# Assignment 1: Vector Database Creation and Retrieval

## Day 6 Session 2 - RAG Fundamentals

**OBJECTIVE:** Create a vector database from a folder of documents and implement basic retrieval functionality.

**LEARNING GOALS:**

- Understand document loading with SimpleDirectoryReader
- Learn vector store setup with LanceDB
- Implement vector index creation
- Perform semantic search and retrieval

**DATASET:** Use the data folder in `Day_6/session_2/data/` which contains multiple file types

**INSTRUCTIONS:**

1. Complete each function by replacing the TODO comments with actual implementation
2. Run each cell after completing the function to test it
3. The answers can be found in the existing notebooks in the `llamaindex_rag/` folder

```
from google.colab import drive
drive.mount('/content/drive')
```

```
%pip install -q -r '/content/drive/MyDrive/RAGDemo/requirements.txt'
```

```
WARNING: huggingface-hub 1.3.7 does not provide the extra 'inference'
```

```
from google.colab import userdata
import os

os.environ['OPENROUTER_API_KEY'] = userdata.get('OPENROUTER_API_KEY')
print("✅ Initialized ENV Variable successfully!")
```

```
✅ Initialized ENV Variable successfully!
```

```
# Import required libraries
import os
from pathlib import Path
from typing import List
from llama_index.core import SimpleDirectoryReader, VectorStoreIndex, StorageContext, Settings
from llama_index.vector_stores.lancedb import LanceDBVectorStore
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
from google.colab import userdata

print("✅ Libraries imported successfully!")
```

```
✅ Libraries imported successfully!
```

```
# Configure LlamaIndex Settings (Using OpenRouter - No OpenAI API Key needed)
def setup_llamaindex_settings():
    """
    Configure LlamaIndex with local embeddings and OpenRouter for LLM.
    This assignment focuses on vector database operations, so we'll use local embeddings only.
    """
    # Check for OpenRouter API key (for future use, not needed for this basic assignment)
    api_key = os.getenv("OPENROUTER_API_KEY")
    #api_key = userdata.get('OPENROUTER_API_KEY')
    if not api_key:
        print("ℹ️  OPENROUTER_API_KEY not found - that's OK for this assignment!")
        print("   This assignment only uses local embeddings for vector operations.")

    # Configure local embeddings (no API key required)
    Settings.embed_model = HuggingFaceEmbedding(
        model_name="BAAI/bge-small-en-v1.5",
        trust_remote_code=True
    )

    print("✅ LlamaIndex configured with local embeddings")
```

```
    print("   Using BAAI/bge-small-en-v1.5 for document embeddings")

# Setup the configuration
setup_llamaindex_settings()
```

```
modules.json: 100%                                                  349/349 [00:00<00:00, 23.8kB/s]
config_sentence_transformers.json: 100%                             124/124 [00:00<00:00, 9.26kB/s]
README.md:         94.8k/? [00:00<00:00, 5.13MB/s]
sentence_bert_config.json: 100%                                     52.0/52.0 [00:00<00:00, 2.97kB/s]
config.json: 100%                                                   743/743 [00:00<00:00, 67.3kB/s]
model.safetensors: 100%                                             133M/133M [00:01<00:00, 149MB/s]
Loading weights: 100%                                               199/199 [00:00<00:00, 517.90it/s, Materializing param=pooler.dense.weight]
BertModel LOAD REPORT from: BAAI/bge-small-en-v1.5
Key                     | Status     |  |
------------------------+------------+--+-
embeddings.position_ids | UNEXPECTED |  |

Notes:
- UNEXPECTED    :can be ignored when loading from different task/architecture; not ok if you expect identical arch.
tokenizer_config.json: 100%                                         366/366 [00:00<00:00, 13.8kB/s]
vocab.txt:         232k/? [00:00<00:00, 4.82MB/s]
tokenizer.json:         711k/? [00:00<00:00, 15.3MB/s]
special_tokens_map.json: 100%                                       125/125 [00:00<00:00, 6.74kB/s]
config.json: 100%                                                   190/190 [00:00<00:00, 8.66kB/s]
✅ LlamaIndex configured with local embeddings
   Using BAAI/bge-small-en-v1.5 for document embeddings
```

## 1. Document Loading Function

Complete the function below to load documents from a folder using `SimpleDirectoryReader`.

