DATA+AI SUMMIT BY databricks

BUILDING ENTERPRISE-GRADE GENERATIVE AI APPLICATIONS WITH MLFLOW AND DATABRICKS VECTOR SEARCH

Denis Kamotsky and Pulkit Chadha
06/11/2024

ABOUT US

Denis Kamotsky

- Principal Software Engineer
- Part of Corning since 2020
- Focused on ML Engineering
- Interested in NLP and information retrieval

Pulkit Chadha

- Senior Enterprise Solutions Architect at Databricks
- Part of Databricks since 2021
- Author of “Data Engineering with Databricks Cookbook”

Agenda

- Mosaic AI Overview
- About Corning
- GenAI at Corning
- Corning + Databricks Journey on GenAI
- Future Direction of Corning + Databricks

Mosaic AI

The Architecture View

Gen AI Architecture Patterns

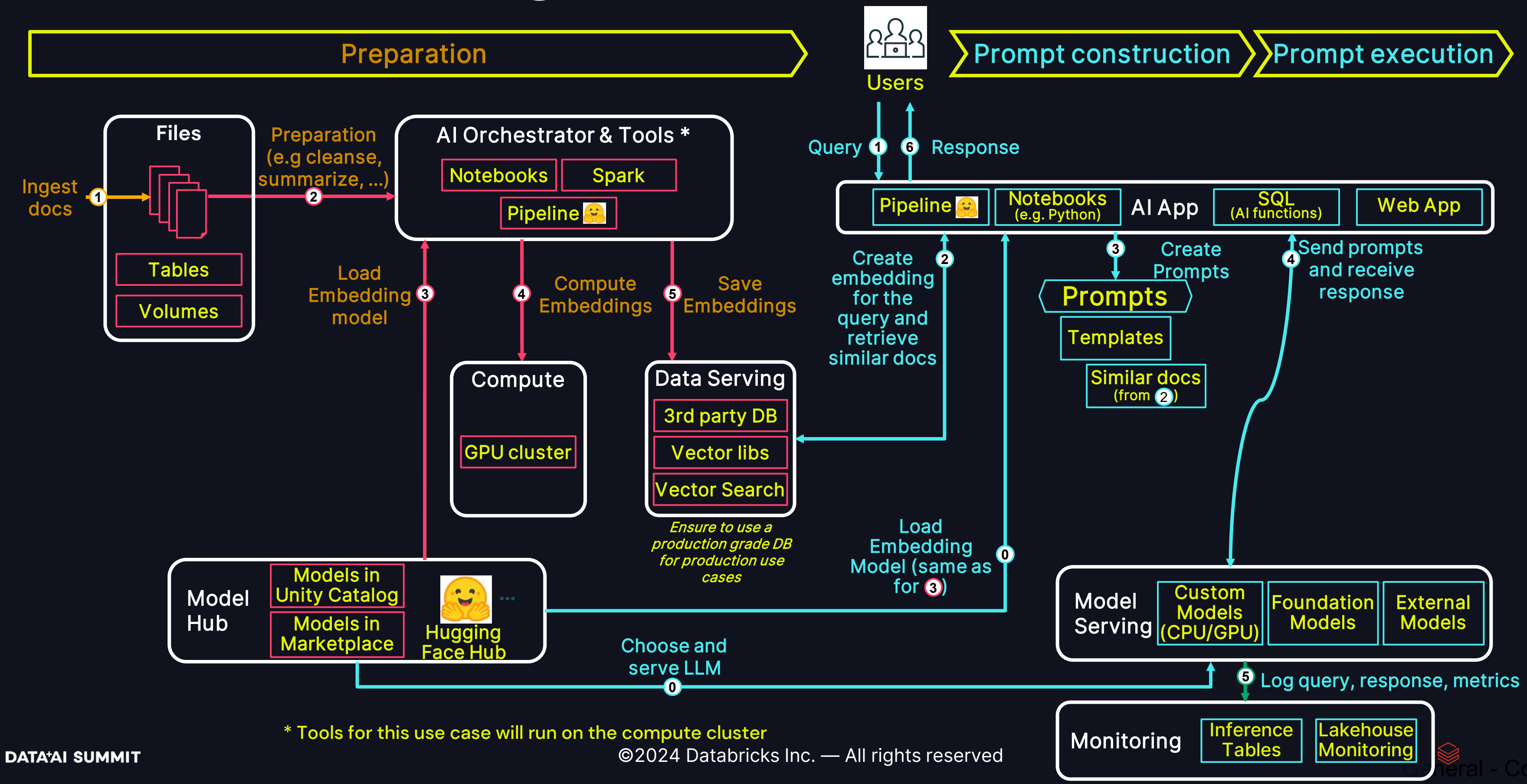
Databricks Mosaic AI is the only provider that enables every architectural pattern

