Assignment 2: Ground the Domain - From Naive RAG to Production Patterns

Start Assignment

- Due Friday by 11:59pm
- Points 100
- Submitting a text entry box, a website url, a media recording, or a file upload

Ground the Domain - From Naive RAG to Production Patterns

1. Assignment Overview

Objective: Develop practical expertise in Retrieval-Augmented Generation (RAG) systems by implementing naive RAG and two production-ready enhancements, then evaluating performance using industry-standard metrics to understand real-world RAG deployment challenges.

Core Learning Goals: By completing this assignment, you will demonstrate proficiency in RAG pipeline architecture, advanced retrieval techniques, systematic evaluation using RAGAs/ARES frameworks, and evidence-based optimization of retrieval-augmented systems for production deployment.

2. Resource Access and Equity

Supported Platforms (Free/Accessible)

- **Large Language Models**: - **Claude.ai** (free tier) - **ChatGPT-3.5** (free tier) - **Google Gemini** (free tier) - **Llama 3.1/3.2** via Ollama (local installation) - **Mistral 7B** via Ollama or Hugging Face - **Qwen 2.5** via Ollama or Hugging Face Transformers - **Flan-T5** (various sizes) via Hugging Face Transformers
- **Embedding Models**: - **sentence-transformers** (all-MiniLM-L6-v2, all-mpnet-base-v2) - **Hugging Face sentence embeddings** (free tier) - **text-embedding-ada-002** (if using free OpenAI credits)
- **Vector Databases**: - **FAISS** (Facebook AI Similarity Search - completely free) - **Milvus Lite** (local, lightweight version) - **Chroma** (open-source vector database) - **Weaviate** (free tier for small datasets)
- **Computational Platforms**: - **Google Colab** (free tier with GPU access) - **Kaggle Notebooks** (free GPU hours) - **Local development** (CPU-based implementations supported)

Alternative Platforms (Optional)

If you have personal subscriptions or free credits, you may use: - **ChatGPT-4o, GPT-4, or GPT-4 Turbo** (OpenAI) - **Claude 3.5 Sonnet or Claude Pro** (Anthropic) - **Gemini Pro or Gemini Advanced** (Google) - **Commercial APIs**: OpenAI, Anthropic, Cohere, or Perplexity Pro - **Cloud Computing**: AWS, Google Cloud, or Azure free/student credits - **Premium Vector Databases**: Pinecone, Qdrant Cloud, or Weaviate Cloud

Equity Policy

- **No Required Purchases**: Students are **NOT required** to obtain premium versions of LLMs, cloud services, or computational resources
- **No Penalties**: Students will **NOT be penalized** for using open-source, open-weight, or free-tier models instead of premium alternatives
- **Equal Assessment**: All submissions will be evaluated based on methodology, analysis quality, and technical understanding—not on the sophistication of paid tools used

Dataset Access

- **Primary Dataset**: RAG Mini Wikipedia (https://huggingface.co/datasets/rag-datasets/rag-mini-wikipedia)

3. Progressive Assignment Structure

Step 1: Domain Documents (Days 1-2)

Deliverable: Dataset Setup and Exploration Report (300 words)

Requirements:

1. **Dataset Access**: Download and explore the RAG Mini Wikipedia dataset (https://huggingface.co/datasets/rag-datasets/rag-mini-wikipedia)

2. **Data Understanding**: Document dataset structure, sample entries, and data quality observations

3. **Infrastructure Planning**: Set up development environment and verify access to chosen platforms

Step 2: Build Naive RAG (Days 3-5)

Deliverable: Functional RAG Pipeline + Documentation (500 words + code)

Technical Requirements:

1. **Embed with sentence-transformers**: Use “all-MiniLM-L6-v2” as recommended

2. **Store embeddings in vector DB**: Use Milvus Lite or FAISS as recommended
3. **Search and generate**: Implement retrieval and response generation

Code Organization: - Modular implementation with clear documentation - Error handling and logging - Configuration files for easy parameter adjustment

Step 3: Evaluation Phase I (Days 6-8)

Deliverable: Initial Evaluation Results (400 words + results)

Requirements:

1. **Generate answers**: Pass **top-1** retrieved document to prompt

2. **Prompting strategies**: Use different approaches (CoT, Persona Prompting, Instruction Prompt, etc.)

