



ENSAKH INTELLIGENT CHATBOT

AI-POWERED EDUCATIONAL ASSISTANT

An Intelligent Assistant Empowering Students with AI and RAG Technology

Produced by
El Abbassi Mouad

Supervised by
Pr.Hafidi Imad

Academic year : 2025 - 2026

INTRODUCTION

03

THEORETICAL FOUNDATION

07

DATA COLLECTION AND PREPARATION

27

MODEL FINE-TUNING AND TRAINING

32

RAG INTEGRATION

38

RLHF

41

CONCLUSION

43

SOMMAIRE

Developing an intelligent chatbot for ENSAKH students using AI and Small Language Models.

PROJECT OVERVIEW & OBJECTIVES

Executive Summary.

This project focuses on designing and deploying a specialized conversational AI assistant specifically for the National School of Applied Sciences of Khouribga (ENSAKH). By leveraging cutting-edge Small Language Model (SLM) technology, the chatbot aims to provide accurate, contextually relevant information about the institution to students, faculty

Project Context

ENSAKH, like many educational institutions, faces challenges in efficiently disseminating information across its community. Students and stakeholders frequently seek answers about academic programs, administrative procedures, campus facilities, admission requirements, and institutional policies. Traditional information retrieval methods such as navigating complex websites or waiting for administrative responses can be time consuming and inefficient.

PROJECT OVERVIEW & OBJECTIVES

Core Objectives

Primary Objective :

Develop an intelligent, domain-specific chatbot capable of understanding and responding to queries about ENSAKH with high accuracy and relevance, while maintaining computational efficiency through the use of optimized small language models.

Specific Technical Goals :

- Model Optimization: Implement distillation and quantization techniques to create a lightweight yet powerful language model suitable for deployment with limited computational resources
- Domain Specialization: Fine-tune the base model on ENSAKH-specific data to ensure responses are institutionally accurate and contextually appropriate
- Knowledge Enhancement: Integrate Retrieval-Augmented Generation (RAG) to provide up-to-date, factually grounded responses drawn directly from ENSAKH's official information sources
- Continuous Improvement: Implement Reinforcement Learning from Human Feedback (RLHF) to refine model performance based on real-world user interactions

Expected Outcomes :

- A fully functional, domain-adapted chatbot deployed and accessible to the ENSAKH community
- Comprehensive documentation of the development pipeline from data collection through RLHF implementation
- Performance demonstrating improvements in response accuracy and user satisfaction

INTRODUCTION TO SMALL LANGUAGE (SLM)

What is SLM ?

Small Language Models (SLMs) represent a new, more efficient, and specialized class of artificial intelligence designed to deliver high-performance natural language understanding and generation within focused domains, while requiring significantly fewer computational resources than their large-scale counterparts.

1. Targeted Scale

- SLMs are characterized by a compact architecture with parameter counts typically ranging from 1 billion to 15 billion parameters - a fraction of the size of large language models. This reduced scale enables deployment on standard hardware without sacrificing performance for specific tasks.

2. Specialist Training

- Unlike generalist models trained on broad internet-scale datasets, SLMs are:
- Trained on curated, domain-specific datasets tailored to particular industries or use cases
- Easily fine-tunable with proprietary organizational data
- Optimized for deep expertise in narrow domains rather than shallow knowledge across all topics

3. Targeted Capabilities

- SLMs excel at lightweight, task-specific applications including:
- Specialized chatbots for institutional or enterprise environments
- On-device mobile assistants with privacy-first design
- Domain-specific analysis and decision support systems
- Real-time language processing for resource-constrained environments

Strategic Positioning:

SLMs represent a paradigm shift from "bigger is better" to "specialized is superior" - proving that for well-defined tasks, a focused, efficient model trained on relevant data outperforms massive generalist models while consuming a fraction of the resources, cost, and energy.

THEORETICAL FOUNDATION

LARGE LANGUAGE MODEL (LLM)

LARGE LANGUAGE MODEL

Definition :

A Large Language Model (LLM) is a type of artificial intelligence (AI) system that can understand, generate, and interact using human language.

It's called "large" because it is trained on massive amounts of text data (from books, websites, articles, code, etc.) and contains billions or even trillions of parameters the internal "connections" that help it learn patterns in language.

How It Works :

We can think of an LLM as a super-advanced text predictor.

It doesn't truly "think," but it learns the structure, meaning, and context of words by analyzing huge datasets.

When you ask it a question or give it a prompt, it predicts the most likely next words but thanks to its scale and training, these predictions sound intelligent and context-aware.

Examples :

OpenAI : GPT-4 / GPT-5, Meta : LLaMA 3, Google : Gemini, Anthropic : Claude, etc

Why They're Important :

LLMs are transforming how we interact with computers. They are used in:

- Chatbots (like ChatGPT, Copilot, Claude)
- Translation systems
- Search engines
- Education assistants
- Code generation tools
- Summarization and writing software

They make it easier for people to communicate, learn, and build with technology using natural language instead of code.

**LLM
VS
SLM**

Critical Distinctions

Aspect	Large Language Model	Small Language Model
Parameter Scale	100B+ to trillions of parameters	1B - 15B parameters
Training Data	Massive, internet-scale datasets	Curated, domain-specific datasets
Knowledge Scope	Broad, generalist "knows a little about everything"	Deep, specialist "expert in specific domain"
Computational Requirements	Requires massive infrastructure, multiple high-end GPUs/TPUs	Runs on single GPU or standard hardware
Deployment	Cloud-based, third-party APIs	On-premise, edge devices, mobile
Data Privacy	High Risk : Queries sent to external servers	Secure: All data remains on local infrastructure
Best Use Case	Complex, multi-domain creative tasks	Task-specific, institutional applications

LLM : The "Generalist" → Big Brain / **SLM** : The "Specialist" → Focused Expert

FROM LLM TO SLM: THE KNOWLEDGE TRANSFER

The Problem :

- LLMs have powerful capabilities but are too large and expensive for practical deployment
- SLMs are efficient but start with less inherent knowledge
- Question: How do we get LLM-level intelligence into an SLM-sized model?

DISTILLATION

The Solution : Knowledge Distillation

"If an LLM is a master professor and an SLM is a promising student, distillation is the teaching process that transfers expertise from teacher to student."

WHAT IS DISTILLATION?

Knowledge Distillation is a machine learning technique that transfers knowledge from a large, complex "teacher" model to a smaller, more efficient "student" model.