Pattern 1	Pattern 2	Pattern 3	Pattern 4	Pattern 5
Use An Existing LLM	Customize LLM with Data:			Build your own LLM: Pre-training
	RAG*	Fine-Tuning	Fine-Tuning + RAG*	
• Model Serving • MLflow • DBRX (or customer's chosen model)	• Model Serving • Vector Search • MLflow • DBRX (or customer's chosen model) • [Mosaic AI RAG Framework] (coming soon)	• Foundation Model Adaptation (Private Preview) • Model Serving • MLflow • DBRX (or customer's chosen model) • Model Evaluation (coming soon)	• Foundation Model Adaptation (Private Preview) • Model Serving • Vector Search • MLflow • DBRX (or customer's chosen model) • Model Evaluation (coming soon)	• Foundation Model Training • Model Serving • DBRX (or customer's chosen model)

Unity Catalog | Lakehouse Monitoring

*Retrieval Augmented Generation

Retrieval-Augmented Generation (RAG)

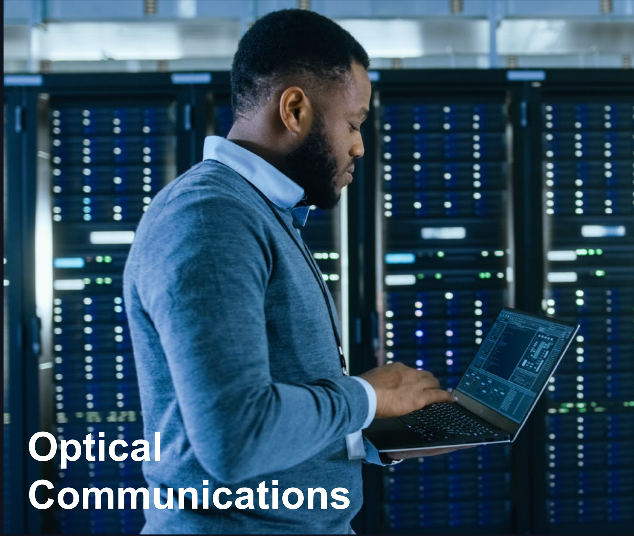


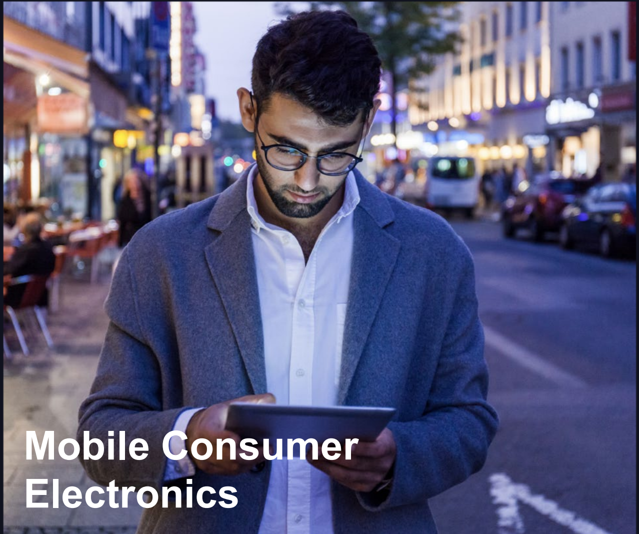
* Tools for this use case will run on the compute cluster

About Corning


Who
are we?

The industries we help shape

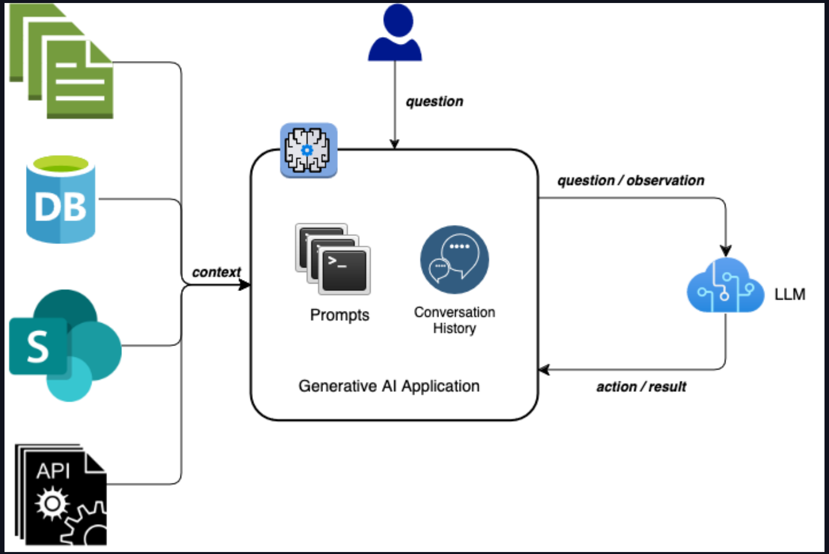




GENERATIVE AI APPLICATIONS

Isn't AI intelligent enough?

