text

```
# Automated Credit Memo Knowledge Base Creation & Integration for
AWS Bedrock

*This comprehensive guide will help you create a synthetic and
extracted dataset of credit memos, process narrative chunking,
automate PDF/DOCX data extraction using LandingAI's ADE API, and
embed your examples for AWS Bedrock-powered RAG workflows.*

---

## **Step 1: Generate 50 Synthetic and Industry-Varied Credit
Memos**

1. **Prepare your synthetic credit memos with industry and risk
diversity:**
```

```
import pandas as pd
 import random

borrowers = [
 "Restaurant", "Logistics Company", "Real Estate Developer", "Retail Store", "Tech Startup",
 "Childcare Center", "Landscaping LLC", "Dental Practice", "Auto Repair Shop",
"Construction Firm",
 "Fitness Gym", "Hair Salon", "Consultancy", "Franchisee", "Farm", "Brewery", "Pharmacy",
 "Private School", "Wholesale Distributor", "Cleaning Service", "Pet Care Business", "Food
Truck"
 ]

loan_types = [
 "SBA Term Loan", "CRE Mortgage", "Revolving LOC", "Equipment Loan", "Working Capital",
 "Refinance Loan", "Startup Capital", "Inventory Loan", "Partner Buyout", "Technology
Upgrade Loan"
 ]

purposes = [
 "Expansion", "Equipment Purchase", "Working Capital", "Refinance Debt", "Acquire
Property",
 "Renovate Facility", "Seasonal Funding", "Launch New Business", "Increase Inventory",
"Marketing Campaign",
 "Technology Upgrade", "Buy out Partner"
 ]
```

```
risk_building_blocks = [
 "Strong historical cash flow and low leverage.",
 "Revenue concentration risk with two major customers.",
 "Collateral consists of recently appraised real estate and new equipment.",
 "Business experienced a recent operating loss due to market disruption.",
 "DSCR above policy minimum; liquidity maintained.",
 "Management team highly experienced but thin succession plan.",
 "Negative trends in same-store sales, inventory turnover below average.",
 "Personal and business credit scores above bank thresholds.",
 "Significant contingent liabilities from outstanding legal claims.",
 "Facility lease expiring next year, renewal terms uncertain.",
 "Business model proven over a decade with consistent profitability.",
 "Applicant lacks relevant industry experience.",
 "Outstanding tax liens discovered in credit review.",
 "Growth tied heavily to local market conditions.",
 "Recent investment in technology improving operations.",
 "Environmental risk due to property location.",
 "Business heavily leveraged; recent refinancing attempts unsuccessful.",
 "Multiple missed payments on previous loans.",
 "New management team, limited track record in sector.",
 "Market position strengthened by recent competitive exits."
 ]

recommendations = {
 5: "Approve on preferred terms; risk is minimal.",
 4: "Approve, collateral covers downside; monitor quarterly.",
 3: "Approve with tight covenants and ongoing review.",
 2: "Conditional approval; reduce loan size or add guarantee.",
 1: "Decline; recommend revisiting after operational improvements."
 }

score_map = {
 1: "Ugly: Severe cash flow issues, poor credit, unproven business.",
 2: "Bad: Weak financials, some collateral, questionable repayment.",
 3: "OK: Average risk, adequate financials, fair collateral.",
 4: "Good: Solid cash flow, strong collateral, proven business.",
 5: "Excellent: Exceptional financials, low risk, excellent reputation."
 }

memos = []
 for i in range(1, 51):
 score = random.randint(1, 5)
 blocks = random.sample(risk_building_blocks, k=random.randint(3, 5))
 risk_narrative = f"{score_map[score]} " + " ".join(blocks)
 memo = {
 "memo_id": f"CM-{i:03}",
 "title": f"{random.choice(borrowers)} seeking {random.choice(loan_types)} [{score}]",
 "borrower": random.choice(borrowers),
```

```
 "loan_type": random.choice(loan_types),
 "amount": f"${random.randint(100_000, 2_000_000):,}",
 "purpose": random.choice(purposes),
 "risk_analysis": risk_narrative,
 "recommendation": recommendations[score],
 "score": score
 }
 memos.append(memo)

df = pd.DataFrame(memos)
 df.to_csv("credit_memo_dataset_diverse.csv", index=False)

text

---

## **Step 2: Chunking Longer Risk Narratives**

*To prepare content for semantic search and embedding, split long
texts into manageable chunks.*



def chunk_text(text, max_length=500):
 sentences = text.split('. ')
 chunks, current = [], ""
 for s in sentences:
 if len(current) + len(s) < max_length:
 current += s + ". "
 else:
 chunks.append(current.strip())
 current = s + ". "
 if current:
 chunks.append(current.strip())
 return chunks
```

