



# Retrieval-Augmented Generation for Agroforestry

Code the Change X The Nature Conservancy



# What is RAG?

**RAG** is a way to make Large Language Models (LLMs) even more powerful in answering domain specific questions.

Gives LLM more **contextual information**, which for us will come from agroforestry research papers



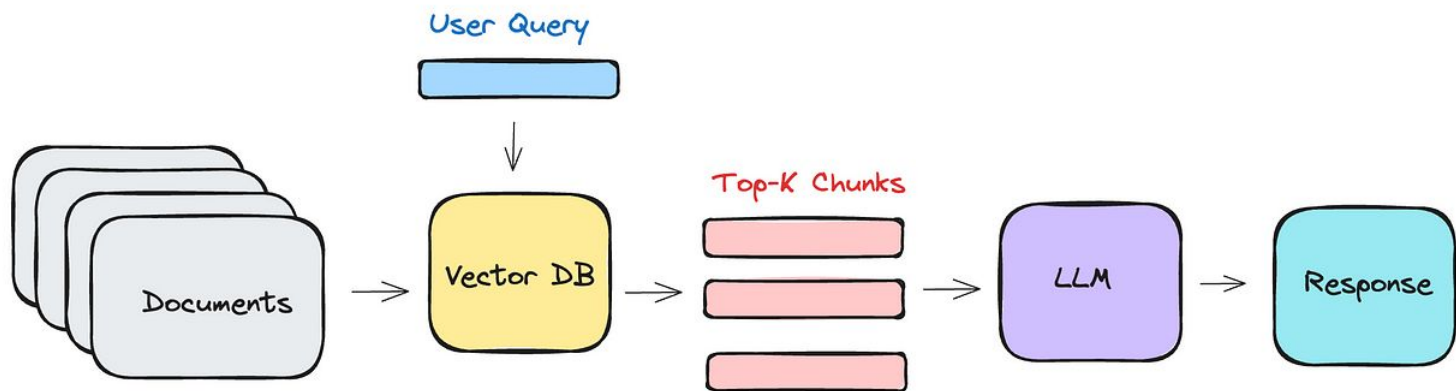


# Our Goals

- 1) Develop a robust RAG system based on the corpus of the annotated articles by TNC scientists specific for the Agroforestry space
- 2) Evaluate and benchmark the RAG system on accuracy and confidence



# Basic RAG Pipeline



Step 1: Data Indexing

Step 2: Data Retrieval & Generation

1. User's question

- Capture User query
- Embed (Vectorise) the query for similarity comparison

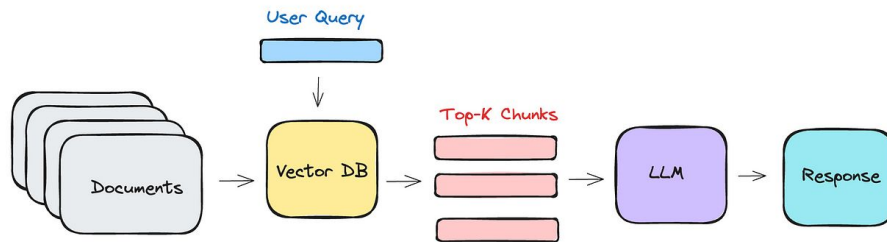
2. Retrieve documents

- Compare the query to the article database and find the most relevant articles

3. Generate response

- Pass the retrieved articles and query into an LLM to generate a response

## Basic RAG Pipeline



Step 1: Data Indexing

Step 2: Data Retrieval & Generation



01

# The User's question



# 1. User Question

Current approach:

Pre processing technique	Methodology
Recognise the <b>key information</b> in the question: Organization, Person, Geopolitical Entity, Location, Product, Nationalities or Religious or Political group, Facility	Python library: <b>spacy</b>
<b>Embed</b> the query as a vector and check for cosine similarity (similarity measure for vectors) with embedded articles	Hugging Face Embedding: "sentence-transformers /all-MiniLM-L6-v2"





02

# Retrieval





## 2. Retrieval

Key questions:

- 1) Given a query, how do we correctly choose the relevant articles?
- 2) Given these articles how do we choose the relevant sections?