- LLMs are string-input/string-output functions
- Very powerful, but cannot perform actions
- Require additional information besides the user's question
- GenAI applications orchestrate information flow between user, LLM and Enterprise systems
- ChatGPT is a great example



COMMON USE CASES

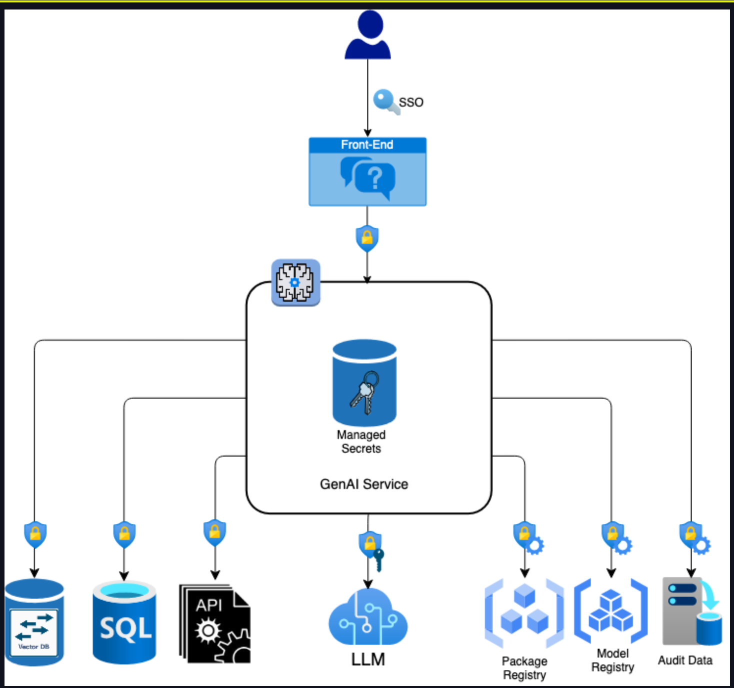
(for interactive assistants)

Retrieval-Augmented Generation (RAG)	Structured Query Building (Text2SQL)	Mixed
• Unstructured data ("talk to PDF") • May contain images/tables • Structured data ("talk to a Markdown table") • Full dataset rarely fits into the LLM's context window • Model collaboration for semantic search and summarization • Vector database for semantic search	• Structured data ("talk to database", "talk to API") • Query language or API spec • Full data set does not fit into the LLM's context window • Another system is responsible for query processing • Model collaboration for structured query building	• Unpredictable number of steps of querying and refining data • Chain of thought Agents • Different types of information retrieval are represented by Tools • Structured queries are typically declarative • Imperative code generation and sandbox execution is imaginable

ENTERPRISE NEEDS

Did you talk with your Architect?

- GenAI application is another interface to enterprise data
- UX is conversational, but NFRs are no different from other types of enterprise apps
- Data security is paramount
- SSO and access controls
- New types of restricted data: conversation history, vector databases, inference tables, prompts



OUR JOURNEY

DATA GOVERNANCE

Ensuring that corporate secrets stay secure

UNITY CATALOG

- Unified governance for all traditional types of data
- Extends to new data specific to GenAI applications: vector stores, prompts, inference tables, models, deployments, secrets
- Mlflow can act as a package manager with built-in UC governance
- Databricks External Model Serving brings even 3rd party LLMs under UC governance (test added latency)

PRIVATE LLM DEPLOYMENT

- On-Prem
- Azure OpenAI (enterprise account)
- Databricks Foundation Models (serverless)

CHAIN FRAMEWORKS

Which to choose?

