■ **Course:** CSCI E-103: Reproducible Machine Learning

■ **Week:** Lecture 11

■ **Instructors:** Anindita Mahapatra & Eric Gieseke

■ **Objective:** Master the fundamentals of Large Language Models (LLMs), Retrieval Augmented Generation (RAG), and AI Agents for enterprise applications

Contents

1 Introduction: The LLM Revolution

This lecture covers one of the most transformative technologies in modern computing: **Large Language Models (LLMs)** and **AI Agents**. We'll explore how these technologies have evolved, how to deploy them effectively in enterprise settings, and how to mitigate their inherent risks.

Lecture Overview

Why This Matters:

- LLMs are reaching a **tipping point** in capability and accessibility
- **AGI (Artificial General Intelligence)** predictions suggest human-level AI could arrive as soon as 2026
- Every company will become a **data and AI company** first, with their core business layered on top
- Understanding LLM deployment is now an essential skill for data engineers and ML practitioners

1.1 Learning Objectives

By the end of this lecture, you will be able to:

1. **Understand** the evolution from rule-based systems to Generative AI
2. **Explain** key LLM terminology (RAG, Fine-tuning, Embeddings, Hallucination)
3. **Design** LLM deployment strategies based on cost-quality tradeoffs
4. **Implement** RAG systems for enterprise applications
5. **Build** AI agents using Databricks tools (Genie, Agent Bricks, AI Gateway)
6. **Manage** LLM risks including hallucinations, bias, and security vulnerabilities

2 The Evolution of AI: From Rules to Generation

2.1 The AI Technology Stack

AI technologies form a hierarchical relationship, with each layer building upon the previous:

Layer	Description
Artificial Intelligence	The broadest category: any system that mimics human intelligence
Machine Learning	Systems that learn from data without explicit programming
Deep Learning	Neural networks with multiple layers (inspired by the human brain)
Generative AI	AI that creates *new* content rather than just classifying existing data

Key Information

Key Distinction - Traditional AI vs. Generative AI:

- **Traditional AI:** ”Is this email spam or not spam?” (classification)
- **Generative AI:** ”Write me a professional email to decline a meeting” (creation)

Generative AI goes beyond analysis—it produces *new* data based on patterns learned from existing data.

2.2 Types of Generative AI Models

Definition:

LLM (Large Language Model) A neural network trained on vast amounts of text data to understand and generate human language. Examples include GPT-4, Claude, Gemini, and Llama.

Analogy: Think of an LLM as a student who has read every book in the world’s largest library and can now discuss any topic or write on any subject.

Definition:

GAN (Generative Adversarial Network) A model architecture primarily used for generating images and videos. Used in applications like:

- **Deepfakes:** Creating realistic fake videos of people
- **Style Transfer:** Transforming photos into paintings
- **Image Generation:** Creating photorealistic images from scratch

Definition:

Diffusion Models Advanced generative models that create content by gradually removing noise from random data. Powers:

- **DALL-E:** Text-to-image generation
- **Stable Diffusion:** Open-source image generation
- **Video Generation:** Creating lip-synced avatars

3 LLM Capabilities and Enterprise Use Cases

3.1 What Can LLMs Do?

LLMs have fundamentally changed what’s possible with AI:

1. **Democratize Knowledge Access**
   - All human knowledge encoded in a single model
   - Accessible in any language (multilingual capabilities)
   - Available to anyone with internet access
2. **Process Unstructured Data**
   - Transcribe video/audio to text

- Summarize documents, meeting minutes, patents
- Extract structured data from free-form text

3. **Improve Knowledge Worker Productivity**
   - Code generation and debugging
   - Legal document review
   - Customer feedback analysis
   - Marketing trend identification
4. **Enable Self-Improvement**
   - Models can evaluate their own outputs
   - Tune product recommendations based on feedback
   - Generate training data for other models

Example:

Real-World LLM Applications (Demo) The lecture demonstrated a multilingual AI assistant with multiple capabilities:

Features Demonstrated:

- **Voice Interaction:** Users speak questions in natural language
- **Multilingual Response:** System answers in Russian, Spanish, etc.
- **RAG Integration:** Answers based on specific business knowledge base
- **Lip-Synced Avatar:** Diffusion model generates realistic speaking animation
- **E-commerce Search:** ”blue sarees” search understands intent despite misspelling
- **Virtual Try-On:** Diffusion model shows how clothes look on uploaded photos

