10 Academy: Artificial Intelligence Mastery

Intelligent Complaint Analysis for Financial Services

Building a RAG-Powered Chatbot to Turn Customer Feedback into Actionable Insights  
Date: 02 July - 08 July 2025

# 

# Table of Contents

# 

[Table of Contents 2](#_hqh6balq47w)

[Business Objective 3](#_v2lj37fuk3pd)

[Motivation 3](#_c7igw6ge2muw)

[Data 4](#_z4zo7fs8378e)

[Learning Outcomes 4](#_1m4xlttsha20)

[Team 5](#_xjnt1wnjiu9f)

[Key Dates 5](#_hzrka62be2ei)

[Deliverables and Tasks to be done 6](#_4mawrfhyasr8)

[Task 1: Exploratory Data Analysis and Data Preprocessing 6](#_j5fs8db8da80)

[Task 2: Text Chunking, Embedding, and Vector Store Indexing 6](#_sqe2moa2x4px)

[Task 3: Building the RAG Core Logic and Evaluation 7](#_nygf0zrgb98y)

[Task 4: Creating an Interactive Chat Interface 8](#_di5olsyuzvou)

[Due Date (Submission) 9](#_q1hhigifr9d3)

[Interim Submission Sunday (06 July 2025): 8:00 PM (UTC) 9](#_5xf5zqh9myc3)

[Final Submission Tuesday (08 July, 2025): 8:00 PM (UTC) 9](#_jfjaaufbsjne)

[Feedback 9](#_fo37eyc50kle)

[Other Considerations: 9](#_tdjgno5s5v6s)

[Tutorials Schedule 10](#_z8j5w6yqu4s5)

[References 11](#_jm854qxlbut)

# 

# Business Objective

CrediTrust Financial is a fast-growing digital finance company serving East African markets through a mobile-first platform. Their offerings span across:

* Credit Cards
* Personal Loans
* Buy Now, Pay Later (BNPL)
* Savings Accounts
* Money Transfers

With a user base of over 500,000 and operations expanding into three countries, CrediTrust receives thousands of customer complaints per month through its in-app channels, email, and regulatory reporting portals.

Your mission is to spearhead the development of an internal AI tool that transforms this raw, unstructured complaint data into a strategic asset. You are building this tool for internal stakeholders like Asha, a Product Manager for the BNPL team. Asha currently spends hours each week manually reading complaints to guess at emerging issues. She needs a tool that lets her ask direct questions and get synthesized, evidence-backed answers in seconds.

The success of your project will be measured against three Key Performance Indicators (KPIs):

* Decrease the time it takes for a Product Manager to identify a major complaint trend from days to minutes.
* Empower non-technical teams (like Support and Compliance) to get answers without needing a data analyst.
* Shift the company from reacting to problems to proactively identifying and fixing them based on real-time customer feedback.

# Motivation

CrediTrust’s internal teams face serious bottlenecks:

* Customer Support is overwhelmed by the volume of incoming complaints.
* Product Managers struggle to identify the most frequent or critical issues across products.
* Compliance & Risk teams are reactive rather than proactive when it comes to repeated violations or fraud signals.
* Executives lack visibility into emerging pain points due to scattered and hard-to-read complaint narratives.

As a Data & AI Engineer at CrediTrust Financial, you are tasked with developing an intelligent complaint-answering chatbot that empowers product, support, and compliance teams to understand customer pain points across five major product categories quickly:

1. Credit Cards
2. Personal Loans
3. Buy Now, Pay Later (BNPL)
4. Savings Accounts
5. Money Transfers

Your goal is to build a Retrieval-Augmented Generation (RAG) agent that:

* Allows internal users to ask plain-English questions about customer complaints (e.g., “Why are people unhappy with BNPL?”)
* Uses semantic search (via a vector database like FAISS or ChromaDB) to retrieve the most relevant complaint narratives
* Feeds the retrieved narratives into a language model (LLM) that generates concise, insightful answers
* Supports multi-product querying, making it possible to filter or compare issues across financial services.

# Data

This challenge uses complaint data from the Consumer Financial Protection Bureau (CFPB). The dataset contains real customer complaints across multiple financial products, including credit cards, personal loans, savings accounts, BNPL, and money transfers.

[**Dataset Link**](https://drive.google.com/drive/folders/1xzGB5_K4IWvQOZDXGpzgrmBmfdaDek4Y?usp=drive_link)

Each record includes:

* A short issue label (e.g., *"Billing dispute"*)
* A free-text narrative written by the consumer
* Product and company information
* Submission date and complaint metadata

You will use the Consumer complaint narrative as the core input for embedding and retrieval, enabling your chatbot to answer questions based on real-world feedback.

