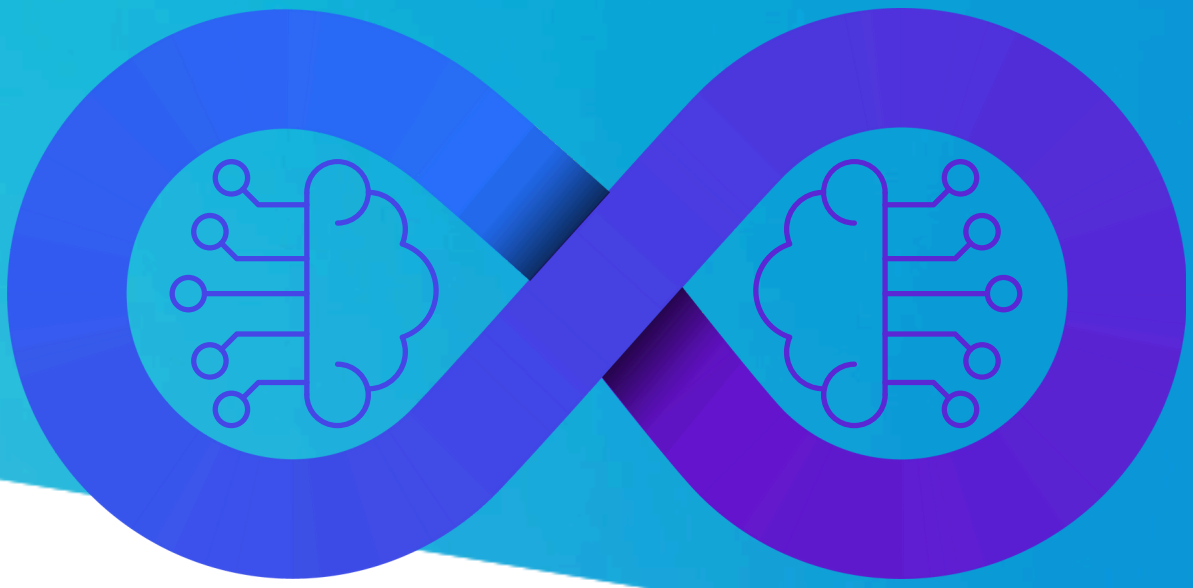
SKILLFYME
Empowering your success

MLOps with Agentic AI

Advanced Certification Program

Curriculum



www.skillfyme.com

Index

Module 1: Python and MLOps Foundations

- Python + Numpy + Pandas
- MLOps overview, CI/CD for ML, MLflow, model versioning

Module 2: Modern Cloud-Native MLOps

- Airflow, Kubeflow, Optuna
- Kubernetes, TF Serving, Prometheus, Grafana
- Advanced deployment (blue-green, canary)
- Feedback loops, retraining, and cost optimization

Module 3: Cloud & Productionization

- AWS pipelines with SageMaker/ML Studio
- Model registry, approval workflows, drift monitoring
- Multi-cloud pipeline integration
- Capstone 1: End-to-end MLOps pipeline

Module 4: Agentic AI & LLMOps

- LLM lifecycle, Fine-tuning vs Prompt-tuning
- RAG architecture, vector DBs
- Deploying LLMs (Kubernetes, LangServe)
- LangChain, CrewAI, LangGraph
- Evaluation tools: RAGAS, LangSmith, Trulens
- Capstone 2: Build & deploy an agentic AI system for enterprise use-case

1 Python and MLOps Foundations

Session 1: Python Refresher for MLOps (Beginner)

Goal: Develop modular Python code structures essential for machine learning engineering, emphasizing reusability, logging, configuration, and maintainability.

