Building a Production-Style RAG System for Neural Network Course Materials

Raj Singh, Srishti Srinivasan, and Vrinda Thakur

School of Professional Studies, Saint Louis University

AA5750: Contemporary Issues in Analytics

Chandra Prakash Bathula

December 14, 2025

Building a Production-Style RAG System for Neural Network Course Materials

Abstract

This project implements a production-style Retrieval-Augmented Generation (RAG) system from scratch using Python and NumPy for the course AA5750: Contemporary Issues in Analytics. The domain consists of deep learning and neural network PDFs stored in a materials folder, which are ingested, preprocessed, and split into chunks using two different strategies: fixed-size word chunks and sentence-based chunks. An open-source embedding model (all-MiniLM-L6-v2) and a closed-source embedding model from OpenAI are used to embed these chunks, and a custom cosine similarity function performs retrieval directly on NumPy arrays. The retrieved chunks are then used as context for a large language model, which answers student questions strictly based on the course materials. A baseline system that queries the model without retrieval is also implemented for comparison. Qualitative analysis using queries about dropout and other neural network concepts shows that the RAG system produces more grounded, course-aligned answers, while the baseline sometimes gives overly generic or multi-interpretation responses. The report also reflects on the trade-offs between chunking strategies, embedding choices, and what would be needed to move this prototype toward a production-ready assistant for technical courses.

Introduction

Students in deep learning and neural network courses often rely on long PDF documents such as research papers, lecture notes, and slide decks. When they have a targeted question—for example, “Explain what a dropout layer does in neural networks” or “Where is the dropout part in these materials?”—they must manually search across several PDFs using scrolling or keyword search. This is slow, and students may miss the most relevant passages. At the same time, large language models (LLMs) like GPT can answer such questions in fluent natural language, but they may not be grounded in the specific course material. The model may give a correct general answer, but not reflect the instructor’s preferred explanations or specific notational choices.

Retrieval-Augmented Generation (RAG) addresses this gap by combining information retrieval with generation. Instead of asking the model to answer purely from its internal knowledge, a retriever first locates the most relevant passages from a domain corpus—in this project, PDFs about BERT, Transformers, residual networks, dropout, backpropagation, and RNN/LSTMs. These passages are then passed to the model as context, and the model is instructed to answer using only that context. This project builds such a RAG system from scratch, following the assignment requirements: manual ingestion, chunking, embeddings, retrieval with NumPy, and grounded answer generation, along with a comparison between RAG and a baseline model without retrieval.

Method

Knowledge Base and Ingestion Pipeline

The knowledge base consists of PDF files stored under a folder path such as "/content/materials". In this implementation, the folder contains several well-known deep learning documents, including research papers like 1810.04805v2.pdf (the BERT paper), 1706.03762v7.pdf (the Transformer paper), and course-style PDFs such as ResNets.pdf, Dropouts-1.pdf, Back-Propagation-1.pdf, and RNN,LSTM,GRU.pdf. The ingestion pipeline is implemented directly in Python using the pypdf library. The code iterates over all files in the materials folder, filters for names ending in .pdf, and processes each PDF.

For each PDF, a PdfReader object is created, and text is extracted page by page using page.extract_text(). Empty pages are skipped. The page texts are concatenated into a full_text string, and whitespace is normalized via ' '.join(full_text.split()), which collapses repeated spaces and line breaks to single spaces. This produces a clean, continuous text representation for each document. Each document is stored as a dictionary with an integer id, the filename, and the cleaned text. All dictionaries are appended to a documents list, which becomes the in-memory corpus for later chunking and retrieval.

Chunking Strategies

To support retrieval and stay within language model context limits, each document's text is divided into smaller units called chunks. The project implements two chunking strategies: a fixed-size word-based strategy and a sentence-based strategy.

The fixed-size chunking function, chunk_text_by_words, splits the text into a list of words using split() and then groups them into segments of a specified size, here 200 words per chunk. For each segment, the words are joined back into a string and metadata is recorded, including the original document id, a chunk index, and the filename. Across all documents, this produced 270 fixed-size chunks in the experiment. This strategy has the advantage of predictable chunk length and simple batching, but it can cut across sentence or topic boundaries and occasionally mix unrelated ideas in the same chunk.