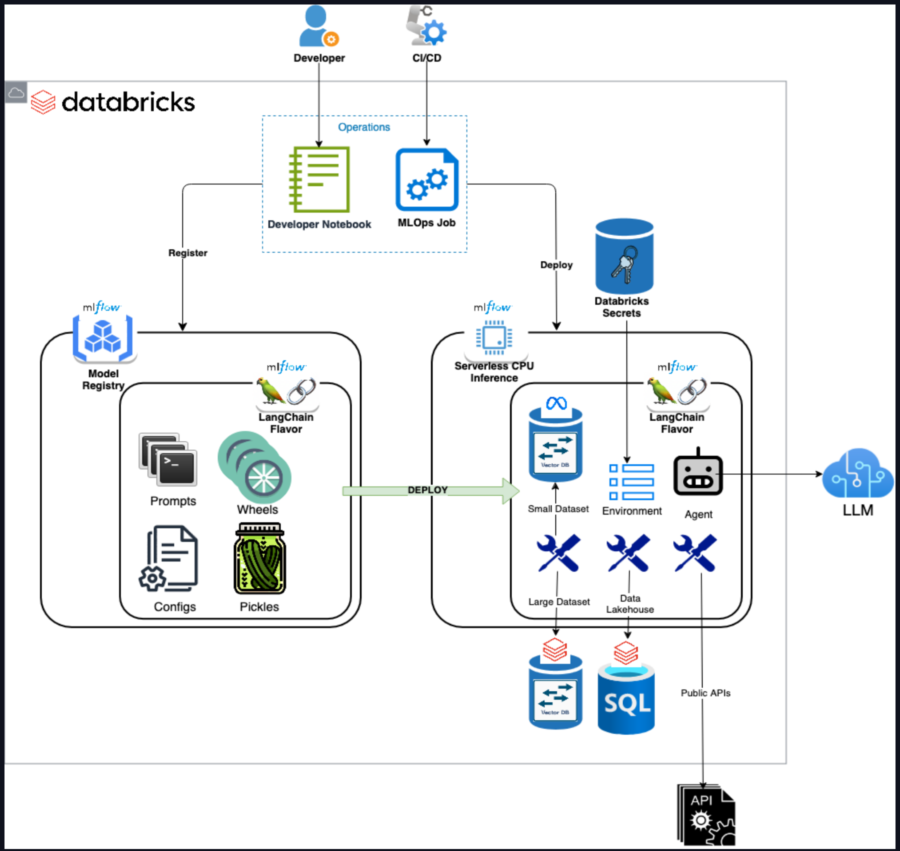
LangChain	LlamaIndex	Others

• Most popular
• License: MIT
• Supports Azure OpenAI
• Supports Databricks
• Supports Databricks Vector Store
• Mlflow Flavor
• Databricks RAG Studio (private preview)
	• Very popular
	• License: MIT
	• Supports Azure OpenAI
	• Supports Databricks
	• Supports Databricks Vector Store
	• No Mlflow Flavor
		• Haystack (deepset.ai)
		  ◦ Well-documented, small community
		  ◦ License: Apache 2.0
		• AutoGen (Microsoft++)
		  ◦ Niche, more like LangGraph
		  ◦ License: Creative Commons 4.0
		• Promptflow, MiniChain, Promptify etc
		• DSPy
		  ◦ Prompt learning

DATABRICKS-CENTRIC DESIGN

Our choices

- Use LangChain to build GenAI services
- Package in Mlflow using `mlflow.langchain` flavor
- Deploy GenAI applications as Databricks Serverless Model Serving endpoints
- Use Databricks Vector Search for large volume of unstructured data
- Use in-memory vector database for small datasets
- Use Databricks SQL Warehouse for structured queries
- Use Databricks Secrets for key management and token rotation



RAG EXAMPLES

Large and small

US PATENT SEARCH

- Hundreds of thousands of large documents
- 25 million chunks
- Ingest data from public API into a Data Lake Bronze table
- Leverage Databricks Vector Search

INTERNAL DOCUMENT SEARCH

- Hundreds to thousands of documents
- Diverse document formats
- Land files from SMB shares in UC Volumes
- Ingest document content into Data Lake Bronze tables
- Store vector database as Mlflow artifact
- Use in-memory vector search compatible with cloudpickle (FAISS)

CHAIN OF THOUGHT

Lessons learned

- Even RAG applications require switching to Agent-based implementation for more flexibility
- Example: US Patent Search uses semantic search when user asks an abstract question...
- ...but it needs to perform lookup by Patent ID if user asks specific question about a patent or to summarize a patent
- Different modes of querying Patent Database are expressed as different tools
- Databricks Vector Search supports Hybrid Search, i.e., restricting search space by a set of filters
- Requires high-end LLMs, such as GPT-4 to make fewer mistakes
- We use `langchain.agents.StructuredChatAgent` with a few modifications

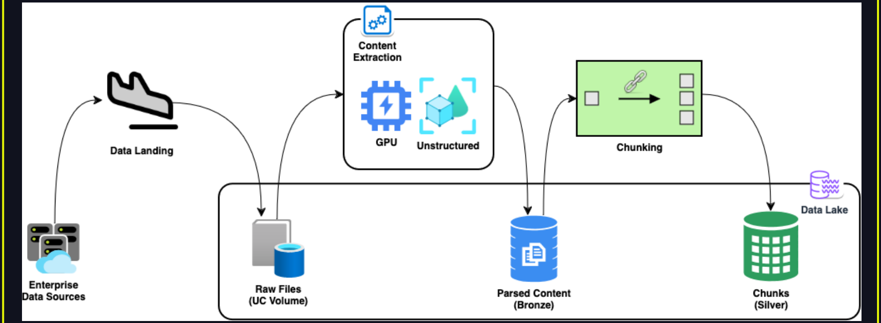
Image credit: https://python.langchain.com/v0.1/docs/use_cases/tool_use/