Topics Covered:

- **Core Python Functions:**
  - def syntax, return statements, lambda expressions
  - *args and **kwargs for flexible APIs
  - Use-cases in ML pipelines for utility or metrics functions

- **OOP Concepts in ML:**
  - Class creation with __init__
  - Instance vs class variables (e.g., for shared configs)
  - Inheritance and overriding methods (e.g., base Trainer with CNN/RandomForest variants)
  - Reusable design for ML code using class-based components (ModelTrainer, Logger, ConfigManager)

- **Decorators in MLOps:**
  - Logging training start-end time
  - Custom decorators for timing and exception handling
  - How to wrap functionality across pipelines

- **Modular Code Organization**:
  - src/, utils/, configs/, models/, logs/
  - Structuring reusable ML components for collaboration
  - Using __init__.py, and best practices for imports

- **File Handling**:
  - Create directories using os.makedirs() and pathlib.Path()
  - Check file existence, write logs/artifacts/model weights programmatically

1 Python and MLOps Foundations

- **Logging for Observability:**
  - Set up logger with formatters and handlers
  - Log at multiple levels: INFO, DEBUG, ERROR
  - Different logs for train.log, error.log, experiment.log

- **Command-Line Interfaces:**
  - Use argparse to configure ML scripts dynamically (e.g., --lr, --epochs, --model_path)
  - Simulate real-world ML pipelines with parameter-driven runs

1 Python and MLOps Foundations

Session 2: NumPy & Pandas Essentials (Beginner)

Goal: Build expertise in handling and transforming numerical/tabular data using NumPy and Pandas, forming the foundation of data pipelines in MLOps.

Topics Covered:

- **Core Python Functions:**
  - def syntax, return statements, lambda expressions
  - *args and **kwargs for flexible APIs
  - Use-cases in ML pipelines for utility or metrics functions

- **NumPy for Matrix Ops & Preprocessing**:
  - Create structured arrays for input pipelines
  - Slice, reshape, and broadcast input tensors (features)
  - Perform linear algebra: dot products, inverses, SVD
  - Use-case: prepare data for ML model input

- **Pandas for Tabular ML Data**:
  - Load, explore, and clean CSV/Excel data
  - Filter/select with loc, iloc, conditionals
  - Treat missing data using .fillna(), .dropna()
  - Encode categorical variables (get_dummies(), LabelEncoder)
  - Perform aggregations and grouped operations for feature engineering

- **Memory & Pipeline Best Practices**:
  - Efficient DataFrame chaining
  - Avoid unnecessary .copy() calls and in-place mutations
  - Use type conversion to reduce memory usage

1 Python and MLOps Foundations

Session 3: MLOps Overview & ML System Design (Beginner)

Goal: Grasp the MLOps lifecycle from development to production, understanding architecture design, tool selection, and team roles in real-world AI systems.

Topics Covered:

- **MLOps Fundamentals**:
  - What is MLOps? Where does it differ from DevOps?
  - Common problems solved by MLOps: reproducibility, automation, monitoring
- **ML System Lifecycle**:
  - Data collection → Cleaning → Experimentation → Training → Evaluation → Deployment → Monitoring → Retraining
- **Tooling Landscape**:
  - Git, MLflow, Docker, Kubernetes, Airflow, Prometheus, Grafana
  - Importance of tracking: data versioning, code changes, experiment results
- **Architectural Styles**:
  - Monolithic training scripts vs modular pipelines
  - Batch inference vs real-time model serving
  - Design patterns: training/inference segregation, microservices
- **Roles in an MLOps Team**:
  - ML Engineer, MLOps Engineer, Data Scientist, Platform Engineer

1 Python and MLOps Foundations

Session 4: Git & CI/CD for ML (Intermediate)

Goal: Learn the principles of version control and build automation pipelines tailored to ML workflows using GitHub Actions.

Topics Covered:

- **Git Fundamentals for ML**:
  - Track notebooks, models, and datasets using .gitignore, .gitattributes
  - Create branches for model experimentation
  - Use tags for tracking production model versions
- **GitHub Collaboration**:
  - Pull requests and code reviews for ML code
  - Manage issues, enhancements, and release cycles
- **Git LFS**:
  - Track large files (datasets, model weights) efficiently
  - Configure .gitattributes and remote storage for ML artifacts
- **CI/CD Concepts for ML**:
  - Continuous Integration: test code, verify environments, linting
  - Continuous Delivery: deploy models, update inference endpoints
- **GitHub Actions Deep Dive**:
  - Define workflows to install Python env, run tests, lint code
  - Add stages: build, train, test, validate

1 Python and MLOps Foundations

Session 5: Model Tracking with MLflow (Intermediate)

Goal: Enable reproducibility and experiment comparison by using MLflow to log, organize, and visualize multiple ML model training runs.