3.2 The Urgency to Adopt LLMs

Important:

Why Companies Must Move Fast

1. **Competitive Pressure:** Early adopters gain significant advantages
2. **Accessibility:** LLMs are now economical enough for any business
3. **Integration Need:** Must connect LLMs to existing company data
4. **Security Requirements:** Need to customize and secure AI for enterprise use

4 Essential LLM Terminology

Understanding these terms is crucial for working with modern AI systems:

5 LLM Deployment Strategies: The Maturity Curve

Choosing the right deployment approach depends on your use case, budget, and accuracy requirements.

Term	Definition and Example
Hallucination	When the model confidently generates false information. *Example:* Generating fake URLs that look real but don't exist
Temperature	Controls randomness in outputs (0 = deterministic, 1 = creative) Setting to 0 reduces hallucinations but may limit creativity
Grounding	Connecting AI responses to real-world facts and data Ensures answers are based on verifiable information
Prompt Engineering	Crafting instructions to get desired behavior from LLMs *Example:* "You are a helpful restaurant server..."
Zero-Shot Learning	Asking the model to perform a task with just a prompt No examples provided—relies entirely on pre-training
Few-Shot Learning	Providing examples in the prompt to guide the model Generally improves accuracy over zero-shot
Chain of Thought	Instructing the model to show its reasoning steps Improves accuracy and interpretability
Modality	Type of data: text, image, audio, or video **Multimodal** models handle multiple types (e.g., GPT-4o)
Transformer	Neural network architecture underlying modern LLMs Components: Encoder, Decoder, Embeddings
RLHF	Reinforcement Learning from Human Feedback Humans rate outputs to improve model behavior

Table 1: *Essential LLM Terminology*

5.1 The Four Levels of LLM Customization

LLM Deployment Strategies (Easy to Hard)

Level 1: Prompt Engineering

- **What:** Craft careful instructions for existing models
- **Cost:** Very low (API costs only)
- **Data Required:** None
- **Quality:** Basic—limited by model's training cutoff
- **Best For:** General tasks, quick prototypes

Level 2: RAG (Retrieval Augmented Generation) — RECOMMENDED

- **What:** Search your data, feed relevant chunks to the model

Warning

Key Insight: Most organizations should start with RAG, not fine-tuning or pre-training. RAG provides the best cost-quality tradeoff and allows you to:

- Include the latest information (no training cutoff)
- Reduce hallucinations (answers grounded in your documents)
- Maintain data privacy (your data stays in your database)
- Update knowledge easily (just update the vector database)

6 RAG: Retrieval Augmented Generation

RAG is the most cost-effective and practical approach to customizing LLMs for enterprise use.

6.1 Why RAG Is Necessary

LLMs have two fundamental limitations:

1. **Training Cutoff:** Models don't know anything after their training date
   - GPT-4 doesn't know who won yesterday's election
   - Can't answer questions about your company's latest policy
2. **No Access to Private Data:** Models never saw your internal documents
   - Can't answer "What's our return policy?"
   - Can't help with company-specific procedures

Key Information

The RAG Solution:
RAG gives the LLM an "open book test"—instead of relying solely on memorized knowledge, we search relevant documents and include them in the prompt. The model then answers based on this retrieved context.

6.2 How RAG Works: The Complete Pipeline

Definition:

RAG Architecture **Phase 1: Data Ingestion (Offline, One-Time Setup)**

1. **Document Collection:** Gather all relevant documents (PDFs, manuals, policies)
2. **Chunking:** Split documents into smaller pieces (800-1200 characters optimal)
3. **Embedding:** Convert text chunks into numerical vectors (embeddings)
4. **Storage:** Store embeddings in a Vector Database (Pinecone, Chroma, etc.)