LESSONS LEARNED

DOCUMENT INGESTION

Lessons learned

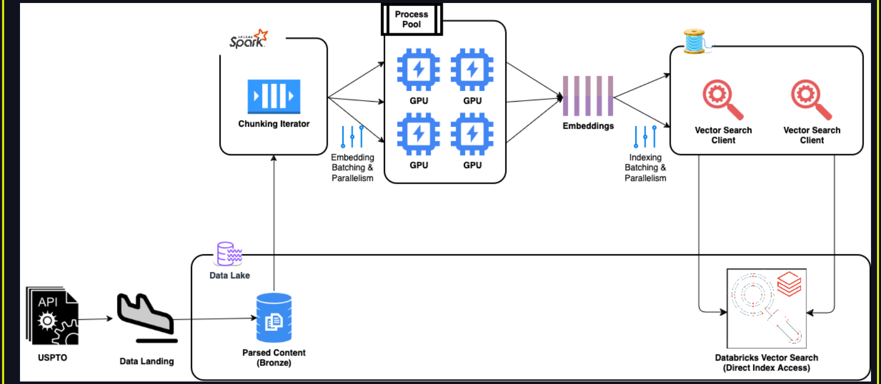
- Separate content extraction from content analysis and chunking
- Leverage medallion architecture
- Unstructured library is great for content extraction!
- Use GPU to speed up extraction from images
- Standard Databricks data engineering best practices apply
- Need to implement self-serve file landing for business users



DOCUMENT VECTORIZATION

Lessons learned

- API-based vectorization is expensive and rate-limited
- Use open-source transformer models for vectorization (we like WhereIsAI/UAE-Large-V1 “AnglE” model)
- Keep track of MTEB leaderboard
- Make sure chunks aren’t longer than embedding context!
- GPU is necessary for vectorizing large datasets
- OSS models run fast enough on CPU at query time
- Implemented custom LangChain VectorStore: DatabricksVectorStore capable of vectorizing in parallel on a multi-GPU node
- Used Direct Access index with pre-scaled endpoints
- Decouple GPU parallelism from indexing API parallelism



DOCUMENT VECTORIZATION

New from Databricks

- Serving for OSS vectorizer models
- Delta Sync vector search with precalculated embeddings
- `from langchain_community.vectorstores import DatabricksVectorSearch`
- (search only: does not use parallel indexing)

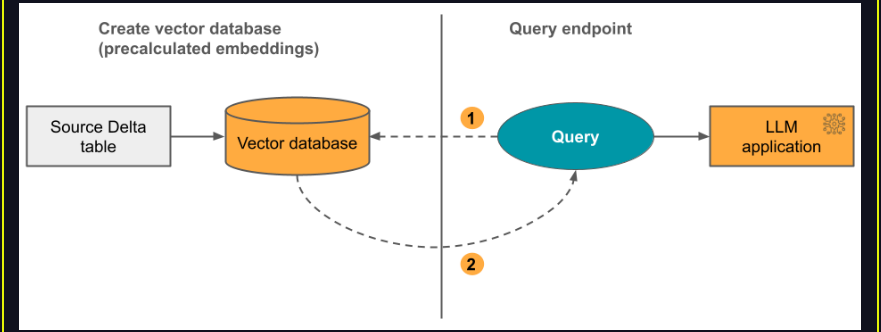
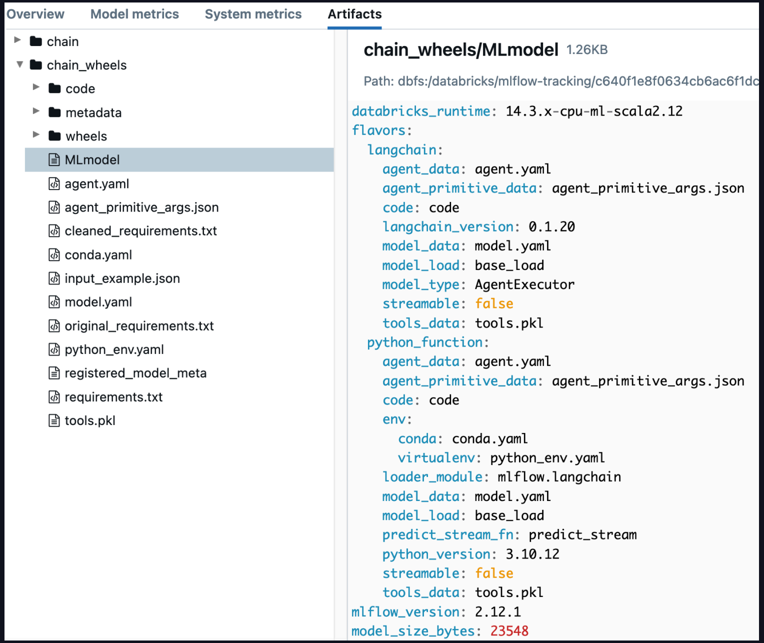