Topics Covered:

- **Why Model Tracking Matters**:
  - Track different experiment runs and hyperparameters
  - Ensure reproducibility and enable collaboration
  - Compare performance metrics across runs

- **Introduction to MLflow**:
  - Components overview:
    - **Tracking**: Logging metrics, params, artifacts
    - **Projects**: Packaging ML code for reproducibility
    - **Models**: Model abstraction and deployment formats
    - **Registry**: Lifecycle and stage control of models

- **MLflow Tracking API**:
  - Use mlflow.start_run() to begin experiment tracking
  - Log:
    - Parameters (mlflow.log_param)
    - Metrics (mlflow.log_metric)
    - Artifacts (mlflow.log_artifact)
    - Models (mlflow.sklearn.log_model)
  - Use autolog() with popular libraries like sklearn, keras

- **MLflow UI**:
  - Launch and navigate the MLflow dashboard
  - Filter, compare, and visualize multiple experiment runs
  - Add tags and notes for better organization

- **Use-Cases**:
  - Comparing hyperparameters (e.g., learning rate, max_depth)
  - Logging dataset version used in training
  - Storing visualizations (e.g., ROC curve, confusion matrix) as artifacts

1 Python and MLOps Foundations

Session 6: MLflow Projects & Model Registry (Intermediate)

Goal: Standardize and operationalize ML training workflows with MLflow Projects and manage production-ready models with MLflow Model Registry.

Topics Covered:

- **MLflow Projects**:
  - Structure ML code with a MLproject file
  - Define entry_points, conda environments, and default parameters
  - Benefits for reproducibility, portability, and automation
  - Run project locally or remotely (e.g., MLflow server)
- **MLflow Model Registry**:
  - **Register Models** via code or UI
  - **Version Control**: Automatically increment versions
  - **Stage Transitions**:
    - None → Staging → Production → Archived
  - **Approval Workflows**:
    - Manually or programmatically promote/demote models
    - Attach metadata and comments (e.g., accuracy, reviewer)
- **Rollback and Reuse**:
  - Retrieve specific model versions
  - Revert to previous production-ready model

1 Python and MLOps Foundations

Session 7: Model Packaging and Versioning (Intermediate)

Goal: Learn how to serialize, version, and manage models for distribution, deployment, and rollback across environments.

Topics Covered:

- **Serialization Techniques**:
  - Use pickle, joblib for Python-based models
  - Use ONNX, TorchScript for framework-agnostic, portable deployment
  - Considerations: file size, compatibility, security

- **Versioning Best Practices**:
  - Follow Semantic Versioning (SemVer): MAJOR.MINOR.PATCH
  - Git tags for syncing code versions with model versions
  - Save metadata (e.g., training timestamp, dataset hash)

- **Reproducibility**:
  - Save environment dependencies: requirements.txt, conda.yaml
  - Log model signature (input/output schema)
  - Capture random seeds and random states for full reproducibility

1 Python and MLOps Foundations

Session 8: End-to-End CI/CD for ML with DVC (Advanced)

Goal: Build a reproducible and automated ML pipeline using DVC for dataset and model versioning, integrated with GitHub Actions for CI/CD.

Topics Covered:

- **Serialization Techniques**:
  - Use pickle, joblib for Python-based models
  - Use ONNX, TorchScript for framework-agnostic, portable deployment
  - Considerations: file size, compatibility, security

- **Versioning Best Practices**:
  - Follow Semantic Versioning (SemVer): MAJOR.MINOR.PATCH
  - Git tags for syncing code versions with model versions
  - Save metadata (e.g., training timestamp, dataset hash)

- **Reproducibility**:
  - Save environment dependencies: requirements.txt, conda.yaml
  - Log model signature (input/output schema)
  - Capture random seeds and random states for full reproducibility

2 Modern Cloud-Native MLOps

Session 9: Orchestration with Apache Airflow (Advanced)

Goal: Build scalable, repeatable ML pipelines using Airflow's DAG-based orchestration.