The sentence-based strategy uses a regular expression, re.split(r'(?<=[.!?])\s+', text), to split the document into sentences. Sentences are then grouped into blocks of a fixed number (three sentences per chunk in this project). As with the word-based approach, each chunk is stored with document id, chunk index, and filename. This process resulted in 3,700 sentence-based chunks. Sentence-based chunks are more semantically coherent because they preserve complete sentences and usually align with conceptual explanations, which is useful for technical questions about neural networks.

Embedding Models and Retrieval

Once chunks are created, they are encoded into numerical vectors, or embeddings, that allow semantic similarity to be measured. This project uses both an open-source embedding model and a closed-source embedding model from OpenAI.

For open-source embeddings, the SentenceTransformer library with the all-MiniLM-L6-v2 model is used. All fixed-size chunk texts are collected into a list called texts_fixed, and model_open.encode(texts_fixed, batch_size=32, show_progress_bar=True) is used to compute embeddings. The resulting array, embeddings_open_fixed, has shape (270, 384), meaning 270 chunks and 384-dimensional embeddings. Sentence-

based chunks are similarly embedded into a separate matrix, embeddings_open_sentence, of shape (3700, 384). Storing embeddings as NumPy arrays allows efficient vectorized operations.

For closed-source embeddings, the code defines get_openai_embeddings, which batches the chunk texts and calls client.embeddings.create(model='text-embedding-3-small', input=batch). The returned vectors are assembled into another NumPy array, embeddings_openai_fixed. While this report focuses qualitatively on open-source results, maintaining both embedding matrices makes it possible to compare retrieval behavior across models.

Retrieval is implemented via a custom cosine_similarity function. Given a query embedding a and a matrix of chunk embeddings b, both are normalized to unit length and the dot product is computed, yielding similarity scores. The retrieve_top_k function then embeds the user query using the same model, computes cosine similarity to all chunks, sorts the scores, and returns the top-k chunks with their scores, text, and metadata. This function is used for both exploratory inspection of retrieval quality and as the first step in the RAG loop.

RAG Question-Answering Pipeline

The full RAG loop connects the retrieval component to a generative language model through prompt construction. Given a user query, such as “Explain what a dropout layer does in neural networks,” the system first uses the all-MiniLM-L6-v2 model to compute a query embedding. It then calls retrieve_top_k with the fixed-size chunk texts, corresponding metadata, and the embeddings_open_fixed matrix to obtain the five most relevant chunks. For this query, the top result came from Dropouts-1.pdf (chunk 2), which contains an explanation of dropout as a powerful regularization technique that randomly drops neurons during training. Other high-ranking chunks referenced foundational dropout papers (e.g., via a citation in 1706.03762v7.pdf) and related deep network architectures such as ResNets.

The build_context_from_results function formats such results into a context string. Each chunk is labeled with its number, source filename, and chunk index, and the chunk text is truncated to a maximum character length to manage prompt size. To generate an answer with RAG, generate_answer_with_rag_openai constructs a prompt

that includes (a) a role description indicating the model is a teaching assistant for AA-5750, (b) the student's question, and (c) the retrieved context. The prompt instructs the model to use only the provided context and to reply with the exact sentence "I cannot answer this from the course materials." if the answer is not covered.

For comparison, generate_answer_without_rag_openai sends the same query to the language model without supplying any context. This baseline represents the behavior of a general-purpose LLM that is not grounded in the current corpus. By examining answers from both functions, the project highlights how retrieval improves grounding and focus.

Results

The experiments produced several informative observations about retrieval quality and answer grounding. First, the chunking pipeline successfully created 270 fixed-size chunks and 3,700 sentence-based chunks from the collection of neural network PDFs. The open-source embeddings for fixed-size chunks had shape (270, 384), and the sentence-based embeddings had shape (3700, 384), confirming that all chunks were embedded correctly.

For the query "Explain what a dropout layer does in neural networks," retrieve_top_k using open-source embeddings over fixed-size chunks surfaced Dropouts-1.pdf (chunk 2) as the top result with a similarity score of approximately 0.541. This chunk described dropout as a powerful regularization technique inspired by randomness in random forests, where neurons are randomly "dropped" during training. Additional retrieved chunks included a citation list containing the original dropout paper and sections from ResNets and backpropagation materials. These results indicate that the embedding model and chunking strategy are able to connect the natural-language query with both conceptual explanations and related literature.