The Concept:

- Teacher Model: Large, powerful LLM with extensive knowledge
- Student Model: Smaller SLM learning to replicate teacher's behavior
- Process: Student learns not just from raw data, but from teacher's "thought process"

Why Distillation Matters for This Project:

1. Preserves Intelligence: Student SLM inherits teacher LLM's reasoning patterns
2. Maintains Performance: Achieves 95%+ of teacher's accuracy at 10% of the size
3. Enables Efficiency: Creates deployment-ready models for ENSAKH
4. Cost-Effective Training: Cheaper than training an SLM from scratch

The Distillation Advantage:

Instead of training our ENSAKH chatbot from zero, we can:

1. Start with a pre-trained LLM (Llama 3.1 70B) as our teacher
2. Use distillation to compress its knowledge into our 8B student model
3. Further fine-tune on ENSAKH-specific data
4. Result: Enterprise-grade intelligence in a deployable package

FINE-TUNING

THE JOURNEY CONTINUES

After distillation, we have a compressed 8B model that inherited the teacher LLM's general intelligence and reasoning patterns.

The Gap : Our distilled model understands language well, but it doesn't know about:

- ENSAKH's specific programs, courses, and schedules
- Administrative procedures unique to the institution
- Faculty information and department structures
- Campus-specific terminology and processes

The Solution : Fine-tuning bridges this gap by teaching our distilled model the specialized knowledge it needs to serve ENSAKH effectively.

Analogy :

- Distillation = Teaching a student how to think like an expert
- Fine-tuning = Teaching that student the specific subject matter they'll work with

WHAT IS FINE-TUNING?

Fine-tuning is the process of taking a pre-trained model and continuing its training on a specific, specialized dataset to adapt for a particular task or domain.

The Concept : Instead of training a model from scratch, we start with an existing model (in our case, the distilled 8B model) and provide it with targeted examples that teach it domain-specific knowledge and behaviors.

Why Fine-Tuning Matters

Without Fine-tuning: Model : "I understand language well, but I don't know your institution."

With Fine-tuning: Model : "I understand language AND I know ENSAKH's programs, schedules, policies, and can answer student questions accurately."

The Power of Fine-Tuning:

- 1.Domain Adaptation: Transforms a general model into a specialist
- 2.Behavior Shaping: Teaches the model how to respond in your desired style
- 3.Knowledge Injection: Embeds institutional-specific information
- 4.Cost-Effective: Much cheaper than training from scratch
- 5.Quick Results: Achieves specialization in hours/days vs months

FINE-TUNING APPROACHES

FINE-TUNING

- 1) Full Fine-Tuning (Traditional) : Update all parameters of the model during training.
 - Best For: When you need maximum performance and have computational resources
- 2) Parameter-Efficient Fine-Tuning (PEFT) : Update only a small subset of parameters while keeping most of the model frozen

PEFT TECHNIQUES			
LoRA (Low-Rank Adaptation)	QLoRA (Quantized LoRA)	Prefix Tuning	Adapter Layers
<i>Instead of modifying the original parameters, LoRA adds small "adapter" matrices alongside them.</i> • Original model weights stay frozen • Small trainable matrices (rank decomposition) are added • Only these tiny matrices are updated during training • At inference, adapter outputs are added to frozen model outputs	Combines LoRA with model quantization for even more efficiency. • Loads the base model in 4-bit precision (instead of 16-bit) • Applies LoRA adapters on top • Reduces memory usage by 75%	Keeps the entire model frozen and only trains special "prefix" tokens that guide the model's behavior • Adds trainable continuous vectors at the input layer • These prefixes act as "soft prompts" that condition the model • Model learns to interpret these prefixes as instructions	Insert small trainable "adapter" modules between the frozen layers of the model • Original transformer layers remain frozen • Small bottleneck layers inserted after each transformer block • Only these adapters are trained • Information passes through: Original Layer → Adapter → Next Layer

FINE-TUNING

Aspect	Full Fine-Tuning	LoRA	QLoRA	Prefix Tuning	Adapters
Parameters Trained	100%	0.1-1%	0.1-1%	< 0.1%	2 - 5%
Memory Usage	Very High	Medium	Low	Very Low	Medium
Training Speed	Slow	Fast	Fast	Very Fast	Fast
Performance	Best	Excellent	Excellent	Good	Very Good
Hardware Needs	Multiple GPUs	1 GPU	1 GPU	1 GPU	1 GPU
Risk of Forgetting	High	Low	Low	Very Low	Low

THE NEXT CHALLENGE

A fine-tuned model that knows ENSAKH well and can answer questions intelligently.

The Problem:

- Models have context limits (can only process just a numbers of tokens at once)
- Cannot fit entire document libraries into every query
- Need to access long-term knowledge beyond what fits in model's context window
- Information updates frequently (new courses, schedule changes, policy updates)

The Question :

How do we give our chatbot access to vast, updatable knowledge bases without overwhelming its context window?

The Solution :

Vector Databases with Retrieval-Augmented Generation (RAG)

VECTOR DATABASE

VECTOR DATABASE :

Definition :

A vector database is a specialized database designed to store, index, and efficiently search high-dimensional vector embeddings that represent the semantic meaning of data.

Traditional Database vs Vector Database:

Traditional Database :

- Stores exact text: "The admission deadline is March 15"
- Searches by exact matching keywords
- Query: "admission deadline" → Finds only exact phrase matches

Vector Database :

- Stores meaning as numbers: [0.2, -0.5, 0.8, ... 1,536 dimensions]
- Searches by semantic similarity
- Query: "When can I apply?" → Finds "admission deadline" (similar meaning, different words)

Semantic Search

Vector databases understand meaning, not just words. They can find relevant information even when :

- Different words are used (synonyms)
- Questions are phrased differently
- Concepts are related but not identical

HOW VECTOR DATABASES WORK

Step 1 : Embedding Creation

What are Embeddings ? Embeddings are numerical representations (vectors) that capture the semantic meaning of text in high-dimensional space.

Example :

Text: "ENSAKH offers computer science programs"

Embedding: [0.23, -0.45, 0.67, 0.12, ..., 0.89] (1,536 numbers)

Texts with similar meanings have embeddings that are close together in vector space.

VECTOR DATABASE

Step 2: Document Processing & Storage

Chunk Documents : Break large documents into smaller chunks (500-1000 tokens each)

- Why? Models have context limits; smaller chunks are more focused

Generate Embeddings : Convert each chunk into a vector using an embedding model

- Popular models : sentence-transformers, etc

Store in Vector DB : Save vectors with metadata

- Vector : [0.23, -0.45, ...]
- Metadata : {source: "student_handbook.pdf", page: 15, topic: "admissions"}

Step 3: Query-Time Retrieval

Embed the Query : Convert user question to vector

- Question : "What are the engineering admission requirements ?"
- Query Vector : [0.21, -0.43, 0.69, ...]

