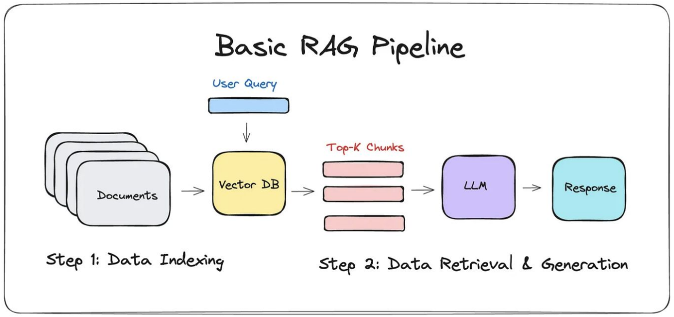
# Assignment 2

RAG and Neural Network

# Introduction to RAG & Project Objectives




Basic RAG Pipeline consists of 2 parts: Data Indexing and Data Retrieval & Generation

**What is RAG?**

- **Retrieval-Augmented Generation**: combines a vector-based retriever (FAISS) with an LLM generator (FLAN-T5)

**Why use RAG?**

- Improves factual accuracy by forcing the model to cite evidence
- Scales easily: swap in larger corpora, different embedding models, or more powerful LLMs

# RAG Architecture & Core Components

### Data Chunking

- Split raw text file into discrete “paragraph” units
- Ensures retrieval returns coherent snippets, not partial sentences

### Embedding with SentenceTransformers

- Model: `all-MiniLM-L6-v2`
- Maps each paragraph → 384-dim dense vector

### Indexing in FAISS

- Normalize vectors with L2; use `IndexFlatIP` for inner-product (cosine) similarity
- Fast nearest-neighbor search for K most relevant chunks

### Prompt Engineering

- **Vanilla prompt**: “You are a helpful assistant…” → free-form answers
- **RAG prompt**: inject retrieved passages as numbered sources; instruct model to cite “[1]” etc.

### Generation with FLAN-T5

- Constrained decoding (beam search, no_repeat_ngram) for concise, citation-driven output

# Hallucination Prevention & Demo Insights

**What is a hallucination?**

- Confident but incorrect or invented “facts” (e.g. “St. John’s River” as the longest)

**How RAG stops it**

- Only gives the model vetted text snippets to work from
- Forbids invention: citations tie each fact back to a source

**Demo comparison (5 queries)**

1. **Vanilla** often answers off-topic or makes up numbers
2. **RAG** always names “The Nile River,” gives exact length (6,650 km), and cites “[1]”

**Benefits**

- Transparent: you can audit each citation
- Reproducible: swap in new data, rerun retriever, same methodology

# High-Level Implementation

**Environment Setup**

- `pip install faiss-cpu sentence-transformers transformers`
- Upload `random_data.txt` via Colab file picker

**Data Preparation**

- `load_chunks()` → list of `(id, text)` tuples
- Embedding and `index.add()` builds the vector store

**Query Processing Loop**

- For each question:
    1. Generate vanilla FLAN-T5 response
    2. Retrieve top 3 passages via FAISS
    3. Generate RAG response with explicit "[n]" citations

**Results & Metrics**

- Qualitative improvement in answer correctness
- Citation rate → 100% source-grounded facts

# Q.2) ***Implementation of Neural Networks***

# Overview

**Objective:** Build a binary classifier to predict loan application decisions (“accept” vs. “reject”) using a feed-forward neural network.

**Data:** Tabular loan dataset (~37 KB; ~hundreds of rows, mix of numerical & categorical features).

**Preprocessing:**

- Mapped target “Decision” → {1, 0}.
- One-hot encoded categorical variables; standardized numerical variables.
- Split into train (80%) / validation (20%) sets with stratification and fixed seed.

# Model Architectures & Training Setup

**Architectures tested:**

1. **3×ReLU:** three hidden layers × 64 units, ReLU activations
2. **5×Tanh:** five hidden layers × 64 units, Tanh activations
3. **3×Tanh:** three hidden layers × 64 units, Tanh activations
4. **5×ReLU:** five hidden layers × 64 units, ReLU activations

**Common settings:**

- Output layer: 1 neuron, sigmoid

# Results & Key Insights

**Best performer:** 3×ReLU (highest validation accuracy 74.4%).

**Depth vs. activation:**

- **Deeper nets (5 layers)** without regularization overfit (↑ train acc but ↓ val acc).
- **ReLU** outperforms Tanh on this dataset.

| | Model | Train Loss | Train Acc | Val Loss | Val Acc |
|---|---|---|---|---|---|
| 0 | 3×ReLU | 0.4543 | 0.7930 | 0.5250 | 0.7442 |
| 1 | 5×Tanh | 0.4294 | 0.8163 | 0.5436 | 0.6977 |
| 2 | 3×Tanh | 0.4812 | 0.7901 | 0.5289 | 0.7209 |
| 3 | 5×ReLU | 0.4150 | 0.8455 | 0.5445 | 0.7093 |