Phase 2: Query Processing (Real-Time)

1. **User Query:** User asks a question
2. **Query Embedding:** Convert the question to a vector

3. **Similarity Search:** Find document chunks with similar vectors
4. **Context Augmentation:** Add retrieved chunks to the prompt
5. **LLM Generation:** Model generates answer using retrieved context
6. **Response:** Return answer to user (optionally with source citations)

Example:

RAG Query Flow **User Question:** ”What is our company’s remote work policy?”
Behind the Scenes:

1. Question converted to vector: [0.23, -0.45, 0.67, ...]
2. Vector DB searched for similar vectors
3. Top 3 most relevant document chunks retrieved:
   - HR Policy Manual, Section 4.2 (similarity: 0.92)
   - Employee Handbook, Chapter 7 (similarity: 0.88)
   - COVID-19 Work Guidelines (similarity: 0.85)
4. Prompt sent to LLM:

   ```
   Based on the following documents, answer the user's question.

   Documents:
   [Retrieved chunks here...]

   Question: What is our company's remote work policy?
   ```

5. LLM generates response grounded in the documents

6.3 RAG Best Practices

Parameter	Recommendation
Chunk Size	800-1200 characters (optimal for semantic coherence)
Chunk Overlap	100-200 characters (ensures context isn’t lost at boundaries)
Number of Retrieved Chunks	3-5 (balance between context and token limits)
Embedding Model	OpenAI Ada, Sentence Transformers, Cohere Embed
Vector Database	Pinecone, Chroma, Milvus, PostgreSQL with pgvector

Table 2: *RAG Configuration Best Practices*

Important:

Common RAG Pitfall: Version Management **Problem:** When policies are updated, old versions remain in the vector database. The LLM might retrieve outdated information.

Solutions:

- Replace entire documents when updating (full CRUD support)
- Add metadata timestamps and filter by date
- Include version numbers in document content
- Periodic reindexing to remove stale content

7 Databricks AI Platform: Tools and Capabilities

Databricks provides an integrated platform for building, deploying, and managing LLM applications.

7.1 Platform Architecture Overview

Component	Purpose
Unity Catalog	Governance for all assets (tables, models, functions, vector indices)
Delta Sharing	Zero-copy data sharing without ETL
Feature Store	Store and serve ML features
Model Registry	Version and manage ML models (part of MLflow)
Vector Search	Managed vector database for RAG applications
AI Gateway	Central entry point for all models with security, logging, rate limiting
Lakehouse Apps	Interactive applications built on top of data and models

7.2 Genie: Natural Language to SQL

Definition:

Genie Genie is an agentic BI tool that allows users to query structured data using natural language.

How It Works:

1. User asks: ”What were our top 5 products by sales last month?”
2. Genie examines table metadata (schemas, column names)
3. Generates SQL: `SELECT name, SUM(sales) FROM products...`
4. Executes query and returns results in natural language

Key Advantage: Because Genie generates SQL from metadata (not data), it hallucinates less than general LLMs. The schema is deterministic—a column either exists or it doesn’t.

```
-- User Question: "Show average premium by occupation and risk level"
-- Genie generates:
```

```sql
SELECT
    occupation_type,
    risk_level,
    AVG(premium_amount) as avg_premium
FROM insurance_policies
GROUP BY occupation_type, risk_level
ORDER BY avg_premium DESC;
```

Listing 1: Genie-Generated SQL Example

7.3 Agent Bricks: No-Code RAG Agent Builder

Definition:

Agent Bricks Agent Bricks is like AutoML for RAG agents—build production-ready chatbots without coding.

Steps to Create a Knowledge Assistant:

1. Upload documents to Unity Catalog Volume
2. Select the volume path in Agent Bricks
3. Provide a name and description
4. Click "Create Agent"
5. System automatically: chunks documents, creates embeddings, builds vector index

Use Cases:

- Customer support chatbots
- Internal policy assistants
- Technical documentation search
- HR FAQ systems

7.4 AI Functions: SQL-Native AI

Execute LLM capabilities directly within SQL queries:

```sql
-- Analyze sentiment and summarize customer reviews
SELECT
    review_id,
    review_text,
    ai_analyze_sentiment(review_text) AS sentiment,
    ai_summarize(review_text, 20) AS summary
FROM customer_reviews
WHERE created_date > '2025-01-01';

-- Extract specific information from text
SELECT
    ai_extract(contract_text, 'termination_clause') AS termination_clause,
    ai_extract(contract_text, 'payment_terms') AS payment_terms
FROM legal_contracts;
```

Listing 2: AI Functions in SQL

7.5 AI Gateway: Central Model Management

Key Information

AI Gateway provides a unified entry point for all model interactions:

- **Rate Throttling:** Prevent API abuse and control costs
- **Credential Management:** Centralized API key handling
- **Payload Logging:** Record all requests/responses for auditing
- **A/B Testing:** Route traffic between model versions
- **Guardrails:** Content filtering and safety checks

Analogy: AI Gateway is like Amazon API Gateway, but specifically designed for ML model endpoints.