Similarity Search : Find most similar document chunks

- Compare query vector to all stored vectors
- Use distance metrics (cosine similarity, Euclidean distance)
- Return top K most similar chunks (K = 3-5 typically)

Augment the Prompt : Combine retrieved context with user query

Generate Answer : Send to fine-tuned model

Where We Are Now

We've Established :

- Vector databases store embeddings (semantic representations of text)
- They enable fast similarity search across millions of documents
- Multiple options exist: Pinecone, Qdrant, Weaviate, pgvector, etc.

The Question : "Which vector database should we actually implement for ENSAKH?"

We need a vector database that is :

- Easy to set up and use (minimal configuration)
- Open-source and free (university budget constraints)
- Python-friendly (integrates with our ML pipeline)
- Powerful enough for production (handles thousands of documents)
- Actively maintained (regular updates and community support)

The Solution : ChromaDB

- ChromaDB bridges the gap between theoretical vector database concepts and practical implementation. It's the "getting started" database that's powerful enough for production.

CHROMADB

CHROMADB

Definition :

ChromaDB is an open-source, AI-native vector database designed to make it easy to build applications with embeddings. It's the vector database that "just works."

"If PostgreSQL is for structured data, ChromaDB is for AI embeddings."

Why ChromaDB ?

- Install in one command: `pip install chromadb`
- Start using in 3 lines of code
- No complex configuration files
- No separate server setup (can run embedded)
- Built specifically for AI/ML workflows

Built-in Intelligence

Automatic Embeddings :

- Uses default embedding function (no setup required)
- Or bring your own embedding model
- Supports OpenAI, Sentence Transformers, etc.

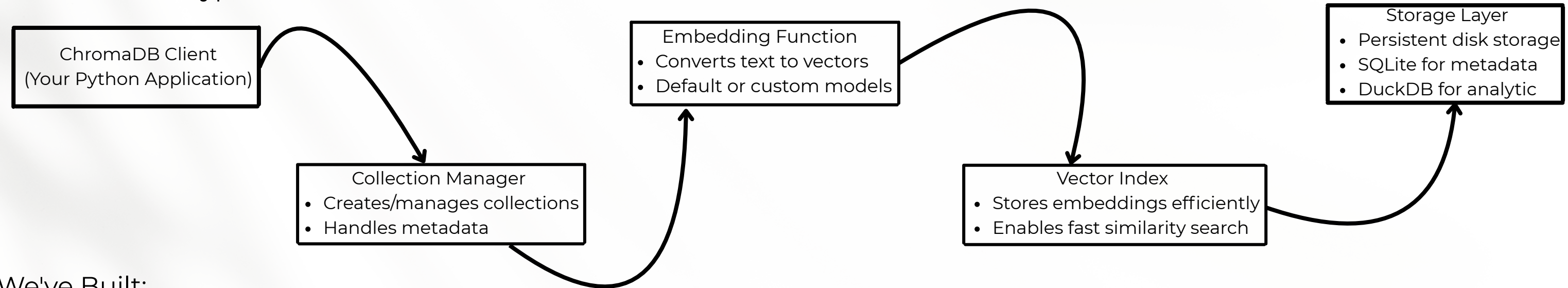
CHROMADB

Smart Filtering :

- Filter by metadata (date, category, source)
- Combine vector search with traditional filters
- Boolean operations (AND, OR, NOT)

Multi-modal Support :

- Text embeddings
- Mixed data types in same collection



We've Built:

- A ChromaDB vector database storing ENSAKH documents
- Documents are embedded and indexed for fast retrieval
- We can search semantically and filter by metadata

The Problem : We have a powerful search engine, but it only returns chunks of text. It doesn't:

- Generate natural conversational answers
- Synthesize information from multiple chunks
- Understand context and nuance
- Provide human-like explanations

Example :

- User asks: "What courses should I take in my third semester for GI?"
- ChromaDB returns: 3 separate text chunks about course sequences, prerequisites, and semester planning
- What we need: A coherent, conversational answer that synthesizes all three chunks

The Solution : RAG (Retrieval-Augmented Generation)

RAG bridges ChromaDB (retrieval) with our fine-tuned LLM (generation) to create intelligent, context-aware responses.

RETRIEVAL-AUGMENTED GENERATION

Definition :

RAG (Retrieval-Augmented Generation) is an AI framework that combines information retrieval from external knowledge sources with large language model text generation to produce accurate, grounded, and contextually relevant responses.

Breaking it down:

Retrieval:

- Fetch relevant information from a knowledge base (ChromaDB)
- Like searching a library for relevant books

Augmented:

- Enhance/supplement the model's internal knowledge
- Give the model external context it didn't have before

Generation:

- Use an LLM to generate natural language answers
- Synthesize retrieved information into coherent responses

The Concept :

Traditional LLM (Without RAG):

User Question → LLM (relies only on training data) → Answer

- Limited to knowledge from training
- Can hallucinate when uncertain
- Can't access current information
- No source attribution

RAG System :

User Question → Retrieve Relevant Docs → LLM + Retrieved Context → Answer

- Accesses external knowledge base
- Grounded in actual documents
- Can provide source citations
- Always up-to-date (database updates instantly)

RAG

WHY RAG MATTERS

The Fundamental LLM Problems RAG Solves

1. Knowledge Cutoff Problem :

- **Problem** : LLMs only know information up to their training date
- **RAG Solution** : Retrieves current information from updated database

2. Hallucination Problem :

- **Problem** : LLMs sometimes "make up" plausible-sounding but false information
- **RAG Solution** : Grounds responses in actual retrieved documents

3. Domain-Specific Knowledge Problem :

- **Problem** : LLMs have broad knowledge but lack deep institutional details
- **RAG Solution** : Retrieves specific institutional documents