Apply chunking to risk_analysis for all memos:

```
chunked_memos = []
 for _, row in df.iterrows():
 chunks = chunk_text(row['risk_analysis'], max_length=300)
 for idx, chunk in enumerate(chunks):
 chunked_memos.append({
```

```
 "memo_id": f"{row['memo_id']}-{idx+1}",
 "title": row['title'],
 "risk_chunk": chunk,
 "score": row['score']
 })

df_chunks = pd.DataFrame(chunked_memos)
 df_chunks.to_csv("credit_memo_risk_chunks.csv", index=False)

text

---

## **Step 3: Automate Extraction from Downloaded PDFs/DOCXs Using LandingAI's ADE API**

*LandingAI's ADE API (Automated Document Extraction) extracts structured field data from financial documents.*

### **Instructions:**

1. **Set up LandingAI ADE account** (follow instructions at https://landing.ai/document-extraction/)
2. **Prepare document files:** e.g. download sample memos/templates in PDF/DOCX format.
3. **Install `requests` and `pandas` if not present:**


pip install requests pandas

text

4. **Send files to ADE API and collect responses:**



import requests
 import pandas as pd

ade_endpoint = "https://api.landing.ai/ade/extract"
 api_key = "YOUR_API_KEY"

def extract_pdf_docx(filepath):
 with open(filepath, "rb") as f:
 response = requests.post(
 ade_endpoint,
```

```
 headers={"Authorization": f"Bearer {api_key}"},
 files={"file": f}
 )
 return response.json()

files = ["sample_memo1.pdf", "sample_memo2.docx"] # Add paths to your files
 extracted_records = []

for file in files:
 data = extract_pdf_docx(file)
 # Example: data['fields'] contains [{'field_name': ..., 'field_value': ...}]
 memo = {field["field_name"]: field["field_value"] for field in data.get("fields", [])}
 memo["source_file"] = file
 extracted_records.append(memo)

df_extracted = pd.DataFrame(extracted_records)
 df_extracted.to_csv("extracted_credit_memos.csv", index=False)

text
*Note: Adjust parsing logic based on the ADE API's actual response
schema.*

---

## **Step 4: Integrate Data with AWS Bedrock—Embedding & KB
Construction for RAG**

### **A. Data Preparation**

1. Use your chunked or extracted memos saved in CSV
(`credit_memo_risk_chunks.csv` or `extracted_credit_memos.csv`).
2. Join all memo data into a single corpus if needed.

### **B. Generate Embeddings Using AWS (Bedrock-Compatible)**

*Use AWS SageMaker and a HuggingFace model locally, or call an
embedding API.*



from sentence_transformers import SentenceTransformer
 import pandas as pd

# Read CSV of chunks
```

```
df_chunks = pd.read_csv("credit_memo_risk_chunks.csv")
 model = SentenceTransformer('all-mpnet-base-v2')

df_chunks['embedding'] = df_chunks['risk_chunk'].apply(lambda x: model.encode(x).tolist())
 df_chunks.to_csv('credit_memo_chunks_with_embeddings.csv', index=False)
```

text

C. Upload to Aurora PostgreSQL with pgvector (for semantic DB)

```
CREATE EXTENSION IF NOT EXISTS vector;

CREATE TABLE memo_kb_chunks (
 id SERIAL PRIMARY KEY,
 memo_id VARCHAR(16),
 title TEXT,
 risk_chunk TEXT,
 score INT,
 embedding VECTOR(768)
 );
```

text

Python insertion example:

```
import psycopg2
 df = pd.read_csv("credit_memo_chunks_with_embeddings.csv")
 conn = psycopg2.connect(host="AWS_AURORA_HOST", dbname="YOUR_DB",
user="USER", password="PASS")
 cur = conn.cursor()

for _, row in df.iterrows():
 cur.execute(
 "INSERT INTO memo_kb_chunks (memo_id, title, risk_chunk, score, embedding) VALUES
(%s, %s, %s, %s, %s)",
 (row['memo_id'], row['title'], row['risk_chunk'], int(row['score']), row['embedding'])
 )
 conn.commit()
 cur.close()
 conn.close()
```

text

D. Semantic Search & RAG for AWS Bedrock

1. **On query**: Embed user prompt (using same model as above).
2. **Semantic SQL search for retrieval:**
    ```
    SELECT risk_chunk, title
    FROM memo_kb_chunks
    ORDER BY embedding <-> '[QUERY_EMBEDDING]'
    LIMIT 5;
    ```
3. **Inject retrieved KB chunks into Bedrock LLM prompt:**
    ```
    System Prompt:
    "You are a commercial banking AI. Using these credit memo risk narratives and context, draft a new memo or respond to questions.
Context:
    [kb_chunk_1]
    [kb_chunk_2]
    ...
    User Instruction: ..."

    # Sample AWS Bedrock call (Python, boto3)
    import boto3
    bedrock = boto3.client('bedrock-runtime')
    response = bedrock.invoke_model(
        modelId='anthropic.claude-v2',
        contentType='application/json',
        body={
            "prompt": "You are a commercial banking AI. ... [context here] ...",
            "max_tokens_to_sample": 1024
        }
    )
    ```

Summary Table

Step	Script/Process	Output
1. Generate memos	Python (variety, structure)	`credit_memo_dataset_diverse.csv`
2. Chunk narratives	Python `chunk_text`	`credit_memo_risk_chunks.csv`
3. Extract PDF/DOCX	Python + LandingAI ADE API	`extracted_credit_memos.csv`
4. Generate Embeddings	SentenceTransformers (or AWS API)	`credit_memo_chunks_with_embeddings.csv`
5. Upload to Aurora	SQL `CREATE TABLE`, psycopg2 Python insert	DB table: `memo_kb_chunks`
6. Semantic Retrieval	SQL semantic search + Bedrock API call	Retrieved context for LLM

Best Practices

- Always preprocess and redact sensitive content.
- Enhance your document diversity by periodically extracting from new samples and templates.
- Review semantic search results for relevancy, diversity, and coverage.
- Update embedding model as needed to keep up-to-date with Bedrock-integration best practices.

This guide provides a full, automated pipeline—from document generation and extraction to AWS-based semantic search and context injection for next-gen credit memo AI workflows!