Topics Covered:

- **Why Workflow Orchestration in MLOps**:
  - Automate and schedule end-to-end ML workflows
  - Enable retry logic, failure notifications, and inter-task dependencies
  - Maintain pipeline reproducibility and auditability

- **Airflow Concepts**:
  - **DAG (Directed Acyclic Graph)**: Defines a pipeline of tasks with dependencies
  - **Operators**: Define task logic (e.g., PythonOperator, BashOperator)
  - **Scheduler & Workers**: Schedule and execute tasks asynchronously
  - **Metadata DB**: Tracks DAG state and task history

- **Modular Pipeline Design**:
  - Break workflow into logical steps: data ingestion → preprocessing → training → evaluation → deployment
  - Use templated variables/macros for reusability

- **Airflow UI**:
  - Visualize DAG structure and execution flow
  - Monitor task retries, failures, and logs
  - Trigger DAGs manually or via schedules (cron)

2 Modern Cloud-Native MLOps

Session 10: Hyperparameter Tuning with Optuna (Intermediate)

Goal: Improve model accuracy and generalization by automatically finding optimal hyperparameters.

Topics Covered:

- **Importance of Hyperparameter Tuning**:
  - Manual tuning is inefficient and non-reproducible
  - Automated tuning ensures systematic exploration of parameter space

- **Optuna Architecture**:
  - **Study**: A full optimization session
  - **Trial**: A single evaluation run with a hyperparameter set
  - **Sampler**: Chooses next trial's parameters (e.g., TPE, Random)
  - **Pruner**: Stops unpromising trials early to save compute

- **Integration**:
  - Plug into Scikit-learn, PyTorch, XGBoost, LightGBM
  - Works with train_test_split, cross-validation, early stopping

- **Visualization**:
  - Use Optuna's dashboard or Matplotlib plots for:
    - Optimization history
    - Parameter importance
    - Slice plots

- **MLflow Logging**:
  - Combine Optuna with MLflow to persist trial metrics and artifacts

2 Modern Cloud-Native MLOps

Session 11: Kubeflow Pipelines (KFP) (Advanced)

Goal: Create scalable, portable ML pipelines using the Kubeflow SDK running on Kubernetes.

Topics Covered:

- **What is Kubeflow?**:
  - End-to-end ML toolkit built on Kubernetes
  - Tailored for repeatable training, tuning, deployment

- **KFP vs Airflow**:
  - Airflow is general-purpose; KFP is ML-specific
  - KFP integrates natively with containerized ML workflows
  - Use KFP when running on cloud-native infra with Kubernetes

- **Pipeline Components**:
  - Each step is a Docker container
  - Input/output artifacts are tracked automatically
  - Use KFP DSL to define Pythonic workflows

- **Pipeline Compilation & Deployment**:
  - Use kfp.Client() to compile and upload pipelines
  - Trigger experiments, monitor via UI, log metrics

2 Modern Cloud-Native MLOps

Session 12: Kubernetes for MLOps (Advanced)

Goal: Understand how Kubernetes manages ML infrastructure and deployment lifecycle.

Topics Covered:

- **Why Kubernetes?**:
  - Automates scaling, resilience, and resource utilization
  - Consistent environment for dev → staging → prod

- **Kubernetes Core Concepts**:
  - **Pods**: Smallest deployable unit, encapsulates containers
  - **Deployments**: Manages Pod replication and rolling updates
  - **Services**: Exposes Pods to internal/external network
  - **ConfigMaps & Secrets**: Manage configuration securely
  - **Namespaces**: Isolate environments (e.g., staging vs prod)

- **Kubernetes for ML**:
  - Deploying models as RESTful microservices
  - Define resource limits for compute-bound workloads
  - Use Horizontal Pod Autoscaler (HPA)

2 Modern Cloud-Native MLOps

Session 13: Model Deployment with Kubernetes & TF Serving (Advanced)

Goal: Serve ML models efficiently using Kubernetes-native serving tools.

Topics Covered:

- **Serving Strategies**:
  - Batch vs real-time inference
  - REST vs gRPC API protocols
- **TensorFlow Serving**:
  - Export SavedModel format
  - Serve multiple model versions with auto-routing
  - Supports batching, latency control
- **Kubernetes + Model Serving**:
  - Use InferenceService from KServe or custom Deployment
  - Deploy TF Serving or custom container
  - Use YAML to define autoscaling, resource quotas
- **Custom Python Predictors**:
  - Write predict() functions with preprocessing/postprocessing
  - Bundle model and logic in container

2 Modern Cloud-Native MLOps

Session 14: Monitoring with Prometheus & Grafana (Intermediate)

Goal: Monitor models and infrastructure for performance, reliability, and drift detection.