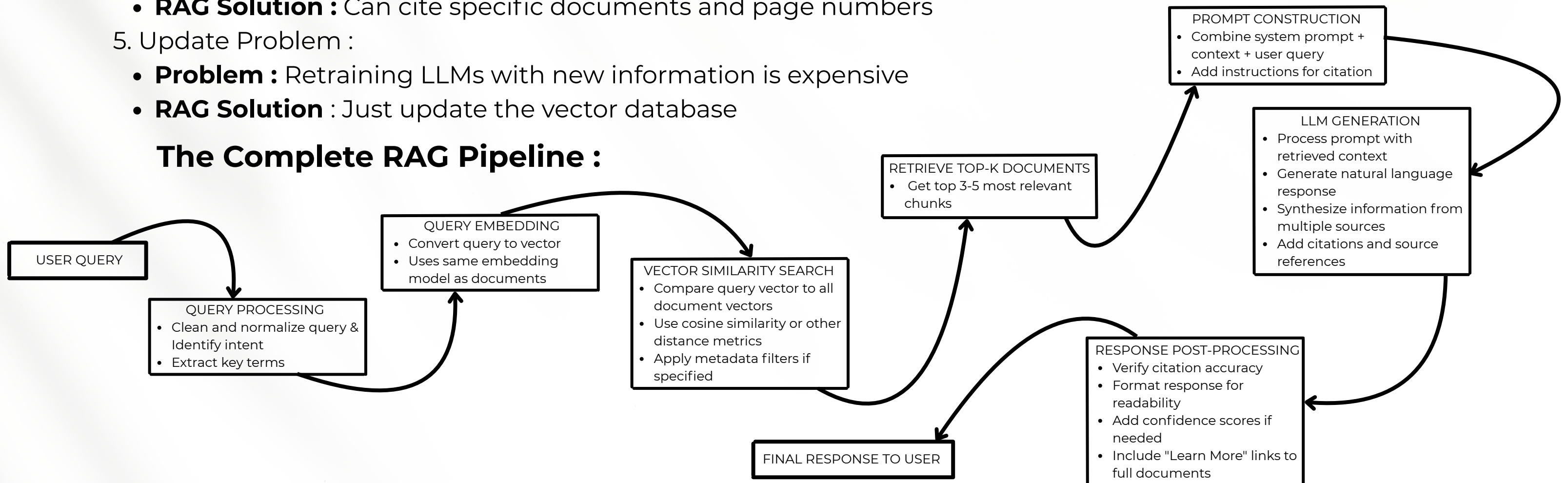
4. Attribution Problem :

- **Problem** : Can't verify where information came from
- **RAG Solution** : Can cite specific documents and page numbers

5. Update Problem :

- **Problem** : Retraining LLMs with new information is expensive
- **RAG Solution** : Just update the vector database

The Complete RAG Pipeline :



RAG

RAG VARIATIONS & ADVANCED TECHNIQUES

Basic RAG : Query → Retrieve → Generate

Characteristics :

- Single retrieval step
- Simple and fast
- Good for most use cases

Advanced RAG Techniques

RAG

Multi-Query RAG :

Problem : User queries can be ambiguous or poorly phrased.

Solution : Generate multiple versions of the query.

Query → Generate Variations → Retrieve documents for each variation → Combine and deduplicate → Generate answer

Self-Query RAG :

Problem: Queries contain both semantic content and metadata filters.

Solution: LLM extracts metadata filters from query.

Query → LLM Extracts → ChromaDB Query with Filters → More precise results

Iterative RAG (Multi-Hop) :

Problem: Complex questions require multiple retrieval steps.

Example: "Compare prerequisites for ML and AI courses"

Query → Retrieval 1: Get ML prerequisites → Retrieval 2: Get AI prerequisites → Retrieval 3: Get policy on prerequisites →

Synthesis: Generate comparison.

FROM RAG TO OPTIMIZATION

We now have a powerful RAG system that retrieves relevant information and generates intelligent responses. But there's a critical challenge:

The Problem:

- Large Language Models are massive (billions of parameters)
- Require significant GPU memory and computational resources
- Slow inference time means poor user experience
- Expensive to deploy and run at scale

The Solution: Model Optimization

- To deploy our RAG system efficiently for ENSAKH students, we need to optimize our LLM without sacrificing quality.

Two Key Techniques:

- Quantization - Reduce model size and memory footprint
- Pruning - Remove unnecessary parameters

QUANTIZATION

QUANTIZATION *“Making Models Lighter”*

Definition :

Quantization is a model optimization technique that reduces the precision of numerical representations (weights and activations) in neural networks, converting them from higher-bit formats to lower-bit formats to decrease model size and accelerate inference.

How It Works:

Standard Model (FP32):

- Each parameter stored as 32-bit floating point
- High precision but large memory footprint

Quantized Model (INT8):

- Each parameter stored as 8-bit integer
- 4x smaller memory footprint
- Minimal accuracy loss (typically <1%)

Precision Formats

QUANTIZATION

- FP16 (16-bit Floating Point):
- Half-precision format
 - Range: $\pm 65,504$
 - 2x memory reduction vs FP32
 - Faster computation on modern GPUs
 - Good balance of speed and accuracy
 - Use case: General-purpose training and inference

- BF16 (Brain Float 16):
- Google's optimized 16-bit format
 - Same range as FP32 ($\pm 3.4 \times 10^{38}$)
 - Better for deep learning than FP16
 - More stable during training
 - Prevents overflow/underflow issues
 - Use case: Training large models, mixed-precision training

BITSANDBYTES

Definition:

bitsandbytes is a lightweight Python library that provides optimized 8-bit and 4-bit quantization techniques for PyTorch models, enabling efficient training and inference of large language models with minimal accuracy degradation.

Key Features :

8-bit Quantization :

- Reduces memory by 75%
- Maintains 99%+ model quality
- Enables training larger models on consumer GPUs

4-bit Quantization (QLoRA):

- Reduces memory by 93.75%
- Fine-tune 65B models on single 48GB GPU
- Minimal performance loss

8-bit Optimizers :

- AdamW optimizer in 8-bit
- Same memory as SGD, performance of Adam
- Critical for fine-tuning large models

Beyond Compression

Quantization reduces the precision of parameters, but what if we could reduce the number of parameters entirely?

The Insight :

- Not all neurons in a neural network contribute equally:
- Some parameters are crucial for model performance
- Others have minimal impact on predictions
- Removing redundant parameters = smaller, faster models

PRUNING

The Solution : Pruning

PRUNING *“Removing What's Unnecessary”*

Definition :

Pruning is a neural network optimization technique that systematically removes redundant or less important parameters (weights, neurons, or entire layers) from a trained model to reduce its size and computational requirements while maintaining performance.