3. **Calculate metrics**: F1-score and Exact Match using HuggingFace Squad metric 4.

4. Document best strategy: Note which prompting strategy performed best with hypothesis

Step 4: Experimentation (Days 9-11)

Deliverable: Parameter Comparison Analysis (500 words + results table)

Experimental Requirements:

1. **Embedding sizes**: Generate results using **at least 2 different embedding sizes** (e.g., 256, 384, 512)

2. **Retrieval variations**: Experiment with top 3/5/10 retrieved documents (**at least 2 different passage selection strategies** beyond top-1)

3. **Repeat evaluation**: Apply Step 3 evaluation methodology to all parameter combinations

4. **Documentation**: Submit results as markdown/PDF document in GitHub repository

Step 5: Add Two Advanced RAG Features (Days 12-13)

Deliverable: Enhanced RAG Implementation (400 words + code)

Enhancement Options (from Merwane Merabet list): - Query rewriting - Reranking
- Metadata filtering - Multi-vector retrieval - Confidence scoring - Context window optimization - Grounded citations

Implementation Requirements: - Choose any two enhancements from the list above - Implement both features and integrate with existing pipeline - Rerun test queries on enhanced system

Step 6: Advanced Evaluation with RAGAs or ARES (Days 14-15)

Deliverable: Automated Evaluation Report (500 words + metrics)

Evaluation Requirements: - Use either RAGAs or ARES to assess naive and enhanced RAG pipelines - Report on **at least** the following metrics: - Faithfulness - Context precision/recall
- Answer helpfulness or relevance - Compare performance between naive and enhanced systems

Step 7: Synthesis and Reflection (Days 16-17)

Deliverable: Final Summary Report (1000-1250 words)

Report Structure: - Description of naive system with all required details - Experimentation results from Steps 3 and 4 - Rationale for chosen enhancements
- Results from RAGAs/ARES evaluation - Key lessons, limitations, and potential for further improvement

Additional Requirements: - Professional formatting appropriate for technical audience - Evidence-based conclusions with supporting data - AI usage documentation as per course policy

4. Enhanced Academic Integrity and AI Collaboration Framework

Comprehensive Usage Documentation

Required Logging (consistent with Assignment 1): - **Tool Inventory**: All GenAI tools, models, APIs, and libraries used - **Usage Log**: Specific purposes, input queries, and output utilization - **Code Generation**: Any AI-assisted code with clear attribution - **Content Creation**: AI involvement in documentation, analysis, or reporting

Permitted AI Assistance

Acceptable Uses: - Code debugging and optimization suggestions - Technical documentation assistance - Literature review and reference formatting - Performance analysis and interpretation guidance

Required Attribution Format:

AI Usage Log
- **Tool**: [Name and Version]
- **Purpose**: [Specific task]
- **Input**: [Query or prompt provided]
- **Output Usage**: [How AI response was incorporated]
- **Verification**: [How accuracy was confirmed]

Prohibited Practices

- Using AI to implement core RAG components without understanding
- Submitting AI-generated analysis as original insights
- Using AI to interpret evaluation results without independent verification
- Undocumented collaboration with AI systems

Academic Integrity Verification

Required Evidence: - Personal contribution statements for each major component - Decision rationale documentation showing independent reasoning - Technical troubleshooting logs demonstrating hands-on problem solving - Original analysis connecting results to broader RAG concepts