Image credit: https://docs.databricks.com/en/generative-ai/vector-search.html

APPLICATION PACKAGING

Lessons learned

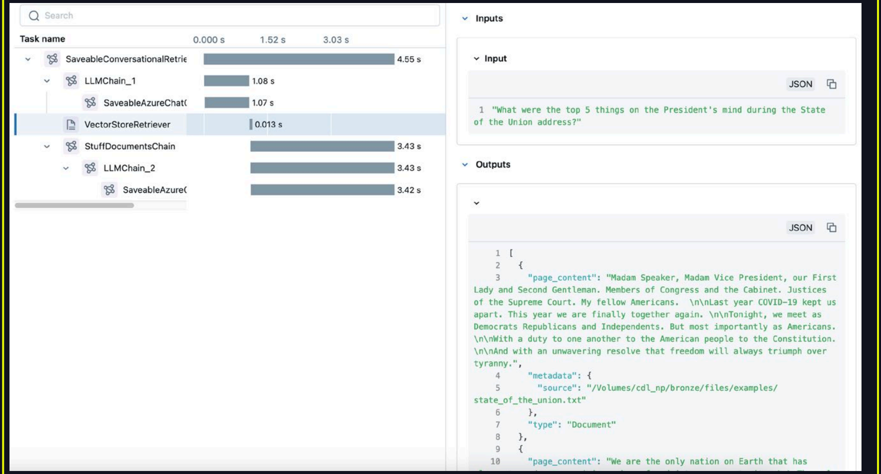
- GenAI applications are essentially microservice deployments
- Mlflow can act as a (very basic) package repository
- Python sources can be attached, but we recommend using Python wheels distributed through private package repository
- Make sure to store all dependent wheels as Mlflow Artifacts!
- Difficult to develop code when Mlflow flavor is evolving at the same time, lots of workarounds...
- ...but you get full transparency of chain internals
- Streaming support is in private preview
- Always unit-test model loading



APPLICATION DEVELOPMENT

Lessons learned

- GenAI applications are interactive by nature
- Databricks Notebooks are a great development environment for interactive testing
- LangChain verbose mode is very helpful for chain of thought debugging
- RAG Studio has Mlflow Tracing UI (like a basic version of LangSmith) in private preview
- We control the entire GenAI runtime through the dependency list of our single internal library
- That same library is passed as `extra_pip_requirements` to Mlflow
- CI/CD support via metadata embedded directly into the notebook as a Python dict



SIMPLE BOT

Step 1: set up

PYTHON

```python
# Install Corning GenAI runtime: similar to Databricks RAG Studio
%pip install --no-cache --extra-index-url https://*****gitlab.toolchain.corning.com***** -U "******"

# Metadata for submitting the notebook from CI/CD to register the application in Production
# CI/CD can use Corning tool to submit the notebook as follows:
# dbxx smart-submit --subconf prod --profile prod notebook.py
__DBXX__ = {
  'job_type': 'PROD',
  'flavor': 'SCALE',
  'custom_tags.*****': '*****',
  'emails': '*****',
  'timeout_seconds': 1800, #30 minutes
  ‘prod': {
    'databricks_environment': '*****'
  }
}
```

SIMPLE BOT

Step 2: vectorize text

PYTHON

```python
# Create serializable in-memory vector index
texts = process_documents()

embedding_model = create_embedding_model("all-MiniLM-L6-v2", use_api_management=False)

db = FAISS.from_documents(texts, embedding_model)

retriever = db.as_retriever(search_kwargs={"k": 10})
```

SIMPLE BOT

Step 3: create LangChain

PYTHON

```python
# Saveable version comes from Corning; “saveable” refers to ability to persist in MLflow
llm = create_langchain_llm(ModelType.APIM)

chain = SaveableConversationalRetrievalChain.from_llm(llm=llm, condense_question_llm=llm, retriever=retriever)

# Test bot
input_example = {
    "question": "What were top 5 things on President's mind?",
    "chat_history": """[["What was the speech about?", "The speech was the State of the Union address."]]"""
}

chain.invoke(input_example)
```

SIMPLE BOT

Step 4: register in MLflow

PYTHON

```python
# Loader function will load vector store from Mlflow artifacts

def loader(persist_dir):
  from langchain_community.vectorstores import FAISS
  db = FAISS.load_local(
    persist_dir,
    embeddings=create_embedding_model("all-MiniLM-L6-v2")
  )
  return db.as_retriever(search_kwargs={"k": 10})
```

PYTHON

```python
# Register in Mlflow using Corning enhancements
# to the mlflow.langchain flavor

with TemporaryDirectory() as persist_directory:
  db.save_local(persist_directory)
  model_info = log_langchain(chain=chain,
                             name=REGISTER_LANGCHAIN_AS,
                             input_example=input_example,
                             persist_directory=persist_directory,
                             loader_fn=loader,
                             gpu_serving=False,
                             uses_api_management=True,
                             mlflow_params={
                                "llm_adapter": type(llm).__name__
                              })
```

SIMPLE BOT

Step 5: test and deploy

PYTHON	PYTHON

`# Locally test model loading from MLflow`
 	
`loaded_model=mlflow.pyfunc.load_model(f"models:/{REGISTER_LANGCHAIN_AS}/latest")`
 	
`loaded_model.predict(input_example)`
	`# Create Serving endpoint`
	
	`print("Deploying to Databricks Serverless Model Serving...")`
	
	`deployed = deploy_endpoint(REGISTER_LANGCHAIN_AS, uses_api_management=True)`
	
	`print(deployed)`

FUTURE DIRECTION

CHALLENGES

What is hard?