The Concept:

Think of pruning like trimming a tree:

- Remove dead branches (zero or near-zero weights)
- Keep the healthy structure (important connections)
- Tree stays alive and functional (model maintains accuracy)

Types of Pruning:

Unstructured Pruning :

- Remove individual weights based on magnitude
- Maximum compression (up to 90% sparsity)
- Requires specialized hardware for speedup

Structured Pruning:

- Remove entire neurons, channels, or layers
- Less compression (30-50% typical)
- Immediate speedup on standard hardware
- Better for deployment

Magnitude-Based Pruning:

- Remove weights with smallest absolute values
- Simple and effective
- Most common approach

PRUNING

HOW PRUNING WORKS

Step-by-Step Process:

1. Train Full Model
 - Start with fully trained, unpruned model
 - Establish baseline performance
2. Identify Unimportant Parameters
 - Calculate importance scores for each weight
 - Methods: magnitude, gradient-based, Hessian-based
3. Remove Parameters
 - Set selected weights to zero
 - Or physically remove neurons/channels
4. Fine-tune
 - Retrain briefly to recover accuracy
 - Remaining weights compensate for removed ones
5. Iterate (Optional)
 - Gradually increase sparsity
 - Iterative pruning often outperforms one-shot

Impact on Our RAG System:

- Faster inference (30-50% speedup)
- Smaller model footprint
- Reduced latency for student queries
- Combined with quantization = maximum efficiency

BRINGING IT ALL TOGETHER

We've optimized our model through : Quantization (Reduced precision) -> Pruning (Removed redundancy)

Now we need the tools to implement this efficiently.

The Challenge :

- Loading and configuring optimized models
- Managing datasets for fine-tuning
- Deploying models efficiently
- Handling tokenization and inference

The Solution : HuggingFace

HUGGING FACE *"The AI Development Platform"*

HUGGING FACE

Definition :

HuggingFace is an open-source platform and ecosystem that provides state-of-the-art machine learning tools, pre-trained models, datasets, and libraries for natural language processing and AI development, serving as the "GitHub of Machine Learning."

it's powerful and essential because of :

- Industry standard for LLM development
- Massive community support
- Seamless integration with our tech stack
- Free tier for education/research

HUGGING FACE LIBRARIES

HuggingFace contains a lot of powerful libraries like :

Transformers :

Definition : A library providing thousands of pre-trained models for NLP, computer vision, and audio tasks with a unified API.

Key Features:

- Load any model with 3 lines of code
- Support for 100+ architectures (BERT, GPT, LLaMA, etc.)
- Easy fine-tuning and inference
- Automatic tokenization and preprocessing

HUGGING FACE

Datasets :

Definition : A library for efficiently loading, processing, and sharing datasets with built-in caching and memory mapping.

Key Features:

- Fast loading of large datasets
- Memory-efficient (doesn't load entire dataset into RAM)
- Built-in preprocessing functions
- Easy integration with training loops

PEFT (Parameter-Efficient Fine-Tuning) :

Definition : A library for fine-tuning large models efficiently using techniques like LoRA, adapters, and prefix tuning.

Key Features:

- Fine-tune with <1% of original parameters
- Dramatically reduced memory requirements
- LoRA, QLoRA, Adapter layers
- Multiple adapters for different tasks

The Result:

A theoretically sound, optimized RAG system ready for implementation.

From Theory to Practice

"We have the perfect system architecture... but what do we feed it?"

In this chapter we've Covered : Large Language Models - The foundation of modern AI, Distillation - Teacher-student knowledge transfer, Fine-tuning Approaches - Adapting models to specific domains, Vector Databases - Semantic storage and retrieval, ChromaDB - Our chosen vector database solution, RAG (Retrieval-Augmented Generation) - Combining retrieval with generation, HuggingFace Ecosystem - Tools for implementation, Quantization & Pruning - Model optimization techniques

We now understand every component needed to build an intelligent, optimized RAG system for ENSAKH.

DATA COLLECTION AND PREPARATION

DATA COLLECTION & PREPARATION

CHAPTER OVERVIEW

Great AI systems are built on great data. In this chapter, we transform raw ENSAKH web content into a production-ready knowledge base.

What We'll Cover:

1. Data Collection Strategy
2. Storage Architecture
 - ChromaDB for vector embeddings and semantic search
 - JSON files for structured metadata and versioning
 - Dual-storage strategy for flexibility
3. Data Quality Pipeline
 - Cleaning techniques: removing HTML artifacts, normalizing text
 - Processing workflows: chunking, metadata extraction
 - Validation and quality assurance
4. Dataset Engineering
 - Training set creation for fine-tuning
 - Test set design for evaluation
 - Split strategy and data distribution

The Goal:

Transform unstructured web content into a clean, structured, semantically-rich knowledge base that powers both retrieval and generation. *We've designed the perfect system. Now we need to feed it.*

Our Challenge:

- ENSAKH information is scattered across web pages
- Unstructured HTML content
- Mixed formats and inconsistent organization
- No existing structured dataset

Our Solution:

Build a comprehensive data collection and preparation pipeline

WEB SCRAPING

COMPLETE ENSAKH DATA EXTRACTION

Scraping Scope - Everything:

We systematically scraped the entire ENSAKH website to capture:

Academic Content:

- All departments and their majors.
- Complete program descriptions for each major
- Course catalogs with detailed syllabi
- Course schedules and timetables
- Prerequisites and course sequences

Institutional Information:

- ENSAKH general information (location, contact details, phone numbers)
- Administration hierarchy (Director, Secretary, Adjoint directors)
- Faculty members directory with names and email addresses
- Department structures and their specializations

Educational Programs:

- Formation Continue (Continuing Education) programs
- Formation Initiale (Initial Education) programs
- Differences and comparison between formation types

Technologies Used:

- BeautifulSoup4 (bs4): For parsing and extracting data from HTML pages.
- Requests: Handles HTTP requests to fetch webpage content.
- urllib / urlparse: Used for URL handling, validation, and manipulation.
- re (Regular Expressions): Detects and extracts patterns such as emails, phone numbers, and structured text.
- json: Stores and serializes extracted data in a structured format.

WEB SCRAPING

Scraping Strategy:

- Perform recursive crawling starting from the homepage.
- Follow internal links systematically to ensure full site coverage.
- Extract structured information (emails, phone numbers, names, etc.).
- Preserve document hierarchy and data relationships for better organization and later use.

