, "question": RunnablePassthrough()}
         | prompt
         | llm
         | StrOutputParser()
)
```

[1] https://www.datacamp.com/blog/what-is-retrieval-augmented-generation-rag

BM25 in RAG

```
print(chain.invoke("How can LLM hallucination impact a RAG application?"))
```

```
The RAG application may generate responses that are off-topic or inaccurate.
```

Let's practice!

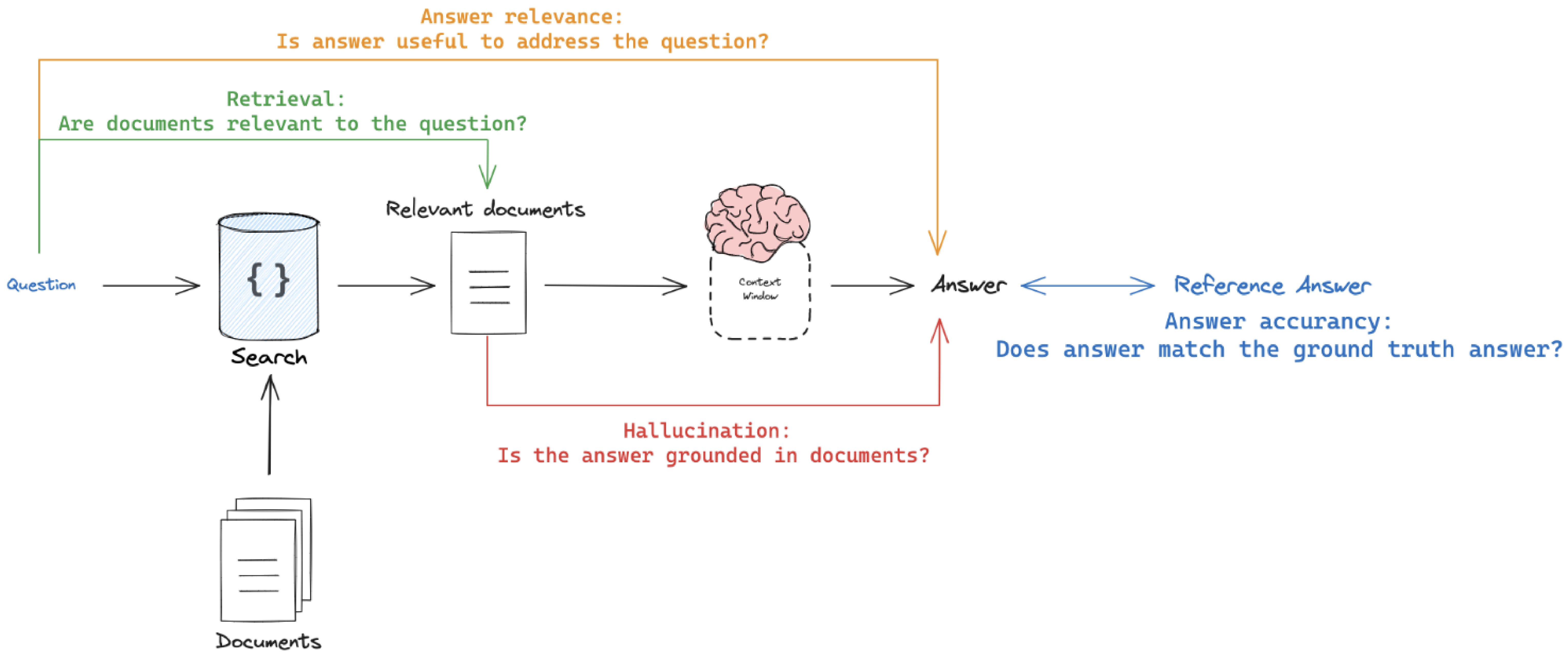
RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Introduction to RAG evaluation

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN

Meri Nova
Machine Learning Engineer

Types of RAG evaluation



[1] Image Credit: LangSmith

Output accuracy: string evaluation

```
query = "What are the main components of RAG architecture?"
predicted_answer = "Training and encoding"
ref_answer = "Retrieval and Generation"
```

Output accuracy: string evaluation

```
prompt_template = """You are an expert professor specialized in grading students' answers to q
You are grading the following question:{query}
Here is the real answer:{answer}
You are grading the following predicted answer:{result}
Respond with CORRECT or INCORRECT:
Grade:"""

prompt = PromptTemplate(
    input_variables=["query", "answer", "result"],
    template=prompt_template
)

eval_llm = ChatOpenAI(temperature=0, model="gpt-4o-mini", openai_api_key='...')
```

Output accuracy: string evaluation

```python
from langsmith.evaluation import LangChainStringEvaluator

qa_evaluator = LangChainStringEvaluator(
    "qa",
    config={
        "llm": eval_llm,
        "prompt": PROMPT
    }
)