7.6 Unity Catalog Functions as Agent Tools

You can register SQL functions in Unity Catalog and use them as tools for AI agents:

```sql
-- Create a function that agents can call
CREATE FUNCTION catalog.schema.get_client_quote(client_name STRING)
RETURNS TABLE (
    quote_id STRING,
    client STRING,
    coverage DECIMAL,
    premium_estimate DECIMAL,
    product_option STRING,
    health_rating STRING
)
COMMENT 'Returns all quote details for a specified client'
AS
SELECT
    quote_id, client, coverage,
    premium_estimate, product_option, health_rating
FROM insurance_quotes
WHERE client = client_name;
```

Listing 3: UC Function as Agent Tool

Example:

Multi-Agent Supervisor The most advanced pattern is a **Multi-Agent Supervisor** that orchestrates multiple specialized agents:

Architecture:

1. **Supervisor Agent:** Understands user intent and routes requests

2. **Genie Agent:** Handles structured data queries (SQL)
3. **Knowledge Assistant:** Handles unstructured data (RAG)
4. **UC Functions:** Execute specific business logic
5. **External MCP Servers:** Connect to external services

Example Flow:

- User: ”What’s the risk score for John Smith and explain our underwriting policy?”
- Supervisor: Routes ”risk score” to Genie, ”underwriting policy” to Knowledge Assistant
- Both agents execute in parallel
- Responses combined and returned to user

8 LLM Risks, Limitations, and Mitigation

With great power comes great responsibility. LLMs introduce new categories of risk that must be actively managed.

8.1 Hallucination: The Confident Liar

Definition:

Hallucination When an LLM generates plausible-sounding but factually incorrect information with complete confidence.

Why It Happens:

- LLMs are trained to generate *fluent* responses, not necessarily *true* ones
- Models are rewarded for providing answers, even when they should say ”I don’t know”
- Pattern matching can produce text that looks correct but isn’t

Example: When asked to generate image URLs, ChatGPT confidently produces URLs that look legitimate but lead to 404 errors.

Warning

Can Hallucination Be Eliminated?

No—hallucination is intrinsic to the generative nature of LLMs. It’s like asking if floating-point rounding errors can be eliminated. However, it can be **significantly reduced**:

- Set temperature to 0 (most deterministic output)
- Use RAG to ground responses in verified documents
- Implement chain-of-thought prompting for reasoning transparency
- Add guardrail models to verify outputs
- Design prompts that explicitly allow ”I don’t know” responses

8.2 Bias: Reflections of Training Data

- **Source:** Models inherit biases present in their training data

- **Example:** A model trained primarily on US medical data may give incorrect recommendations for diseases more prevalent in other regions
- **Impact:** Can perpetuate discrimination, exclusion, and unfair treatment

Mitigation Strategies:

1. Ensure training data diversity
2. Explicitly document known limitations
3. Implement fairness testing and auditing
4. Use human review for high-stakes decisions

8.3 Security Threats

Threat	**Description**	**Mitigation**
Prompt Injection	Malicious inputs that trick the model into ignoring instructions	Input validation, prompt sanitization
Jailbreaking	Bypassing safety guardrails to generate harmful content	Multiple layers of content filtering
Data Exfiltration	Model reveals sensitive training data	Data sanitization, PII masking
Multilingual Attacks	Using non-English prompts to bypass safety filters	Multilingual safety testing

Table 3: *LLM Security Threats and Mitigations*

8.4 Privacy and Regulatory Compliance

Important:

EU AI Act Europe's **EU AI Act** mandates disclosure of:

- Training data sources
- Compute resources used
- Model deployment details
- Known limitations and biases

Implication: Significant documentation overhead, but ensures responsible AI development.