SCRAPING RESULTS & THE DUPLICATION CHALLENGE

Initial Scraping Results:

- Complete ENSAKH website scraped successfully
- All pages, documents, and resources extracted
- Faculty directory: [X] members with emails
- Departments: All majors and programs covered
- Courses: Complete catalog with schedules
- Administration: Full organizational structure
- Documents: All available resources captured

The Critical Problem Discovered : Massive Duplication

Initial Dataset Size: Thousands of raw text entries about 12000 examples

After Deduplication: Significant reduction in unique content (after cleaning, havig just 2.1k examples)

DATA STORAGE

Dual Storage Architecture

Why Two Storage Systems? Each serves a distinct purpose in our pipeline:

1. ChromaDB (Vector Database):

Purpose: Semantic search and RAG retrieval

What We Store:

- Text embeddings from cleaned data
- Document chunks optimized for retrieval

WEB SCRAPING

2. JSON Files (Structured Storage):

Purpose: Data preservation, cleaning, and fine-tuning preparation

What We Store:

- Raw scraped content initially
- Cleaned and deduplicated data
- Structured information (faculty, courses, departments)

COMPLETE DATA PIPELINE

- Step 1: Web Scraping ↓ Raw HTML from <http://ensakh.usms.ac.ma/ensak/>
- Step 2: Initial JSON Storage ↓ Save all scraped content to JSON files (Thousands of entries with heavy duplication)
- Step 3: Deduplication & Cleaning ↓ Remove duplicate content, HTML artifacts, boilerplate. Filter out navigation menus, headers, footers
- Step 4: Quality Filtering ↓ Keep only unique, valuable content Validate completeness and accuracy
- Step 5: Structured Organization ↓ Categorize by type: faculty, departments, courses, documents. Add metadata and contextual information
- Step 6: Final Clean Dataset : 2.1K high-quality, unique examples. Ready for fine-tuning
- Step 7: Parallel Storage : JSON: Clean dataset for fine-tuning, ChromaDB: Embedded vectors for RAG retrieval

We scraped everything from ENSAKH website.

But we faced a harsh reality:

Initial Data: Thousands of scraped entries

After Deduplication: Massive reduction

Final Clean Dataset: Only 2.1K examples

The Challenge :

- Most AI projects recommend 10K-100K examples for fine-tuning.
- We had only 2.1K high-quality examples.

My Choice:

- Bad dataset with low-quality duplicates
- Prioritize quality over quantity
 - "2K perfect examples is better than 10K mediocre ones"

FINE-TUNING AND MODEL TRAINING

FROM DATA TO INTELLIGENCE

The Challenge Ahead:

How do i fine-tune a powerful LLM with only 2.1K examples and make it work effectively?

Before all this, first of all we have to choose the right open-source model by asking ourself : Which Large Language Model should we fine-tune?

The Challenge:

- Hundreds of open-source LLMs available
- Different sizes, capabilities, and requirements
- Must balance performance vs. resources

MODEL SELECTION

We've Selected: Meta LLaMA 3.1:8B-Instruct

But here's the reality:

LLaMA 3.1 has:

- Massive general knowledge
- Strong reasoning capabilities
- Zero knowledge about ENSAKH

Our 2.1K Dataset Contains:

- ENSAKH programs, courses, faculty
- Admission requirements
- Department structures
- Formation Continue vs. Initiale

The Goal : Transfer our ENSAKH knowledge into LLaMA 3.1 without destroying its base capabilities.

The Solution : LoRA Fine-Tuning → Train only 0.92% of the model's parameters while keeping 99.08% frozen.

Teaching LLaMA About ENSAKH

- Instead of modifying the entire model, LoRA adds small "adapter" layers that learn ENSAKH-specific knowledge.

FINE-TUNING

WHY ?

We chose LoRA (Low-Rank Adaptation) instead of standard fine-tuning because it offers a far more efficient and safer approach. In standard fine-tuning, all 8 billion parameters of the model are updated, which demands over 32GB of VRAM, takes several days to train, and carries a high risk of catastrophic forgetting, where the model loses previously learned knowledge. In contrast, LoRA updates only 41.9 million parameters (about 0.92%), reducing the hardware requirement to around 10GB of VRAM and cutting training time to just 1–2 hours. This method significantly minimizes the risk of forgetting while effectively preserving the model's base knowledge, making it a more practical and resource-efficient choice.

LoRA :

Definition : Low-Rank Adaptation (LoRA) is a parameter-efficient fine-tuning technique that freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture.

What This Means:

r=16 (Rank):

- Controls adapter size
- Higher = more capacity, more memory

lora_alpha=32:

- Scaling factor for adapters
- Prevents adapters from being too weak

target_modules:

- We adapt 7 key projection layers
- Covers both attention and MLP

```
LORA_R = 16
LORA_ALPHA = 32
LORA_DROPOUT = 0.1
LORA_TARGET_MODULES = [
    "q_proj", "k_proj", "v_proj", "o_proj",
    "gate_proj", "up_proj", "down_proj"
]
```

The LoRA configuration from my notebook

Fitting an 8B Model into Limited VRAM

Running a large 8-billion-parameter model in full precision (FP32) requires massive memory — roughly 4 bytes per parameter, totaling around tens of gigabytes. On standard GPUs with limited VRAM, this makes training or even loading the model impossible without optimization.

Our Optimization Stack:

4-Bit Quantization (QLoRA)

```
# 4-bit Quantization
USE_4BIT = True
BNB_4BIT_COMPUTE_DTYPE = "float16"
BNB_4BIT_QUANT_TYPE = "nf4"
USE_NESTED_QUANT = True
```

FINE-TUNING

Effect: Reduces memory use by up to 8×, bringing the model size from ~32GB down to ~5GB.

Mixed Precision Training (FP16)

```
fp16=True,
```

Uses 16-bit floats for faster computation, doubling training speed and reducing gradient memory consumption.

TRAINING CONFIGURATION

Optimized Training Parameters

3 Epochs:

- With 2.1K examples, 3 epochs = 603 training steps

Learning Rate 2e-4:

- Optimal for LoRA

Cosine Scheduler:

- Better final convergence

```
NUM_EPOCHS = 3
BATCH_SIZE = 4
GRADIENT_ACCUMULATION = 2
LEARNING_RATE = 2e-4
WARMUP_RATIO = 0.1
LR_SCHEDULER = "cosine"
WEIGHT_DECAY = 0.01
MAX_SEQ_LENGTH = 512

EVAL_STRATEGY = "steps"
EVAL_STEPS = 50
SAVE_STEPS = 50
SAVE_TOTAL_LIMIT = 2
LOAD_BEST_AT_END = True
METRIC_FOR_BEST = "eval_loss"
```

DATA SPLIT STRATEGY

Intelligent Stratified Splitting

Because the random splits can:

- Put similar questions in train vs. test
- Fail to test true generalization
- Miss important topic coverage

Our Solution: Stratified Splitting by Topic

Final Dataset Split

Split	Examples	Purpose
Training	1,601 (80%)	Model learning
Validation	98 (5%)	Overfitting monitoring
Test	206 (15%)	True evaluation

Smart Deduplication:

- Removed test examples with >85% similarity to training
- Reduced from 296 → 206 test examples
- Ensures truly unseen evaluation

TRAINING PROGRESS VISUALIZATION

Training Loss: 0.75 → 0.17 (77% reduction)

- Steady decrease = model learning effectively
- No sudden jumps = stable training

Validation Loss: 0.68 → 0.41 (40% reduction)

- Follows training loss = good generalization
- No divergence = no overfitting

Key Insight: The gap between train (0.17) and validation (0.41) is healthy , indicates the model learned ENSAKH knowledge without memorization.