## Learning Outcomes

By completing this challenge, you will:

* Learn how to combine vector similarity search with language models to answer user questions based on unstructured data.
* Gain experience handling noisy, unstructured consumer complaint narratives and extracting meaningful insights.
* Learn how to create and query a vector database (e.g., FAISS or ChromaDB) using embedding models to power semantic search.
* Develop a chatbot that uses real retrieved documents as context for generating intelligent, grounded answers using LLMs.
* Create a system that can analyze and respond across multiple financial product categories, simulating a real business environment.
* Build and test a simple user interface that allows natural-language querying of large-scale complaint data.

## Team

Facilitator:

* Mahlet
* Kerod
* Rediet
* Rehmet

# Key Dates

* **Challenge Introduction** - 9:30 AM UTC time on Wednesday 02 July 2025.
* **Interim Submission -** 8:00 PM UTC time on Sunday 06 July 2025.
* **Final Submission** - 8:00 PM UTC time on Tuesday 08 July 2025.

# Deliverables and Tasks to be done

Please follow the tasks below to build your complaint-answering chatbot using Retrieval-Augmented Generation (RAG). The project should be completed step by step, with clean code, organized folders, and clear documentation.

## Task 1: Exploratory Data Analysis and Data Preprocessing

To understand the structure, content, and quality of the complaint data and prepare it for the RAG pipeline.

* Load the full CFPB complaint dataset.
* Perform an initial EDA to understand the data.
  + Analyze the distribution of complaints across different Products.
  + Calculate and visualize the length (word count) of the Consumer complaint narrative. Are there very short or very long narratives?
  + Identify the number of complaints with and without narratives.
* Filter the dataset to meet the project's requirements:
  + Include only records for the five specified products: Credit card, Personal loan, Buy Now, Pay Later (BNPL), Savings account, Money transfers.
  + Remove any records with empty Consumer complaint narrative fields.
* Clean the text narratives to improve embedding quality. This may include:
  + Lowercasing text.
  + Removing special characters or boilerplate text (e.g., "I am writing to file a complaint...").
  + (Optional) Consider other text normalization techniques.

**Deliverables**:

* A Jupyter Notebook or Python script in notebooks/ or src/ that performs the EDA and preprocessing.
* A short summary (2-3 paragraphs) in your report describing key findings from the EDA.
* The cleaned and filtered dataset saved to data/filtered\_complaints.csv.

## Task 2: Text Chunking, Embedding, and Vector Store Indexing

To convert the cleaned text narratives into a format suitable for efficient semantic search.

* Long narratives are often ineffective when embedded as a single vector. Implement a text chunking strategy.
  + Use a library like LangChain's RecursiveCharacterTextSplitter or write your own function.
  + Experiment with chunk\_size and chunk\_overlap to find a good balance. Justify your final choice in your report.
* Choose an embedding model.
  + A good starting point is sentence-transformers/all-MiniLM-L6-v2.
  + In your report, briefly explain why you chose this model.
* Embedding and Indexing:
  + For each text chunk, generate its vector embedding.
  + Create a vector store using **FAISS** or **ChromaDB**.
  + Store the embeddings in the vector database. Crucially, ensure you store metadata alongside each vector (e.g., the original complaint ID, the product category) so you can trace a retrieved chunk back to its source.

Deliverables:

* A script that performs chunking, embedding, and indexing.
* The persisted vector store is saved in the vector\_store/ directory.
* A section in your report detailing your chunking strategy and embedding model choice.

## Task 3: Building the RAG Core Logic and Evaluation

To build the retrieval and generation pipeline and, most importantly, to evaluate its effectiveness.

* **Retriever Implementation:**
  + Create a function that takes a user's question (string) as input.
  + Embeds the question using the same model from Task 2.
  + Performs a similarity search against the vector store to retrieve the top-k most relevant text chunks. k=5 is a good starting point.
* **Prompt Engineering:**
* Design a robust prompt template. This is critical for guiding the LLM. The template should instruct the model to act as a helpful analyst, use only the provided context, and answer the user's question based on that context.
  + Example Template:
    - You are a financial analyst assistant for CrediTrust. Your task is to answer questions about customer complaints. Use the following retrieved complaint excerpts to formulate your answer. If the context doesn't contain the answer, state that you don't have enough information. Context: {context} Question: {question} Answer:
* **Generator Implementation**:
  + Combine the prompt, the user question, and the retrieved chunks.
  + Send the combined input to an LLM (e.g., using **Hugging Face's** pipeline or **LangChain's** integrations with Mistral, Llama, etc.).
  + Return the LLM's generated response.
* **Qualitative Evaluation**: This is the most important step.
  + Create a list of 5-10 representative questions you want your system to answer well
  + For each question, run your RAG pipeline and analyze the results.
  + Create an evaluation table in your report (Markdown format is fine) with columns: Question, Generated Answer, Retrieved Sources (show 1-2), Quality Score (1-5), and Comments/Analysis.

Deliverables:

* Python modules (.py file) containing your RAG pipeline logic.
* The evaluation table and your analysis in the final report, explaining what worked well and what could be improved.

## Task 4: Creating an Interactive Chat Interface

To build a user-friendly interface that allows non-technical users to interact with your RAG system.