Topics Covered:

- **What to Monitor in MLOps**:
  - Model: latency, throughput, error rate, drift
  - Infra: CPU/GPU utilization, memory, disk, network

- **Prometheus Overview**:
  - Pulls metrics from /metrics endpoint
  - Stores time-series data for querying and alerting

- **Grafana**:
  - Visualize metrics with customizable dashboards
  - Integrate alerts (Slack, email, PagerDuty)

- **Integration with ML APIs**:
  - Use Prometheus Python client to expose metrics
  - Monitor business-level KPIs: avg response time, failure count

2 Modern Cloud-Native MLOps

Session 15: Advanced Deployment Patterns (Intermediate)

Goal: Release models safely in production using advanced deployment strategies and observability.

Topics Covered:

- **Deployment Strategies**:
  - **Blue-Green**: Deploy two environments, switch traffic
  - **Canary**: Gradual rollout to subset of users (e.g., 20% first)
  - **Shadow Testing**: Send traffic to new model but don't expose results
- **Traffic Control in Kubernetes**:
  - Use Ingress controllers or KServe traffic splits
  - Automate traffic shifting based on time or metrics
- **Rollbacks**:
  - Track model version and metrics
  - Revert automatically if thresholds are breached

2 Modern Cloud-Native MLOps

Session 16: Feedback Loops & Cost Optimization (Intermediate)

Goal: Build self-learning pipelines by incorporating feedback, and manage cloud cost efficiently.

Topics Covered:

- **Human-in-the-Loop Feedback**:
  - Log predictions + user feedback (e.g., "was this correct?")
  - Store in a feature store or feedback table

- **Drift & Retraining Triggers**:
  - Data drift: change in input distributions
  - Concept drift: change in label relationship
  - Use Kolmogorov-Smirnov tests, population stability index

- **Cost Optimization Techniques**:
  - Use spot/preemptible instances for training
  - Schedule batch inference instead of real-time
  - Apply model quantization to reduce resource usage

3 Cloud with Productionization

Session 17: AWS ML Stack Overview (Intermediate)

Goal: Get hands-on with AWS-native ML tools to build automated, event-driven pipelines from data ingestion to model inference.

Topics Covered:

- **AWS ML Ecosystem Overview**:
  - **Amazon SageMaker**: End-to-end ML lifecycle platform (train, deploy, monitor, pipeline)
  - **AWS Lambda**: Serverless compute triggered by events (e.g., new S3 file)
  - **Amazon S3**: Object storage for datasets and model artifacts
  - **Amazon ECR**: Container registry for custom Docker images
  - **CloudWatch**: Logging and monitoring for serverless and ML workloads

- **Event-Driven ML Pipelines**:
  - Real-world scenario: A new file in S3 triggers a Lambda function
  - Lambda parses file → launches SageMaker job → saves result → logs to CloudWatch
  - No need for managing servers or cron jobs

3 Cloud with Productionization

Session 18: SageMaker Pipelines with Model Registry (Advanced)

Goal: Automate and manage the entire ML lifecycle with SageMaker Pipelines and register models using built-in model registry.

Topics Covered:

- **SageMaker Pipeline Components**:
  - **ProcessingStep**: For data cleaning, feature engineering
  - **TrainingStep**: Launch ML training jobs
  - **ModelStep**: Create model from training artifacts
  - **ConditionStep**: Conditional logic based on metric thresholds
  - **RegisterModel**: Store trained model with metadata

- **Pipeline Building with Python SDK**:
  - Define parameters (e.g., learning_rate)
  - Chain steps and define dependencies
  - Visualize pipeline execution in SageMaker Studio

- **Model Registry**:
  - Register trained model to track versions
  - Transition models from "None" → "Staging" → "Production"
  - Store metadata like metrics, training job ID, deployment URL

3 Cloud with Productionization

Session 19: Drift Monitoring in Production (Advanced)

Goal: Detect when your production model starts to degrade due to changes in data or concept.