TRAINING

[603/603 1:07:41, Epoch 3/3]					
Step	Training Loss	Validation Loss	Entropy	Num Tokens	Mean Token Accuracy
50	0.751500	0.678829	0.652085	64127.000000	0.835780
100	0.532200	0.564191	0.564944	127282.000000	0.852584
150	0.557800	0.503192	0.525366	189894.000000	0.863804
200	0.468000	0.466347	0.469037	252407.000000	0.871329
250	0.337100	0.471968	0.388235	314782.000000	0.874312
300	0.360400	0.437233	0.388619	377840.000000	0.880299
350	0.307500	0.425830	0.362746	440023.000000	0.884832
400	0.318400	0.405934	0.353955	503520.000000	0.890194
450	0.170300	0.432955	0.266945	566165.000000	0.895179
500	0.193400	0.419806	0.269118	630536.000000	0.898654
550	0.158600	0.419728	0.263090	691746.000000	0.898622
600	0.169500	0.419469	0.262693	754780.000000	0.898949

TRAINING

Test Set Performance

The test set performance on unseen data shows strong generalization and accuracy. The model correctly identified the S5 module for IID, recognized “Diplôme d'Ingénieur d'État” the engineering programs, and appropriately recommended the GI program in French for a Full-Stack developer query, demonstrating solid domain knowledge, institutional awareness, and multilingual capability.

Training Achievements

The training process was highly efficient, updating only 0.92% of parameters (41.9M out of 8B) and completing in about one hour on a consumer GPU. The model effectively transferred ENSAKH-specific knowledge while preserving the base LLaMA 3.1 capabilities. Validation loss consistently decreased, confirming no overfitting and strong generalization on unseen data. Finally, LoRA adapters were merged into the base model, producing a standalone 8B version ready for production deployment.

We Have a Trained Model, But There's a Problem

Current Situation: Model trained on 2.1K examples, Knows ENSAKH programs, courses, faculty, Understands questions in 3 languages, 89.9% token accuracy on test set

The Limitation: Knowledge frozen at training time, Cannot access new documents or updates, Limited to what was in training data

Example Problems:

New course added next semester? Model doesn't know

Updated admission criteria? Model uses old information

New faculty member? Not in model's knowledge

The Solution : RAG (Retrieval-Augmented Generation)

Our model is trained. It's intelligent. It understands ENSAKH.

But, It needs eyes. It needs access to documents. It needs to stay current.

That's where RAG comes in.

RAG

What is RAG?

Retrieval-Augmented Generation (RAG) allows a language model to consult external documents dynamically at inference time, ensuring access to the most recent information.

Without RAG, a query is answered solely from the model's internal memory.

With RAG, the system retrieves semantically relevant documents from a vector database, injects them into the model's context, and then generates a grounded, up-to-date response.

The model can cite sources and deliver accurate, current answers.

RAG

Overview

In this chapter, we focus on integrating RAG with our fine-tuned ENSAKH model.

We begin with the vector database implementation, continue with the RAG system architecture (currently in progress), and conclude with testing and feedback collection a crucial step toward RLHF preparation

Vector Database Implementation

We successfully deployed ChromaDB as our vector store, embedding ENSAKH documents with sentence-transformers.

All institutional PDFs were processed, chunked, and stored as high-dimensional embeddings. Also i have an automated ingestion pipeline that continuously monitors ENSAKH document directories, detects new files, extracts and chunks their content, generates embeddings, and stores them in ChromaDB.

Each document is enriched with metadata to have in the end all the new document becomes available to the RAG system and keeping knowledge always current.

From Storage to Retrieval

Having built a structured and dynamic knowledge base, the next challenge is leveraging it effectively.

- How can our fine-tuned LLM access this data?
- How do we ensure retrieved content is both relevant and coherent in context?

RAG ARCHITECTURE DESIGN

RAG Architecture

Our full RAG pipeline connects the user interface, FastAPI backend, retrieval engine (ChromaDB), context builder, and the fine-tuned ENSAKH model.

This architecture ensures accuracy, interpretability, and responsiveness.

Query Processing Flow

Each user question follows a structured flow:

- Query Reception
- Intent Detection
- Vector Search
- Context Injection
- LLM Generation

This pipeline guarantees factual, contextually relevant answers.

LLM Integration

The RAG system integrates seamlessly with our fine-tuned ENSAKH model, loaded through the Hugging Face transformers library.

Queries enriched with context are tokenized, fed into the model, and decoded into final responses producing domain-specific and multilingual answers.

With the RAG system now operational, the next objective is measuring how well it performs across accuracy, speed, and user satisfaction.

A user feedback interface collects real-time evaluations (thumbs up/down, optional comments) for every generated answer. This data forms the foundation for RLHF training, allowing the system to learn directly from user preferences and continuously improve.

Having validated the RAG system's technical performance, we now move beyond correctness toward human alignment.

Chapter 7 introduces Reinforcement Learning from Human Feedback (RLHF) — the final step toward making our assistant not just accurate, but truly helpful, natural, and user-centered.

RHLF

CHAPTER OVERVIEW

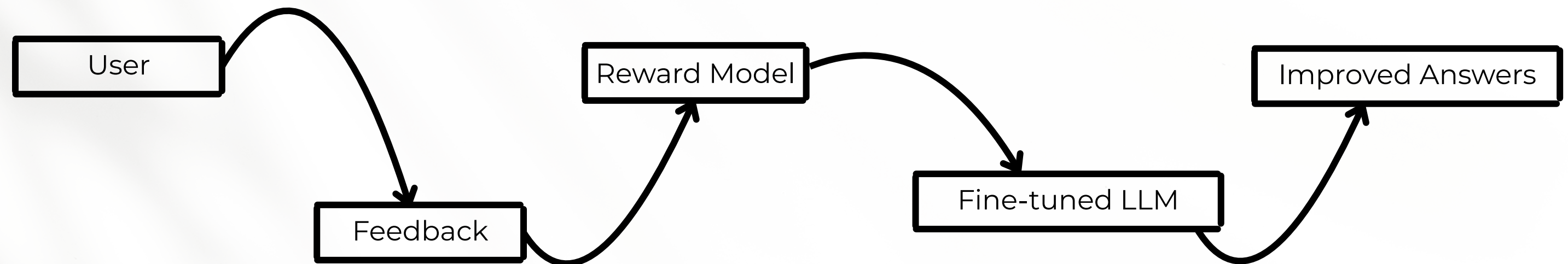
Goal:

This chapter focuses on enhancing the assistant's intelligence through human-aligned learning.