5. Detailed Technical Specifications

Reproducibility Requirements

Code Organization:

```
assignment2-rag/
├── src/
│   ├── naive_rag.py
│   ├── enhanced_rag.py
│   ├── evaluation.py
│   └── utils.py
├── data/
│   ├── processed/
│   └── evaluation/
├── results/
│   ├── naive_results.json
│   ├── enhanced_results.json
│   └── comparison_analysis.csv
├── notebooks/
│   ├── data_exploration.ipynb
│   ├── system_evaluation.ipynb
│   └── final_analysis.ipynb
├── docs/
│   ├── setup_instructions.md
│   ├── evaluation_report.pdf
│   └── technical_appendix.md
└── requirements.txt
```

Documentation Standards: - Comprehensive README with setup instructions - Inline code comments explaining complex logic - Configuration files with parameter explanations - Error handling and logging throughout

Evaluation Methodology Standards

Minimum Testing Requirements: - **Query Diversity**: Minimum 100 test queries across different complexity levels - **Statistical Significance**: Report confidence intervals where applicable - **Failure Analysis**: Systematic categorization of error types - **Baseline Comparison**: Clear improvement measurements over naive RAG

6. Deliverables and Assessment Structure

Required Submissions

Component	Format	Word Count/Size	Weight	Due Phase
Architecture Document	.md or .pdf	750 words	10%	Phase 1
Naive RAG Implementation	.py/.ipynb + report	500 words + code	25%	Phase 2
Enhanced RAG System	.py/.ipynb + analysis	750 words + code	25%	Phase 3
Evaluation Report	.pdf	1000 words	20%	Phase 4
Technical Report	.pdf	1250-1500 words	15%	Phase 5
Complete GitHub Repository	URL	N/A	5%	Final
Total			**100%**	

Technical Report Structure (Phase 5)

Executive Summary (200 words): Key findings, performance improvements, deployment readiness

System Architecture (300 words): Technical implementation details, design decisions, trade-offs

Experimental Results (400-500 words): Comprehensive performance analysis with statistical support

Enhancement Analysis (300-400 words): Advanced technique effectiveness, implementation challenges

Production Considerations (200-300 words): Scalability, deployment recommendations, limitations

Appendices: Complete AI usage logs, technical specifications, reproducibility instructions

7. Aligned Evaluation Rubric

Excellent (90-100%)

- **Technical Implementation**: Robust, well-documented code with comprehensive error handling
- **Experimental Design**: Systematic approach with appropriate controls and statistical analysis
- **Enhancement Integration**: Sophisticated implementation with measurable performance improvements
- **Evaluation Rigor**: Thorough analysis using multiple metrics with insightful interpretation
- **Professional Communication**: Clear, comprehensive documentation appropriate for technical audience

Proficient (80-89%)

- **Technical Implementation**: Functional code with adequate documentation and basic error handling
- **Experimental Design**: Sound methodology with appropriate comparisons and basic statistical analysis
- **Enhancement Integration**: Competent implementation with demonstrable improvements
- **Evaluation Rigor**: Adequate analysis using required metrics with reasonable interpretation
- **Professional Communication**: Generally clear documentation with minor gaps

Developing (70-79%)

- **Technical Implementation**: Basic functionality with limited documentation or error handling
- **Experimental Design**: Minimal methodology with superficial comparisons
- **Enhancement Integration**: Incomplete or poorly integrated enhancements with unclear impact
- **Evaluation Rigor**: Partial analysis with limited metric usage or interpretation
- **Professional Communication**: Unclear documentation with significant gaps

Needs Improvement (Below 70%)

- **Technical Implementation**: Non-functional or poorly implemented code
- **Experimental Design**: Inadequate methodology without proper controls
- **Enhancement Integration**: Missing or ineffective enhancements
- **Evaluation Rigor**: Minimal or absent evaluation with no meaningful analysis
- **Professional Communication**: Poor documentation hindering reproducibility