**Note:** This assignment uses local embeddings only - no OpenAI API key required! We're configured to use OpenRouter for future LLM operations.

```
def load_documents_from_folder(folder_path: str):
    """
    Load documents from a folder using SimpleDirectoryReader.

    This implementation uses SimpleDirectoryReader with recursive=True
    so all files under the folder (including nested directories) are loaded.

    Args:
        folder_path (str): Path to the folder containing documents

    Returns:
        List of documents loaded from the folder
    """
    folder = Path(folder_path)
    if not folder.exists():
        print(f"Papers folder does not exist: {folder}")
        return []

    # Use SimpleDirectoryReader to load all files recursively
    try:
        reader = SimpleDirectoryReader(str(folder), recursive=True)
        documents = reader.load_data()
        print(f"✓ Loaded {len(documents)} documents from {folder}")
        return documents
    except Exception as e:
        print(f"Error loading documents from {folder}: {e}")
        return []

# Test the function after you complete it
test_folder = "/content/drive/MyDrive/RAGDemo/papers/agents"
documents = load_documents_from_folder(test_folder)
```

```
print(f"Loaded {len(documents)} documents")
```

```
✓ Loaded 229 documents from /content/drive/MyDrive/RAGDemo/papers/agents
Loaded 229 documents
```

## 2. Vector Store Creation Function

Complete the function below to create a LanceDB vector store.

```
def create_vector_store(db_path: str = "./vectordb", table_name: str = "documents"):
    """
    Create a LanceDB vector store for storing document embeddings.

    TODO: Complete this function to create and configure a LanceDB vector store.
    HINT: Use LanceDBVectorStore with uri and table_name parameters

    Args:
        db_path (str): Path where the vector database will be stored
        table_name (str): Name of the table in the vector database

    Returns:
        LanceDBVectorStore: Configured vector store
    """
    try:
        import lancedb
    except Exception:
        print("lancedb package is not installed. Install with: pip install lancedb")
        return None

    db_path_obj = Path(db_path)
    try:
        db_path_obj.mkdir(parents=True, exist_ok=True)
    except Exception as e:
        print(f"Error creating directory {db_path_obj}: {e}")
        return None

    try:
        # Ensure LanceDB files exist by connecting
        lancedb.connect(str(db_path_obj))
        vector_store = LanceDBVectorStore(uri=str(db_path_obj), table_name=table_name)
        print(f"✓ LanceDB vector store created at: {db_path_obj} (table: {table_name})")
        return vector_store
    except Exception as e:
        print(f"Error creating LanceDB vector store at {db_path_obj}: {e}")
        return None

# Test the function after you complete it
vector_store = create_vector_store("./assignment_vectordb")
print(f"Vector store created: {vector_store is not None}")
```

```
WARNING:llama_index.vector_stores.lancedb.base:Table documents doesn't exist yet. Please add some data to create it.
✓ LanceDB vector store created at: assignment_vectordb (table: documents)
Vector store created: True
```

## 3. Vector Index Creation Function

Complete the function below to create a vector index from documents.

```
def create_vector_index(documents: List, vector_store):
    """
    Create a vector index from documents using the provided vector store.

    TODO: Complete this function to create a VectorStoreIndex from documents.
    HINT: Create StorageContext with vector_store, then use VectorStoreIndex.from_documents()

    Args:
        documents: List of documents to index
        vector_store: LanceDB vector store to use for storage

    Returns:
```

```
        VectorStoreIndex: The created vector index
    """
    if not documents:
        print("No documents provided to index")
        return None

    if not vector_store:
        print("No vector store provided")
        return None

    try:
        # Create storage context with the vector store
        storage_context = StorageContext.from_defaults(vector_store=vector_store)

        # Create the index from documents
        # This will generate embeddings for each document chunk and store in the vector store
        index = VectorStoreIndex.from_documents(documents, storage_context=storage_context, show_progress=True)

        print(f"✓ Vector index created from {len(documents)} documents")
        return index

    except Exception as e:
        print(f"Error creating vector index: {e}")
        return None

# Test the function after you complete it (will only work after previous functions are completed)
if documents and vector_store:
    index = create_vector_index(documents, vector_store)
    print(f"Vector index created: {index is not None}")
else:
    print("Complete previous functions first to test this one")
```

```
Parsing nodes: 100%                                                 229/229 [00:03<00:00, 163.98it/s]
Generating embeddings: 100%                                         406/406 [04:38<00:00,  1.47it/s]
✓ Vector index created from 229 documents
Vector index created: True
```