8.5 Ethical Considerations and ESG

Definition:

ESG (Environmental, Social, Governance) Companies are increasingly evaluated on their ESG scores:

- **Environmental (E):** LLM training consumes enormous energy. A single GPT-4 training run can emit hundreds of tons of CO2.

- **Social (S):** AI systems that discriminate or generate toxic content harm society.
- **Governance (G):** Proper oversight, audit trails, and accountability mechanisms.

Key Insight: Laws can be enforced, but ethics must be culturally adopted. As AI practitioners, we have a collective moral responsibility.

9 The Future: AGI, Superintelligence, and Societal Impact

9.1 The Path to AGI

Definition:

AGI and Superintelligence

- **AGI (Artificial General Intelligence):** AI as capable as humans across all cognitive tasks. Predictions suggest this could arrive as early as 2026.
- **Superintelligence:** AI that surpasses human intelligence by orders of magnitude. Once achieved, could rapidly self-improve.

Warning

Book Recommendation: "Superintelligence" by Nick Bostrom

This book explores what happens when AI becomes smarter than humans:

- The "alignment problem": ensuring AI goals match human values
- Existential risks from uncontrolled AI development
- The importance of safety research before capability research

As practitioners building these systems, we have a responsibility to understand these risks.

9.2 Impact on Employment

Key Information

The Changing Job Landscape:

- **Entry-level jobs:** Most at risk—routine cognitive tasks easily automated
- **Experienced practitioners:** Demand increasing—someone must build and maintain AI systems
- **New roles emerging:** Prompt engineers, AI ethics officers, AI trainers

The Paradox: If entry-level jobs disappear, how do workers gain experience to become senior practitioners? This is a fundamental societal challenge we must address.

9.3 AI as a Democratizing Force

Despite risks, LLMs also offer tremendous positive potential:

1. **Education:** AI tutors available 24/7, personalized to each student (Khan Academy's vision)
2. **Knowledge Transfer:** Bridging the gap between retiring domain experts and new workers

3. **Global Access:** Multilingual capabilities enable knowledge access regardless of native language
4. **Government Efficiency:** Albania using LLMs to reduce corruption in government contracting
5. **Healthcare:** Democratizing access to medical knowledge in underserved regions

10 AI Maturity Curve: Building Enterprise AI Capability

10.1 The Progression from BI to AI

Stage	Description	Complexity
BI/Dashboards	Traditional reporting: what happened?	Low
AI Functions	SQL-native AI: sentiment, summarization	Medium-Low
Knowledge Agents	RAG-based Q&A on documents	Medium
Multi-Tool Agents	Agents that call multiple functions/APIs	Medium-High
Multi-Agent Orchestration	Supervisor routing to specialized agents	High

10.2 Use Case Selection: Start Smart

Important:

Selecting Your First LLM Use Case Your first LLM project determines organizational adoption momentum. Choose wisely:

Ideal First Use Case:

- **High Feasibility:** Technical complexity is manageable
- **High Value:** Clear ROI that stakeholders can see
- **Low Risk:** Mistakes don't cause major harm
- **Reusable:** Learnings apply to future projects

Bad First Use Case:

- The "hottest" or most ambitious idea
- Customer-facing without extensive testing
- Mission-critical with no fallback

10.3 Key Principles for Enterprise AI

1. **Every company will be a data and AI company first**—core business becomes secondary
2. **Differentiation comes from your data**, not the model—everyone can use ChatGPT
3. **Many small, purpose-built models** often beat one giant model
4. **Models are improving and getting cheaper**—what's impossible today may be trivial tomorrow
5. **Benefits outweigh risks**, but only with proper guardrails

11 Practical Lab Guide: Building Agents in Databricks

11.1 Lab Overview

The lecture included hands-on demonstrations of building three types of agents:

1. **Genie Space:** Natural language queries on structured insurance data
2. **Knowledge Assistant:** RAG agent for underwriting guidelines PDF
3. **Multi-Agent Supervisor:** Orchestrating multiple specialized agents