8. Timeline and Resources

Recommended Timeline

Estimated Total Time: 18-24 hours over 17 days

Days 1-2: Dataset setup and exploration (2 hours)

Days 3-5: Naive RAG implementation (6 hours)

Days 6-8: Initial evaluation and prompting experiments (3 hours)

Days 9-11: Parameter experimentation across embedding sizes and retrieval counts (4 hours)

Days 12-13: Advanced RAG enhancements implementation (5 hours)

Days 14-15: RAGAs/ARES evaluation (3 hours)

Days 16-17: Final synthesis report and documentation (3 hours)

Technical Resources

Required Tools and Libraries: - Python 3.8+ with virtual environment - Core libraries: sentence-transformers, FAISS/Milvus, transformers - RAGAs or ARES evaluation frameworks - Jupyter notebooks for experimentation and analysis - Git for version control

Recommended Development Setup: - Local Python environment with GPU support (if available) - Google Colab or Kaggle for cloud-based development - VS Code or PyCharm for code development - GitHub for repository hosting and submission

9. Enhanced Tool and Resource Specifications

Recommended Technology Stack

Core Libraries (with fallback options): - **Vector Storage**: Milvus Lite (primary), FAISS (fallback) - **Embeddings**: sentence-transformers (primary), OpenAI embeddings (if available) - **LLMs**: Llama 2/3, Qwen, Flan-T5 (primary), OpenAI GPT-3.5/4 (if available) - **Evaluation**: RAGAs (primary), ARES (alternative), custom metrics

Development Environment: - Python 3.8+ with virtual environment - Jupyter notebooks for experimentation and analysis - Git for version control with meaningful commits - Requirements.txt with pinned versions for reproducibility

Updated Reading List

Required Reading (building on Assignment 1):

1. Merabet, M. "**17 Advanced RAG Techniques to Turn Your RAG App Prototype into a Production-Ready Solution (https://towardsdatascience.com/17-advanced-rag-techniques-to-turn-your-rag-app-prototype-into-a-production-ready-solution-5a048e36cdc8/)** ." *Towards Data Science*, 2024.
2. **RAGAs GitHub Repository and Documentation (https://github.com/explodinggradients/ragas)**
3. **ARES: An Automated Evaluation Framework for RAG (https://github.com/stanford-futuredata/ARES)**

4. **Evaluating RAG Applications with RAGAs** (https://medium.com/data-science/evaluating-rag-applications-with-ragas-81d67b0ee31a) (Medium Article)
5. Alammar & Grootendorst, ***Hands-On LLMs*** (https://learning.oreilly.com/library/view/hands-on-large-language/9781098150952/) (Ch. 6–7)
6. Bouchard & Peters, ***Building LLMs for Production*** (https://learning.oreilly.com/library/view/building-llms-for/9798324731472/) (Ch. 4)

Supplementary Resources: - Vector database comparison guides - Embedding model benchmarks and selection criteria - Production RAG deployment case studies - Evaluation metric interpretation guides

10. Connection to Course Progression

Building on Assignment 1

- **Domain Expertise**: Apply domain knowledge from Assignment 1 to RAG system design
- **Evaluation Skills**: Extend prompt evaluation experience to RAG-specific metrics
- **Systematic Analysis**: Build on comparative methodology skills

Preparing for Assignment 3

- **Technical Foundation**: RAG pipeline serves as base for reasoning enhancement
- **Evaluation Framework**: Metrics and methodology extend to fine-tuning assessment
- **Production Readiness**: Advanced techniques prepare for deployment considerations

Portfolio Integration

- Document technical growth and learning progression
- Build comprehensive case study of LLM application development
- Develop professional portfolio materials for industry applications

Link to the Starter Code for this assignment -
https://github.com/sayakbanerjee1999/Application-of-LLM-Assignment-2-Starter-Code (https://github.com/sayakbanerjee1999/Application-of-LLM-Assignment-2-Starter-Code)