DEVELOPMENT

- mlflow.langchain flavor is evolving in parallel with LangChain itself: not easy to predict what is currently supported and what is not
- LLM latency and Agent latency
- Streaming feedback in chain of thought
- Vectorizing very large datasets at reasonable throughput and cost
- Using frameworks other than LangChain (e.g. LlamaIndex Packs)

PRODUCTION

- Data Engineering
- Token rotation and management
- UC permissions management
- Streaming support in Model Serving
- GenAI application evaluation and human feedback collection
- Scaling of the Databricks Vector Search infrastructure
- MLOps environment flow

NEXT STEPS

What are we working on?

DEVELOPMENT

- Adopt latest RAG Studio (private preview) changes: keep deleting code!
- Adopt modern LangChain coding style: LCEL and LangGraph
- Use simpler open-source LLMs for simpler tasks: reduce latency
- Migrate from Direct Access Index to Delta Sync with precomputed embeddings
- Make it easier for non-developers to contribute prompts (Mlflow Prompt Engineering UI?)
- Multi-modal applications!

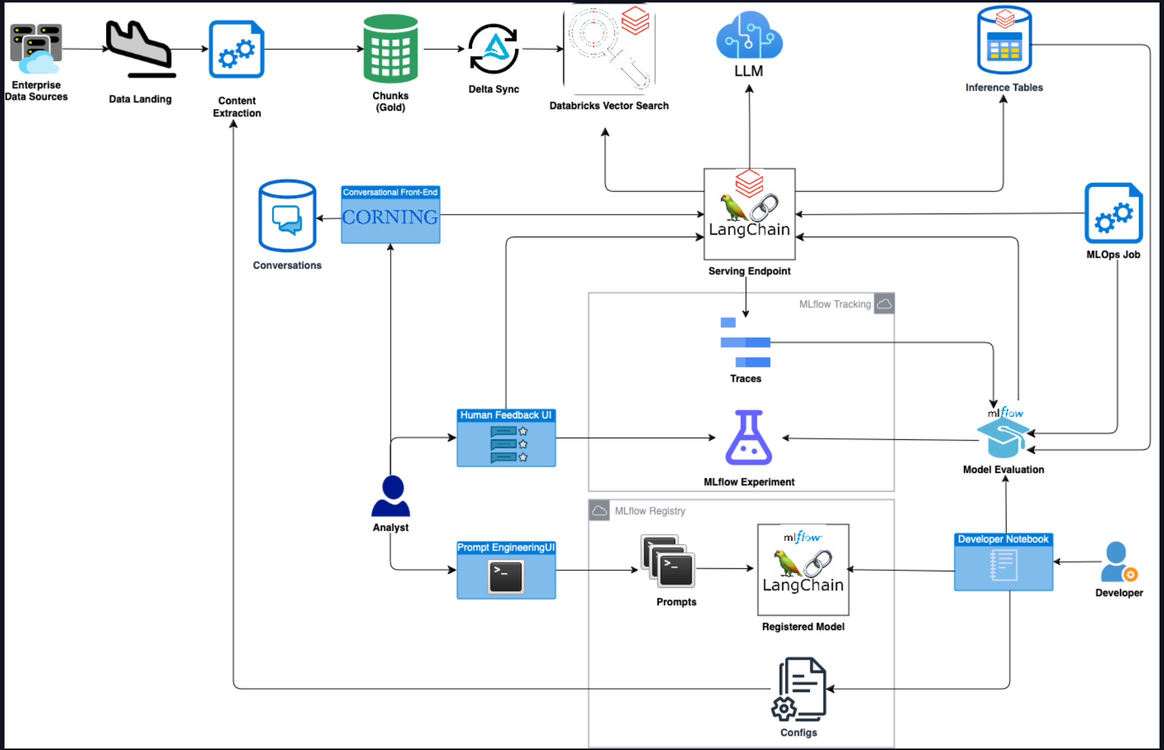
PRODUCTION

- Improve streaming feedback to the user
- Incorporate RAG Studio model evaluation into CI/CD
- Build flexible document ingestion pipelines
- UC permission management automation
- Automatic discovery of new Model Serving endpoints in the UI Portal
- Expand the use of Inference Tables for tracking model performance

OUR VISION

Closed-loop architecture

- Automated data ingestion
- Automated model deployment
- Human in the loop feedback harvesting
- Expert-crafted prompts



[Mosaic AI Agent Framework]

BACKGROUND

[Mosaic AI RAG Framework]

Where are we?

Production -quality GenAI is difficult.

Organizations are struggling to put GenAI application into production

What is the pain?