# Relevant Article Selection

**Current approach:** Assign a score to each article and choose the highest ranked

- **Filtering:** Remove articles that don't mention all these entities (extracted from Spacy) in the query their text to ensure contextual relevance.
- **Keyword Scoring:** Assign partial scores to articles whose keywords (high frequency in the article) appear in the query.
- **Semantic Similarity:** Compute cosine similarity between the query embedding and each article's LLM generated summary embedding. Add this to the article's score.
- **Top-N Selection:** Rank all articles by total score (keywords + semantic match) and select the top 5



# Relevant Article Selection

## Challenges/Issues:

- **Over-filtering:** Entity checks may exclude relevant articles.
- **Shallow matching:** Keyword overlap misses synonyms.
- **Vector bias:** Summary embeddings can misrepresent full content.



# Relevant Chunk selection

Current approach:

- **Adaptive Chunking:** Tokenized sentences(aka break down words so model can better digest), breaking articles into chunks at most 510 tokens long. Checks sentence length before chunking to prevent cutting sentence off in the middle.
- **Semantic Similarity:** Pass in embedded query and all chunks, use cosine similarity to find top k most semantically similar chunks compared to query




# Relevant Chunk selection

## Challenges/Issues:

- **Inaccurate Chunk Selection:** Even when the correct article is passed in, unable to select the relevant chunks, so LLM end up outputting that no relevant context was given for the query.
- **Deficiencies With Tables:** Chunking cuts off tables, full table sometimes goes over token limit for LLM





03

Generation



# 3. Generation

Key questions:

- 1) How can we pass this information to the LLM in a way which is clear and non-contradictory?
- 2) Which LLM should we use to perform the task?





# Passing information in

Current ideas:

- 1) Make it clear that the generation needn't source multiple articles given in the prompt
- 2) Robust few-shot prompting with varied examples





# Choosing LLMs

## Things to consider

- 1) Token limits/context windows of LLMs
- 2) Accuracy/performance on available benchmarks
- 3) Possibilities to host locally and associated impact





04

# Benchmark



# 4. Benchmark

Key questions:

- 1) How can we measure model accuracy and precisions
- 2) Which factors do we need to prioritize when considering model performance?



# Benchmarking

Current ideas:

- 1) Using “S3 Measurements” tab in spreadsheet to test if model can accurately answer numerical answers
- 2) Using previous general purpose RAG benchmarking datasets



# Benchmarking

Current approach:

- **Question Creation:** Python script automatically created unique question for every row in spreadsheet.
  - question = "What is the {'variable.name'} of planting {'refor.type'} in {'site.sitename'}?"
  - "What is the total soil organic carbon of planting woodlot in Central Kenya?"
- **Semantic Similarity:** Pass in question to model, record semantic similarity score between model answer and actual answer



# Benchmarking

## Challenges/Issues:

- **Long Runtime:** Retrieval stage takes quite a bit to run for every question, and runtime for benchmarking with ~700 questions is around 5-6 hours
- **Units Adjustments:** When model returns a value that is in different units as actual answer, similarity score does not accurately reflect this, which makes our current measured model 50% accuracy to be unreliable





# Questions From CTC Team

- What would be a reasonable final deliverable?
- How should we prioritize our time/which portions of the deliverable is most important?
- Potentially relevant papers that might give advice on RAG process specific to academia?
- Advice for benchmarking? Which factors should we focus on?
- Best practices for presenting our final code repository for future work?
- Possibilities for future work with TNC?





# Thanks!

Do you have any questions?

CREDITS: This presentation template was created by [Slidesgo](#), and includes icons by [Flaticon](#) and infographics & images by [Freepik](#)