## 4. Document Search Function

Complete the function below to search for relevant documents using the vector index.

```
def search_documents(index, query: str, top_k: int = 3):
    """
    Search for relevant documents using the vector index.

    TODO: Complete this function to perform semantic search on the index.
    HINT: Use index.as_retriever() with similarity_top_k parameter, then retrieve(query)

    Args:
        index: Vector index to search
        query (str): Search query
        top_k (int): Number of top results to return

    Returns:
        List of retrieved document nodes
    """
    if not index:
        print("No index provided for search")
        return []

    if not query or not query.strip():
        print("Empty query provided")
        return []

    try:
        # Create a retriever from the index with specified top_k
        retriever = index.as_retriever(similarity_top_k=top_k)

        # Retrieve documents for the query
        results = retriever.retrieve(query)

        print(f"✓ Search completed for query: '{query}'")
```

```
        print(f"  Found {len(results)} relevant documents")

        return results

    except Exception as e:
        print(f"Error searching documents: {e}")
        return []

# Test the function after you complete it (will only work after all previous functions are completed)
if 'index' in locals() and index is not None:
    test_query = "What are AI agents?"
    results = search_documents(index, test_query, top_k=2)
    print(f"Found {len(results)} results for query: '{test_query}'")
    for i, result in enumerate(results, 1):
        print(f"Result {i}: {result.text[:100] if hasattr(result, 'text') else 'No text'}...")
else:
    print("Complete all previous functions first to test this one")
```

```
✓ Search completed for query: 'What are AI agents?'
  Found 2 relevant documents
Found 2 results for query: 'What are AI agents?'
Result 1: AI Agents vs. Agentic AI: A Conceptual
Taxonomy, Applications and Challenges
Ranjan Sapkota *‡, Kons...
Result 2: AI Agents
&
Agentic AI
Architecture
Mechanisms
Scope/
Complexity
Interaction
Autonomy
Fig. 2: Mindma...
```

## 5. Final Test - Complete Pipeline

Once you've completed all the functions above, run this cell to test the complete pipeline with multiple search queries.

```
# Final test of the complete pipeline
print("🚀 Testing Complete Vector Database Pipeline")
print("=" * 50)

# Re-run the complete pipeline to ensure everything works
data_folder = "/content/drive/MyDrive/RAGDemo/papers/agents"
vector_db_path = "./assignment_vectordb"

# Step 1: Load documents
print("\n📁 Step 1: Loading documents...")
documents = load_documents_from_folder(data_folder)
print(f"   Loaded {len(documents)} documents")

# Step 2: Create vector store
print("\n🗄️ Step 2: Creating vector store...")
vector_store = create_vector_store(vector_db_path)
print("   Vector store status:", "✅ Created" if vector_store else "❌ Failed")

# Step 3: Create vector index
print("\n🔗 Step 3: Creating vector index...")
if documents and vector_store:
    index = create_vector_index(documents, vector_store)
    print("   Index status:", "✅ Created" if index else "❌ Failed")
else:
    index = None
    print("   ❌ Cannot create index - missing documents or vector store")

# Step 4: Test multiple search queries
print("\n🔍 Step 4: Testing search functionality...")
if index:
    search_queries = [
        "What are AI agents?",
        "How to evaluate agent performance?",
        "Italian recipes and cooking",
        "Financial analysis and investment"
    ]
```

```
    for query in search_queries:
        print(f"\n    🔎 Query: '{query}'")
        results = search_documents(index, query, top_k=2)

        if results:
            for i, result in enumerate(results, 1):
                text_preview = result.text[:100] if hasattr(result, 'text') else "No text available"
                score = f" (Score: {result.score:.4f})" if hasattr(result, 'score') else ""
                print(f"      {i}. {text_preview}...{score}")
        else:
            print("      No results found")
else:
    print("    ❌ Cannot test search - index not created")

print("\n" + "=" * 50)
print("🎯 Assignment Status:")
print(f"   Documents loaded: {'✅' if documents else '❌'}")
print(f"   Vector store created: {'✅' if vector_store else '❌'}")
print(f"   Index created: {'✅' if index else '❌'}")
print(f"   Search working: {'✅' if index else '❌'}")

if documents and vector_store and index:
    print("\n🎉 Congratulations! You've successfully completed the assignment!")
    print("   You've built a complete vector database with search functionality!")
else:
    print("\n📝 Please complete the TODO functions above to finish the assignment.")
```

```
🚀 Testing Complete Vector Database Pipeline
==================================================

📁 Step 1: Loading documents...
✓ Loaded 229 documents from /content/drive/MyDrive/RAGDemo/papers/agents
   Loaded 229 documents

🗄️ Step 2: Creating vector store...
✓ LanceDB vector store created at: assignment_vectordb (table: documents)
   Vector store status: ✅ Created

🔗 Step 3: Creating vector index...
Parsing nodes: 100%                                                 229/229 [00:01<00:00, 243.42it/s]
Generating embeddings: 100%                                         406/406 [04:46<00:00,  1.44it/s]
✓ Vector index created from 229 documents
   Index status: ✅ Created

🔍 Step 4: Testing search functionality...

   🔎 Query: 'What are AI agents?'
✓ Search completed for query: 'What are AI agents?'
  Found 2 relevant documents
      1. AI Agents vs. Agentic AI: A Conceptual
Taxonomy, Applications and Challenges
Ranjan Sapkota *‡, Kons... (Score: 0.6701)
      2. AI Agents vs. Agentic AI: A Conceptual
Taxonomy, Applications and Challenges
```