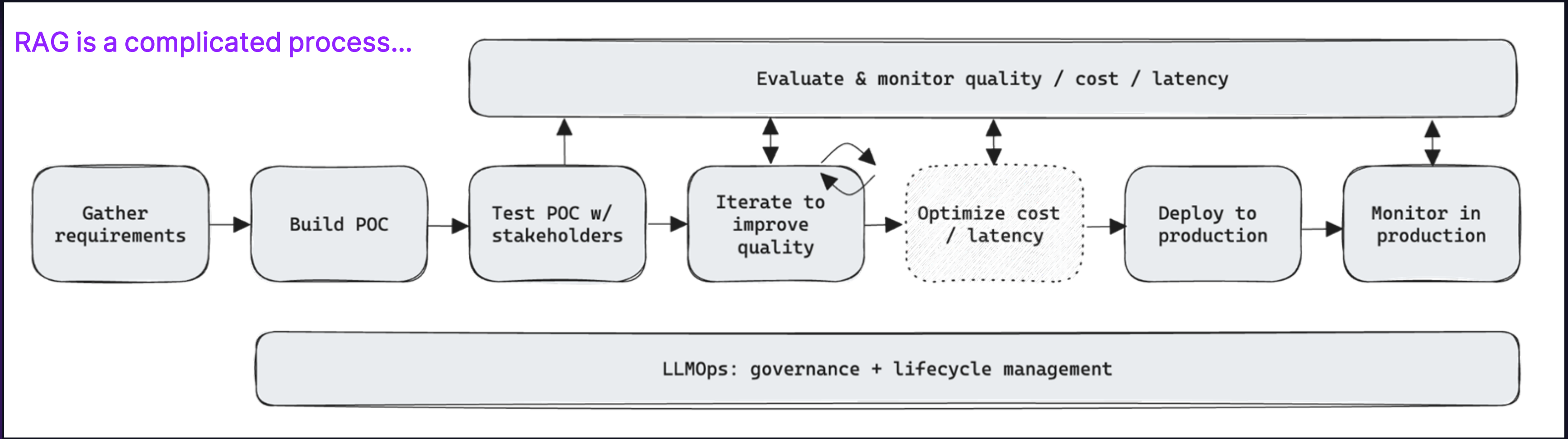
Don’t know when the app is producing responses that are not accurate, safe, or governed(no evaluation tools), and how to fix it (dev tools).

What is the gain?

[Mosaic AI RAG Framework] makes it easy to evaluate the quality of the app, iterate quickly and test hypothesis, and redeploy the application easily

HOW DOES IT WORK?

[Mosaic AI RAG Framework]



[Mosaic AI RAG Framework] helps deploy with a workflow and evaluation tools

Upgrades to Mosaic AI to help deploy RAG easier

Quality Lab with AI-assisted judges and human review UI

BENEFITS

[Mosaic AI RAG Framework] helps deploy production -quality Generative AI applications

1

Understand Quality

[Mosaic AI RAG Framework] has built-in proprietary AI-assisted evaluation that can automatically determine if outputs are high quality as well as an intuitive UI to get feedback from human stakeholders.

2

Rapid Development

[Mosaic AI RAG Framework] makes it easy for developers to take feedback, and rapidly iterate on changes to test every hypothesis. They can then redeploy their application into production with no code changes using an end-to-end LLMOps workflow. Developers can iterate on all aspects of the RAG process.

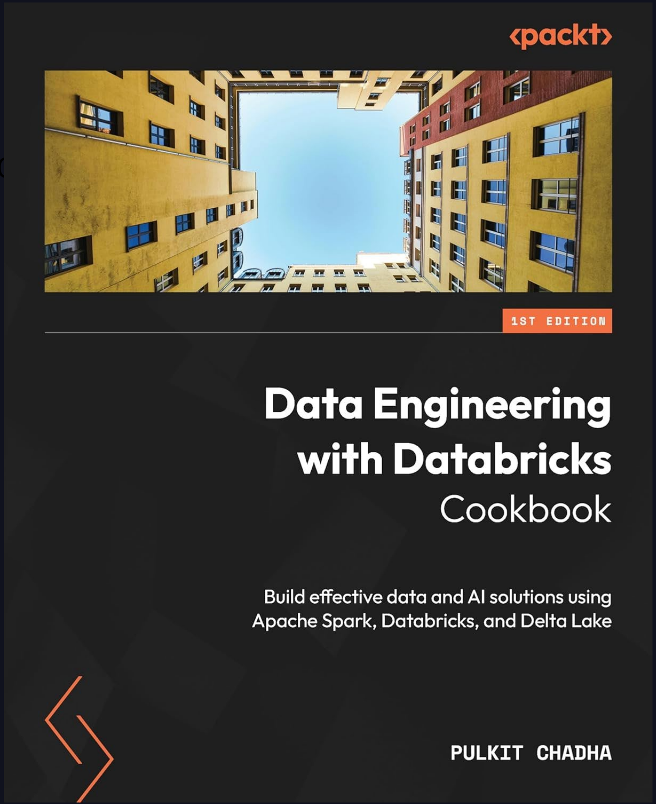
3

Governance

[Mosaic AI RAG Framework] is seamlessly integrated with the rest of the Databricks Data Intelligence Platform. This means you have everything you need to deploy an end-to-end RAG system from security and governance, to data integration, vector databases, quality evaluation, and one-click optimized deployment.

Giveaway

25DED



25DEDC

25% Discount Code

(Valid June 10th - 25th)

databricks