score = qa_evaluator.evaluator.evaluate_strings(
    prediction=predicted_answer,
    reference=ref_answer,
    input=query
)
```

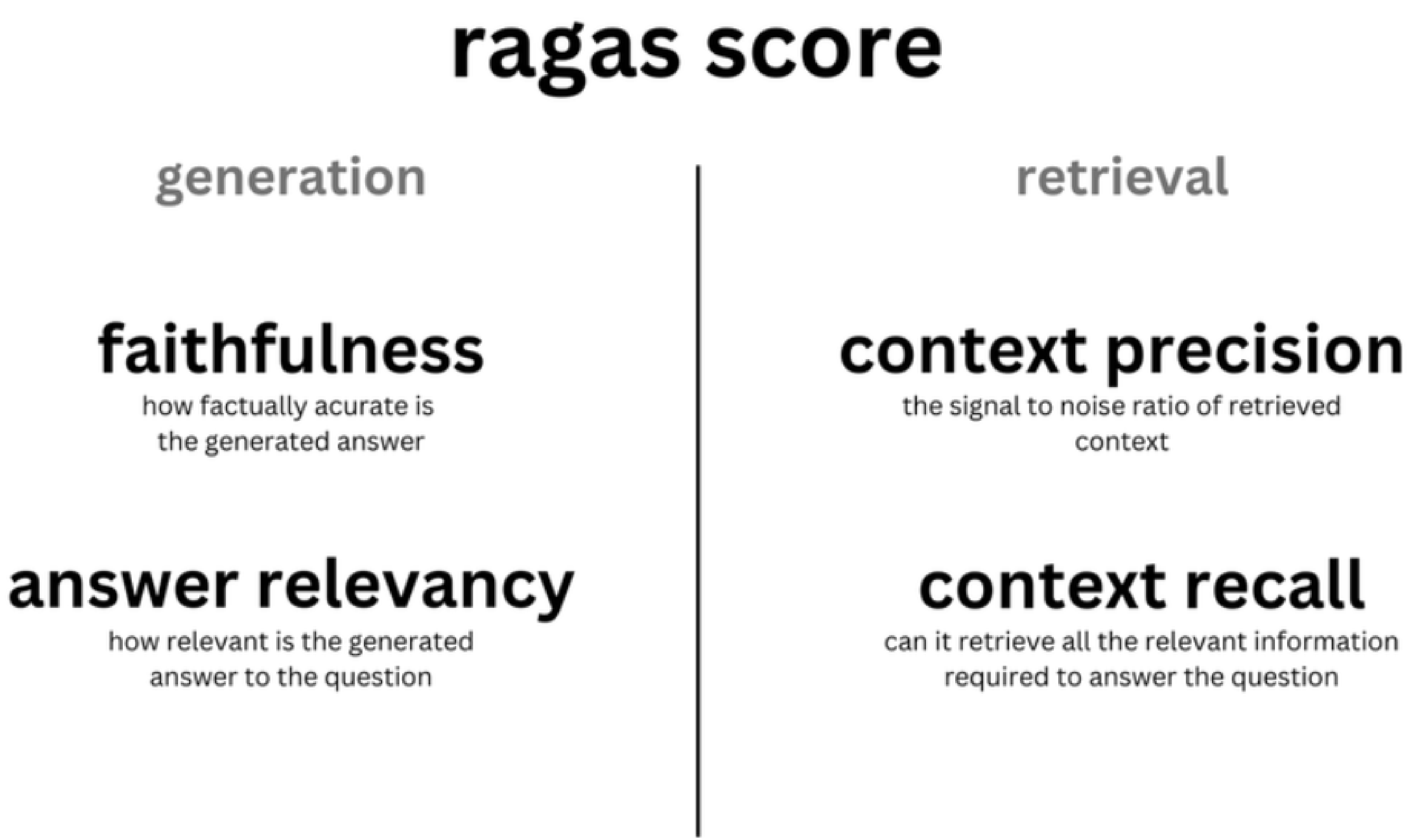
Output accuracy: string evaluation

```
print(f"Score: {score}")
```

```
Score: {'reasoning': 'INCORRECT', 'value': 'INCORRECT', 'score': 0}
```

```
query = "What are the main components of RAG architecture?"
predicted_answer = "Training and encoding"
ref_answer = "Retrieval and Generation"
```

Ragas framework



[1] Image Credit: Ragas

Faithfulness

- *Does the generated output faithfully represent the context?*

$$\text{Faithfulness} = \frac{\text{No. of claims made that can be inferred from the context}}{\text{Total no. of claims}}$$

- **Normalized** to `(0, 1)`

Evaluating faithfulness

```
from langchain_openai import ChatOpenAI, OpenAIEmbeddings
from ragas.integrations.langchain import EvaluatorChain
from ragas.metrics import faithfulness

llm = ChatOpenAI(model="gpt-4o-mini", api_key="...")
embeddings = OpenAIEmbeddings(model="text-embedding-3-small", api_key="...")

faithfulness_chain = EvaluatorChain(
    metric=faithfulness,
    llm=llm,
    embeddings=embeddings
)
```

Evaluating faithfulness

```
eval_result = faithfulness_chain({
  "question": "How does the RAG model improve question answering with LLMs?",
  "answer": "The RAG model improves question answering by combining the retrieval of documents...",
  "contexts": [
    "The RAG model integrates document retrieval with LLMs by first retrieving relevant passages...",
    "By incorporating retrieval mechanisms, RAG leverages external knowledge sources, allowing the...",
  ]
})

print(eval_result)
```

```
'faithfulness': 1.0
```

Context precision

- *How relevant are the retrieved documents to the query?*
- **Normalized** to `(0, 1)` → `1` = highly relevant

```
from ragas.metrics import context_precision

llm = ChatOpenAI(model="gpt-4o-mini", api_key="...")
embeddings = OpenAIEmbeddings(model="text-embedding-3-small", api_key="...")

context_precision_chain = EvaluatorChain(
    metric=context_precision,
    llm=llm,
    embeddings=embeddings
)
```

Evaluating context precision

```
eval_result = context_precision_chain({
  "question": "How does the RAG model improve question answering with large language models?",
  "ground_truth": "The RAG model improves question answering by combining the retrieval of...",
  "contexts": [
    "The RAG model integrates document retrieval with LLMs by first retrieving...",
    "By incorporating retrieval mechanisms, RAG leverages external knowledge sources...",
  ]
})

print(f"Context Precision: {eval_result['context_precision']}")
```

```
Context Precision: 0.99999999995
```

Let's practice!

RETRIEVAL AUGMENTED GENERATION (RAG) WITH LANGCHAIN