* Use Gradio or Streamlit to build the web interface.
* Core Functionality: The interface must have:
  + A text input box for the user to type their question.
  + A "Submit" or "Ask" button.
  + A display area for the AI-generated answer.
* Enhancing Trust and Usability (Key Requirements):
  + Display Sources: Below the generated answer, display the source text chunks that the LLM used. This is crucial for user trust and allows for verification.
  + Streaming (Optional but Recommended): If possible, implement response streaming so the answer appears token-by-token, improving the user experience.
  + A "Clear" button to reset the conversation.

**Deliverables**:

* An app.py script that runs the Gradio/Streamlit application.
* Screenshots or a GIF of your working application included in your final report.
* Ensure the code is clean and the UI is intuitive.

# Due Date (Submission)

#### **Interim Submission Sunday (06 July 2025): 8:00 PM (UTC)**

* GitHub link to your main branch, showing merged work from task-1 and task-2.
* Interim report - Covering task-1 and task-2 summarizing your work

#### **Final Submission Tuesday (08 July, 2025): 8:00 PM (UTC)**

* GitHub Link to your main branch
* A polished **final report** in the format of a Medium blog post. This should be a self-contained, professional artifact that includes:
  + Introduction: The business problem and your RAG solution.
  + Technical Choices: Your final decisions on data, chunking, embedding models, and the LLM.
  + System Evaluation: A table showing your test questions, the AI's answers, and your quality analysis.
  + UI Showcase: Screenshots of your working Gradio app.
  + Conclusion: Key challenges, learnings, and future improvements

### Feedback

You may not receive detailed comments on your interim submission but will receive a grade.

### **Other Considerations:**

* **Documentation:** Encourage detailed documentation in code and report writing.
* **Collaboration:** Emphasise collaboration through Github issues and projects.
* **Communication**: Regular check-ins, Q&A sessions, and a supportive community atmosphere.
* **Flexibility:** Acknowledge potential challenges and encourage proactive communication.
* **Professionalism:** Emphasise work ethics and professional behavior.
* **Time Management:** Stress the importance of punctuality and managing time effectively.

# 

# Tutorials Schedule

In the following, the color **purple** indicates morning sessions, and non-purple indicates afternoon sessions.

* Day 1 Wednesday:
  + Introduction to the Challenge (Mahlet)
  + Applied EDA, Preprocessing, and Building the Retrieval Engine (Rediet)
* Day 2 Thursday
  + Building Your Vector Database with ChromaDB or FAISS. (Kerod)
  + Assembling the RAG Pipeline with LangChain and RAG Evaluation (Rehmet)
* Day 3 Friday
  + Introduction to open source LLMs(Kerod)
  + Building an Interactive UI with Gradio(Rehmet)
* Day 4 Monday :
  + Q&A and Project Clinic

# 

# References

* **Chat Elements Documentation**
  + <https://www.gradio.app/docs>
  + <https://docs.streamlit.io/library/api-reference/chat>
* **Vector Databases**
  + <https://docs.trychroma.com/getting-started>
  + <https://github.com/facebookresearch/faiss/wiki/Getting-started>
* **AI Chatbots**
  + <https://realpython.com/python-application-layouts/>
  + <https://www.sbert.net/docs/quickstart.html>
  + <https://huggingface.co/blog/rag>
  + <https://github.com/mayooear/ai-pdf-chatbot-langchain>
  + <https://www.youtube.com/watch?v=AOTiEbIOu98>
  + <https://www.youtube.com/watch?v=M6figgtyuWo>
  + <https://huggingface.co/learn/cookbook/en/rag_with_hf_and_milvus>
  + <https://medium.com/@akriti.upadhyay/implementing-rag-with-langchain-and-hugging-face-28e3ea66c5f7>
  + <https://github.com/langchain-ai/langchain/blob/master/cookbook/Multi_modal_RAG.ipynb>
* **Version control – Git**
  + [What is version control | Atlassian](https://www.atlassian.com/git/tutorials/what-is-version-control)
  + [Learn Git branching -- interactive way to learn Git](https://learngitbranching.js.org/)
  + [Git with large files](https://stackoverflow.com/a/19494211)
  + [Which files to not track and how to not track them? | Atlassian](https://www.atlassian.com/git/tutorials/saving-changes/gitignore)
  + [.gitignore docs](https://git-scm.com/docs/gitignore)
  + [Conventional commits -- lightweight convention on top of commit messages.](https://www.conventionalcommits.org/en/v1.0.0/)
* **CI/CD**
  + [What is Continuous Integration | Atlassian](https://www.atlassian.com/continuous-delivery/continuous-integration)
  + [DevOps Pipeline | Atlassian](https://www.atlassian.com/devops/devops-tools/devops-pipeline)
  + [7 Popular Open Source CI/CD Tools - DevOps.com](https://devops.com/7-popular-open-source-ci-cd-tools/)
  + [Setting up a CI/CD pipeline on Github](https://blog.travis-ci.com/2019-05-30-setting-up-a-ci-cd-process-on-github)