Topics Covered:

- **Types of Drift**:
  - **Data Drift**: Input distribution has changed (e.g., new age group in user data)
  - **Concept Drift**: The relationship between features and labels has changed (e.g., sentiment meaning changes over time)
  - **Label Drift**: Class distribution changes in output (e.g., more fraudulent transactions)
- **Drift Detection Techniques**:
  - **KS Test**: Statistical test for comparing distributions
  - **Population Stability Index (PSI)**: Measures how stable a variable's distribution is over time
  - **Model Score Drift**: Distribution of predicted scores changing
- **SageMaker Model Monitor**:
  - Automatically collects endpoint invocations
  - Compares them against a **baseline** dataset
  - Can log alerts, metrics, and trigger retraining pipelines

3 Cloud with Productionization

Session 20: Multi-Cloud MLOps Integration (Advanced)

Goal: Understand how to design cloud-agnostic ML pipelines that combine tools across AWS, GCP, and Azure while maintaining governance and scalability.

Topics Covered:

- **Challenges in Multi-Cloud MLOps**:
  - Data silos and inconsistent APIs
  - Governance and security across platforms
  - Managing costs and compliance in regulated industries

- **Service Comparison**:
  - **SageMaker (AWS)** vs **Vertex AI (GCP)** vs **Azure ML**
    - Differences in training orchestration, model registry, pipelines, SDKs
    - When to prefer one over another

- **Cloud-Agnostic Pipelines**:
  - Use open tools like **MLflow**, **KServe**, **Kubernetes**, **DVC**
  - Abstract storage with S3, GCS, or MinIO
  - Use Terraform to define infra across clouds

- **Hybrid Strategy Example**:
  - Train model on GCP Vertex AI
  - Register model in MLflow
  - Deploy on AWS using SageMaker or EKS

3 Cloud with Productionization

Sessions 21 & 22: Capstone Project 1 – End-to-End MLOps Pipeline (Advanced)

Goal: Build a full production-grade MLOps pipeline integrating tools from previous sessions, simulating a real business problem.

Topics Covered:

- **Use-Case Definition**:
  - Choose a real-world use case (e.g., loan default prediction, email classification, review sentiment, product recommendations)
  - Decide problem type: regression, classification, NLP
- **Architecture Blueprint**:
  - **Data ingestion**: From CSV, API, or database
  - **Preprocessing**: Transform, clean, validate data
  - **Model training**: Train with versioned code and data
  - **Experiment tracking**: MLflow logging of metrics/artifacts
  - **CI/CD pipeline**: Automate testing, packaging, and deployment
  - **Model Registry**: Store versions and manage promotion
  - **Deployment**: Kubernetes + InferenceService (or SageMaker endpoint)

Monitoring: Track latency, errors, and drift with Prometheus/Grafana or Model Monitor

4 Agentic AI & LLMOps

Session 23: LLM Lifecycle + Prompt Engineering (Intermediate)

Goal: Learn how to design effective prompts and understand the lifecycle of LLM applications.

Topics Covered:

- **LLM Lifecycle Overview**:
  - **Pretraining**: Models trained on large corpora (e.g., GPT, LLaMA)
  - **Fine-tuning**: Specialized domain training
  - **Inference**: Deployment in real-world apps

- **Prompt Engineering**:
  - **Zero-shot**: Ask directly without examples
  - **Few-shot**: Provide 1–3 examples to guide generation
  - **Chain-of-thought (CoT)**: Guide reasoning through intermediate steps

- **Prompt Templates**:
  - Use Jinja-style format in LangChain for reusable prompts
  - Introduce **guardrails** to filter or sanitize output

- **LangChain Prompt Utilities**:
  - PromptTemplate, ChatPromptTemplate
  - Manage dynamic inputs, temperature, roles

4 Agentic AI & LLMOps

Session 24: Fine-tuning vs Prompt-tuning (Advanced)

Goal: Compare cost-effective LLM adaptation strategies, and fine-tune a small model using LoRA or QLoRA.

Topics Covered:

- **Adaptation Strategies**:
  - **Full Fine-Tuning**: Update all weights — high cost, large gains
  - **PEFT (Parameter-Efficient Fine-Tuning)**: Modify adapters (LoRA), saves memory

- **Trade-Offs**:
  - Inference latency, storage size, data sensitivity
  - Quality vs cost for large vs small models

- **Tools & Frameworks**:
  - Hugging Face transformers, peft, accelerate
  - Google Colab with T4/A100 GPUs

4 Agentic AI & LLMOps

Session 25: RAG Architecture with Vector DBs (Advanced)

Goal: Build scalable RAG (Retrieval-Augmented Generation) pipelines with embedding-based search.