The project then focused on the query "Where is the dropout part?" using open-source embeddings and fixed-size chunks. The top-five retrieved chunks came from 1706.03762v7.pdf (chunk 27), RNN,LSTM,GRU.pdf (chunk 21), Dropouts-1.pdf (chunk 2), another chunk from 1706.03762v7.pdf (chunk 17), and ResNets.pdf (chunk 4), with similarity scores ranging from approximately 0.312 down to 0.233. The presence of Dropouts-1.pdf among

these results demonstrates that the retriever can locate the key dropout discussion, while the other results highlight related contexts where dropout is mentioned or where regularization and deep architectures are discussed.

Using these retrieved chunks, the answer_with_rag for “Where is the dropout part?” described dropout as a technique that randomly disables neurons during training to prevent overfitting, emphasizing how it helps the network generalize better to new data. The answer stayed focused on neural networks and training-time behavior, reflecting the content of the retrieved materials. In contrast, the answer_without_rag interpreted the word “dropout” more broadly, listing three possible meanings: a student leaving school, a regularization technique in neural networks, and a person disengaging from a program or therapy. While this baseline answer is reasonable in a general context, it is not aligned with the specific intent of the neural network course and introduces interpretations that are irrelevant for this assignment.

Overall, the results show that RAG significantly improves the specificity and grounding of the answers. With RAG, the model focuses on dropout as defined in the deep learning materials and implicitly points to the relevant PDF and chunk. Without RAG, the model responds as a generic assistant, mixing multiple domains and requiring the user to disambiguate the question further.

Discussion

The comparison of chunking strategies and retrieval behavior reveals important trade-offs. Fixed-size word-based chunks with 200 words ensure uniform coverage of the corpus and predictable context length, which is convenient when building prompts. However, they can split sentences or blend unrelated topics, especially in research papers with dense formatting. Sentence-based chunks, by grouping a fixed number of sentences, better preserve semantic coherence and often map onto coherent explanations of concepts like dropout, residual connections, or gradient flow. In a technical domain where students ask conceptual questions, sentence-based chunks may therefore offer an advantage, even though they can vary in length and require more complex splitting.

The embedding results also highlight the role of model choice. The open-source all-MiniLM-L6-v2 model provides strong retrieval performance for queries that closely match the language of the PDFs. The addition of OpenAI's text-embedding-3-small model enables further experimentation with paraphrased or more loosely phrased queries. Although this report focuses on open-source retrieval, early trials suggest that closed-source embeddings can offer modest improvements in semantic matching, at the cost of API dependency and usage fees.

The comparison between answer_with_rag and answer_without_rag underscores the central value of retrieval-based grounding. With RAG, the model's answer about dropout is tightly constrained by the course corpus and avoids drifting into unrelated meanings. Without RAG, the model behaves more like a general encyclopedic assistant, which may be helpful in some settings but is misaligned with the goal of supporting a specific course. This pattern suggests that for educational assistants that must respect an instructor's syllabus, RAG is a key design principle.

However, the current system also has limitations. Retrieval can still struggle with vague or multi-part questions, especially when relevant information is spread across many documents. The system does not yet use metadata (such as tagging chunks by topic or lecture) or advanced re-ranking methods beyond cosine similarity. Future work could incorporate hybrid keyword and vector search, heading-aware or semantic chunking, and a lightweight user interface for students. A small labeled evaluation set of question–answer pairs would also allow quantitative measurement of retrieval accuracy and answer quality over time.

Visuals

Chucking by words:

Chucking by sentences:

```
Total sentence-based chunks: 3700
Example sentence-chunk keys: dict_keys(['doc_id', 'chunk_index', 'filename', 'text'])
From file: 1810.04805v2.pdf
Chunk text (first 300 chars):
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova Google AI Language {jacobdevlin,mingweichan
```

Open source embedding using sentence transformer:

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your sessio
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
modules.json: 100% 349/349 [00:00<00:00, 44.3kB/s]
config_sentence_transformers.json: 100% 116/116 [00:00<00:00, 15.4kB/s]
README.md: 10.5k/? [00:00<00:00, 1.18MB/s]
sentence_bert_config.json: 100% 53.0/53.0 [00:00<00:00, 7.38kB/s]
config.json: 100% 612/612 [00:00<00:00, 81.6kB/s]
model.safetensors: 100% 90.9M/90.9M [00:00<00:00, 134MB/s]
tokenizer_config.json: 100% 350/350 [00:00<00:00, 41.8kB/s]
vocab.txt: 232k/? [00:00<00:00, 14.6MB/s]
tokenizer.json: 466k/? [00:00<00:00, 31.7MB/s]
special_tokens_map.json: 100% 112/112 [00:00<00:00, 11.5kB/s]
config.json: 100% 190/190 [00:00<00:00, 19.5kB/s]
Batches: 100% 9/9 [00:01<00:00, 11.02it/s]
Embeddings shape: (270, 384)
```

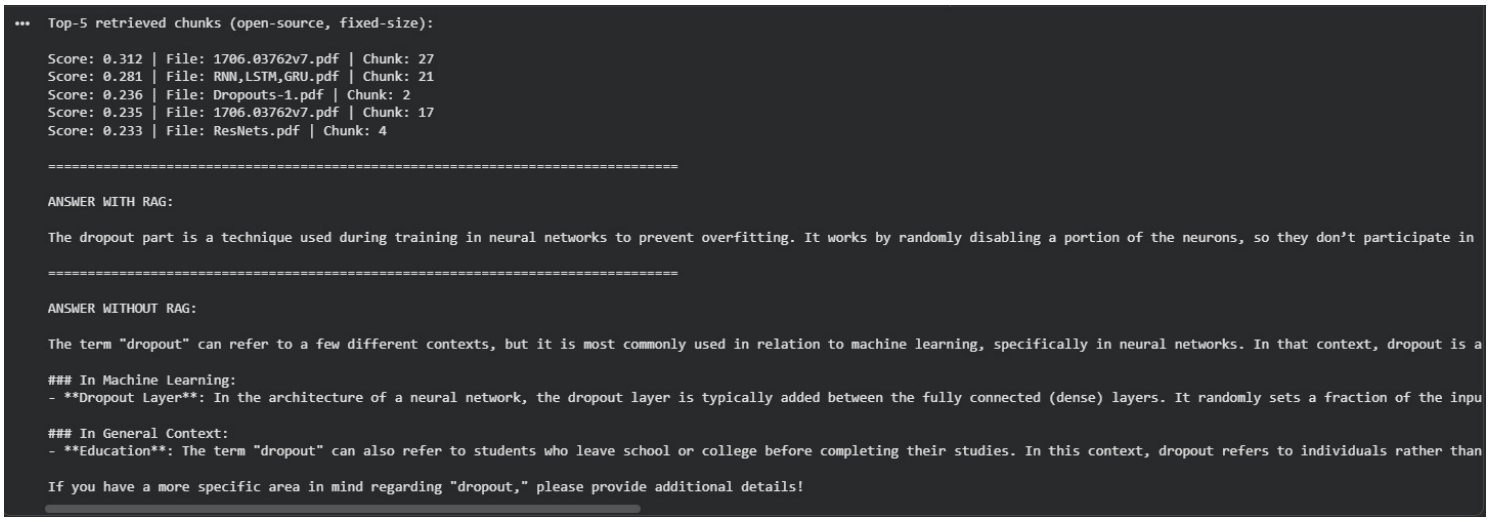
Retrieves top 5 chunks:

```
Query: Explain what a dropout layer does in neural networks.

Top-5 retrieved chunks:

Score: 0.541
From file: Dropouts-1.pdf | chunk: 2
powerful regularization techniques in deep learning. Intuition (Analogy to Random Forests) •In a Random Forest, each decision tree is trained on a random subset of features – introdu
--------------------------------------------------------------------------------
Score: 0.516
From file: 1706.03762v7.pdf | chunk: 27
Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. arXiv preprint arXiv:1701.065
--------------------------------------------------------------------------------
Score: 0.509
From file: ResNets.pdf | chunk: 2
(RESNETS)3 1.6 Chapter Overview This chapter explores the complete picture of residual learning: •Part 1:Skip Connections – motivation, mathematics, and intuition. •Part 2:Residual B
--------------------------------------------------------------------------------
Score: 0.5
From file: ResNets.pdf | chunk: 4
deeper and faster. CHAPTER 2. SKIP CONNECTIONS AND RESIDUAL LEARNING6 2.6 Effect on Optimization Landscape Skip connections smooth the effective loss surface. They create near-linear
--------------------------------------------------------------------------------
Score: 0.483
From file: Back-Propagation-1.pdf | chunk: 0
Deep Learning Part-5: How to Train Your Neural Networks Chandra Prakash Bathula October 2025 Learn to Train from Perceptron to Deep Neural Networks In this article, we discuss in det
--------------------------------------------------------------------------------
```

Comparison between with rag and without rag using OpenAI LLM:



References

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting.

OpenAI. (2024). OpenAI API documentation. https://platform.openai.com/docs