11.2 Step-by-Step: Creating a Knowledge Assistant

```python
# Step 1: Get catalog and schema from widgets
catalog_name = dbutils.widgets.get("catalog_name")
schema_name = dbutils.widgets.get("schema_name")

# Step 2: Create schema if not exists
spark.sql(f"CREATE SCHEMA IF NOT EXISTS {catalog_name}.{schema_name}")

# Step 3: Upload documents to Volume
# Documents should be in: /Volumes/{catalog}/{schema}/{volume_name}/

# Step 4: Create tables and functions
spark.sql(f"""
CREATE TABLE IF NOT EXISTS {catalog_name}.{schema_name}.insurance_quotes (
    quote_id STRING,
    client STRING,
    coverage DECIMAL(10,2),
    premium_estimate DECIMAL(10,2),
    product_option STRING,
    health_rating STRING
)
""")
```

Listing 4: Setting Up the Data

11.3 Agent Testing in Playground

Once your agent is created, use the Playground for testing:

1. Navigate to the agent in the Databricks UI
2. Click ”Open in Playground”
3. Enter test questions
4. Review:
   - Response content and accuracy
   - Token usage and latency
   - Source citations (for RAG)

- Chain of thought reasoning (if enabled)

5. Enable "AI Judge" for automated evaluation
6. Compare responses across different models

Example:

Playground Testing **Question:** "How are applications categorized into preferred plus, preferred, standard, or substandard?"
Agent Response: "The system uses a multi-tier risk classification system to categorize applicants based on their overall risk profile..."
Citations:

- Page 1: Risk Classification Criteria
- Page 1: Underwriting Process Flow

Metrics:

- Tokens used: 847
- Latency: 2.3 seconds
- AI Judge score: 4.5/5

12 Quick Summary: One-Page Review

Key Summary

Key Takeaways from Lecture 11:

1. **AI Evolution:** Rule-based → ML → Deep Learning → **Generative AI (LLMs)**
2. **LLM Deployment Strategy:** Start with **RAG** (not fine-tuning or pre-training)
   - Most cost-effective approach
   - Grounds responses in your data
   - Reduces hallucinations
   - Easy to update knowledge
3. **RAG Pipeline:** Chunk → Embed → Store in Vector DB → Retrieve → Generate
4. **Databricks Tools:**
   - **Genie:** Structured data (SQL), high accuracy
   - **Agent Bricks:** Unstructured data (PDF), rapid prototyping
   - **AI Gateway:** Security, logging, rate limiting
   - **AI Functions:** SQL-native AI capabilities
   - **UC Functions:** Custom tools for agents
5. **Critical Risks:**
   - Hallucination (false confidence)
   - Bias (discriminatory outputs)
   - Security (prompt injection, data leakage)

- Privacy (PII exposure)

6. **Mitigation Strategies:**
   - RAG for grounding
   - Temperature = 0 for determinism
   - Guardrail models for verification
   - Human-in-the-loop for high stakes
   - RLHF for continuous improvement
7. **Future Outlook:**
   - AGI potentially by 2026
   - Every company becomes AI-first
   - Differentiation through proprietary data
   - Ethical AI is mandatory, not optional

13 Frequently Asked Questions (From Class Discussion)

Q: Are hallucinations intrinsic to LLMs, or can they be eliminated?

A: Hallucinations are inherent to the generative nature of LLMs. The models are trained to produce fluent outputs, not necessarily true ones. While they can be significantly reduced (RAG, temperature=0, careful prompting), they likely cannot be completely eliminated. Think of it like irreducible error in traditional ML—some error will always remain.

Q: How do we handle document versioning in RAG?

A: This is a complex problem. If you have Policy v1, v2, and v3 all indexed, the model might retrieve outdated information. Solutions:

- Replace entire documents when updating (full CRUD)
- Add timestamp metadata and filter by recency
- Use relationship graphs to link document versions
- Explicitly include version numbers in your documents

Q: Why does ChatGPT seem to ”remember” me across sessions?

A: Modern LLMs like ChatGPT store conversation histories and can summarize past interactions. They don't have true memory, but they can retrieve and reference previous conversations to provide more personalized responses. This feels like memory but is actually sophisticated context retrieval.

Q: What are we doing to prevent prompt injection and jailbreaks?

A: This is the ”alignment problem”—ensuring models behave as intended. Current approaches include:

- Guardrails and content filtering
- Input validation and sanitization

- Using larger models as judges/guards
- RLHF to train models to refuse malicious requests
- Regulatory frameworks (EU AI Act)

As practitioners, we have a responsibility to implement these safeguards and advocate for responsible AI development.