Topics Covered:

- **What is RAG?**
  - Combine a retrieval step with an LLM generation step
  - Reduce hallucinations, ground responses in documents

- **Chunking & Embedding**:
  - Chunk long docs into semantically meaningful blocks
  - Convert to vectors using OpenAI, Cohere, or HuggingFace embeddings

- **Vector Databases**:
  - **FAISS**: Local, fast, easy to use
  - **Pinecone**, **Weaviate**: Scalable, managed
  - Indexing strategies and search accuracy

- **LangChain Integration**:
  - RetrievalQA, VectorStoreRetriever, ConversationalRetrievalChain

4 Agentic AI & LLMOps

Session 26: LLM Caching & Routing Techniques (Advanced)

Goal: Optimize system performance and cost by using caching and routing between models.

Topics Covered:

- **Caching Techniques**:
  - Cache common prompts + responses to speed up
  - Store embeddings or final completions (e.g., Redis, file, memory)

- **Routing**:
  - Use fallback models: expensive → cheaper
  - Context-aware routing based on user, input length, or budget

- **LangChain Tools**:
  - MultiPromptChain, LLMRouter, Cache, RetrieverRouter

4 Agentic AI & LLMOps

Session 27: Agentic Frameworks – LangChain, CrewAI, LangGraph (Advanced)

Goal: Build systems with autonomous, role-based agents that collaborate to complete tasks.

Topics Covered:

- **Agentic AI Explained**:
  - Agents have tools, memory, and goals
  - Act autonomously with decision-making ability
- **LangChain Agents**:
  - Tool usage, memory integration (e.g., search, calculator)
- **CrewAI Framework**:
  - Define **Roles** (e.g., researcher, writer)
  - Build **Crew** of agents for collaborative tasks
- **LangGraph**:
  - Use graph-based workflows for deterministic execution

4 Agentic AI & LLMOps

Session 28: Evaluation Tools – RAGAS, LangSmith, TruLens (Advanced)

Goal: Learn how to evaluate LLM app performance using open and proprietary evaluation frameworks.

Topics Covered:

- **Why LLM Evaluation is Hard**:
  - No one-size-fits-all metric
  - Output is variable and subjective

- **Metrics**:
  - **Faithfulness**: grounded in source
  - **Relevance**: is it useful/related?
  - **Conciseness**, **Toxicity**, **Hallucination rate**

- **Tools**:
  - **RAGAS**: Evaluate RAG pipelines
  - **LangSmith**: Track execution traces, latency, tool calls
  - **TruLens**: Quantitative feedback + human scoring

4 Agentic AI & LLMOps

Session 29: Evaluating LLMs – Metrics, Hallucination, Quality (Advanced)

Goal: Build LLM evaluation pipelines and integrate them with CI/CD for regression testing.

Topics Covered:

- **Types of Evaluation**:
  - **Static**: Against ground truth labels
  - **Dynamic**: Involving humans-in-the-loop
  - **Hybrid**: Combine human feedback + auto-metrics

- **Tools**:
  - **PromptLayer**: Prompt tracking and logging
  - **Promptfoo**: Prompt testing toolkit
  - **OpenAI Evals**: Model vs model comparison

- **Integration with CI/CD**:
  - Regression tests for prompts
  - Threshold-based alerting on quality drops

4 Agentic AI & LLMOps

Session 30: Capstone Project 2 – Build an Agentic AI System (Advanced)

Goal: Build and deploy a complete agentic AI system with RAG, vector DB, evaluation, and deployment.

Topics Covered:

- **Project Definition Examples**:
  - Legal contract analyzer
  - Sales email writer from CRM notes
  - Support ticket summarizer with RAG + memory

- **Architecture**:
  - Input → Vector Store → Retriever → Planner Agent → Tool → Summarizer Agent → Output
  - Use LangChain or CrewAI for orchestration
  - Deploy using LangServe or FastAPI

- **Evaluation Integration**:
  - RAGAS for scoring
  - LangSmith for pipeline trace logging
  - TruLens for response quality rating

www.skillfyme.com