Objectives:

- Integrate human feedback for continuous improvement
- Optimize responses for clarity, tone, and relevance
- Build a feedback-training loop to make the model adaptive and trustworthy

RHLF



Why RLHF?

Traditional fine-tuning focuses on correctness, but not helpfulness or tone.

RLHF introduces a human preference layer, teaching the model how to respond in ways that are both accurate and user-friendly.

Example:

- Two responses might be factually correct but RLHF ensures the model chooses the one that best fits the user's intent and communication style.

The final ENSAKH Assistant combines:

- A fine-tuned LLaMA 3.1-8B model
- A dynamic RAG knowledge base
- Continuous improvement via RLHF feedback loop

Together, they create an intelligent, evolving assistant — accurate, human-aligned, and always up to date.

CONCLUSION

PROJECT JOURNEY

- ENSAKH students struggle to find information
- Generic chatbots don't understand our specific context
- Manual information retrieval is time-consuming
- Our Vision: Build an intelligent, specialized AI assistant that understands ENSAKH deeply and serves our academic community effectively.

The Result : A production-ready chatbot powered by fine-tuned LLaMA 3.1 + RAG + RLHF

CONCLUSION

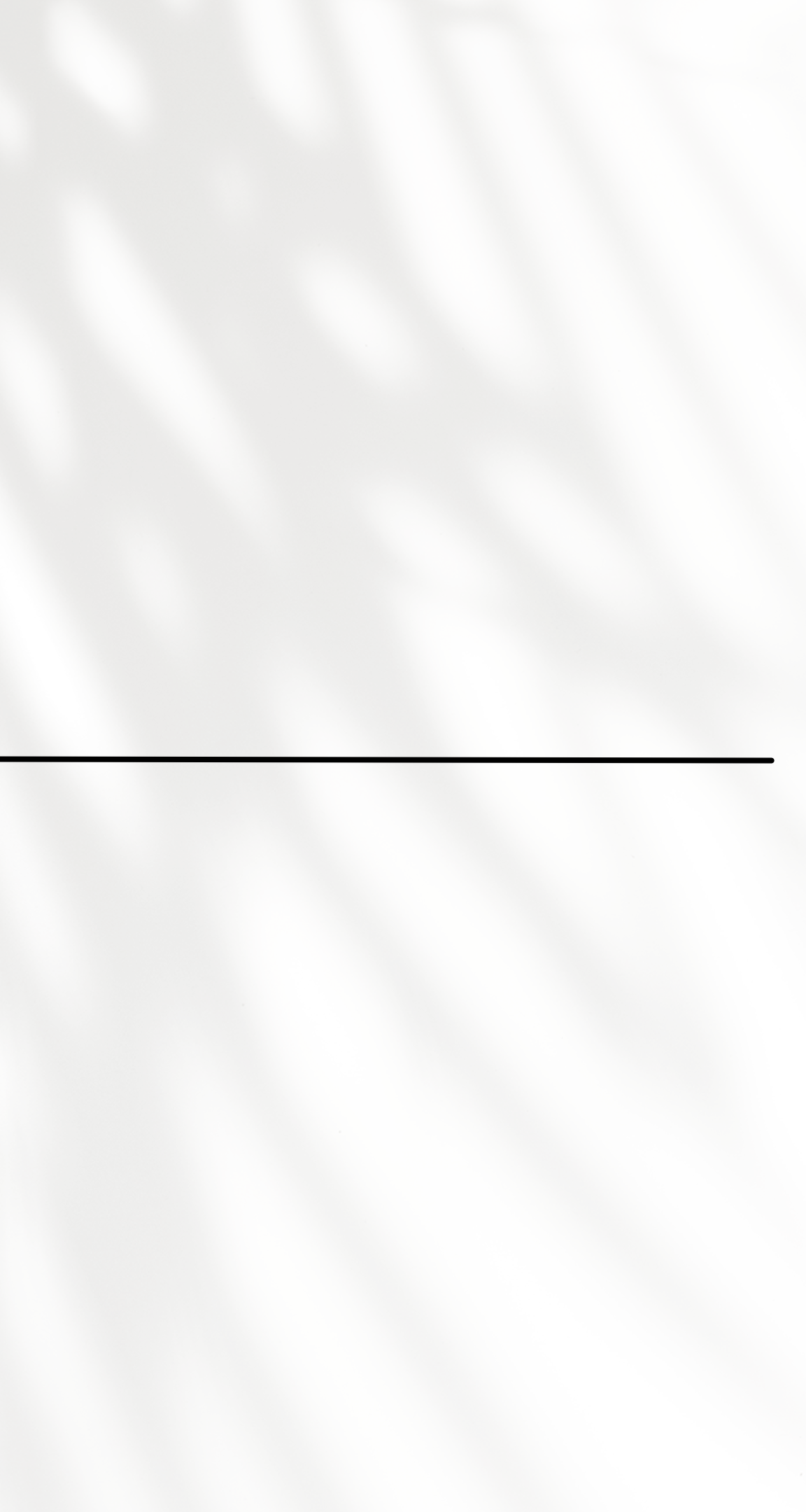
CONCLUSION

This project represents a comprehensive journey in developing a specialized AI assistant tailored specifically for ENSAKH's academic community. Starting with the challenge of information accessibility, we embarked on creating a solution that combines cutting-edge AI technologies with practical implementation.

we transformed 2,100 carefully collected examples into a production-ready chatbot system. Our approach integrated multiple advanced techniques: fine-tuning Meta's LLaMA 3.1:8B model using efficient LoRA methodology, implementing a dynamic Retrieval-Augmented Generation system with ChromaDB, and aligning the model with human preferences through Reinforcement Learning from Human Feedback.

The journey from raw data to an intelligent, helpful assistant has been challenging and rewarding. We've contributed code, methodologies, and insights back to the open-source community, ensuring that others can build upon our work.

As ENSAKH continues to grow and evolve, so too will this assistant a living system that learns, adapts, and serves its community better with each passing day.



THANK YOU FOR YOUR ATTENTION

Questions? Discussion? Let's talk!