# Theme

# From Agentic Framework to AI-based Artificial Engineer

# NexComm 2025 & DigitalWorld 2025

# Open Discussion #5

NICE
MAY 2025

## Ignitor

## Prof. Dr. Petre Dini, IARIA, USA/EU

## Driver

## Prof. Dr. Mohamed El-Darieby, Ontario Tech University, Canada


# AI, AI, AI, …

NICE
MAY 2025

## AI landscape; Allover AI or AI where is needed?

## The changing landscape of data and AI

- The complexity of AI & bigdata technologies?
- Leveraging data and insights
- Building a data-driven culture?
- Enhancing decision-making with AI

## Rapid technological advancements & changes?

- Navigating the AI ecosystem: How  cohesive these AI efforts are?
- Overall resource allocation and skill gaps challenges

## Mechanic engineer vs AI engineer


# Themes under discussion

NICE
MAY 2025

Tools for AI engineers

AI-engineer domains

AI pipe engineering stack

LLM-specific engineering stack
Real-world LLM use case

AI agents

Agentic engineering

AI vs Agentic Engineering

Goals and Intent

Conflicting goals and conflict mediation

## Q: Formalisms


# Tools for AI-engineers

NICE
MAY 2025

## Tools for AI engineers

**Designing machine learning and deep learning models**

- **Managing data pipelines and preprocessing**
- **Deploying models into production environments**
- **Monitoring model performance and updating models**
- **Ensuring compliance, explainability, and robustness**

### 1. Model Development Tools

Tool || Purpose

> TensorFlow, PyTorch || Core deep learning frameworks
> scikit-learn || Classic ML algorithms & pipelines
> Keras || High-level neural network API
> Hugging Face Transformers || Pretrained LLMs and NLP models

### 2. Data Engineering Tools

> Apache Airflow || Workflow orchestration
> Pandas / Dask ||Data manipulation and parallel processing

Expectations: Data quality and validation checks

### 3. Model Deployment & Serving

> TensorFlow Serving / TorchServe || Serving models as APIs
> ONNX Runtime|| Cross-platform model inference
> MLflow / Seldon Core / BentoML || Model tracking, packaging, deployment

### 4. Monitoring and Observability

> WhyLabs / Evidently AI || Monitor drift and model quality
> Prometheus + Grafana || Infrastructure metrics and dashboards
> Arize AI / Fiddler AI || ML model observability and bias detection

### 5. MLOps and Versioning

> DVC (Data Version Control) || Track datasets and experiments
> Weights & Biases / Comet.ml || Track experiments, compare results
> Kubeflow / MLflow Pipelines || End-to-end ML workflows

### 6. AutoML and No-code AI

> Google AutoML / Azure ML Studio || Automated ML workflows
> DataRobot / H2O.ai || Business-friendly model training
> RunwayML / MakeML || Visual AI tools for creative work

### 7. Security, Fairness, and Explainability

> IBM AI Fairness 360 / Aequitas || Bias detection and mitigation
> LIME / SHAP || Model interpretability
> Adversarial Robustness Toolbox (ART) || Security testing for ML models


# AI-Engineer Domains

NICE
MAY 2025

## Who Uses These Tools?

**AI/ML Engineers – for full-cycle model development and deployment**

**Data Engineers – for data pipelines, ingestion, and preprocessing**

**MLOps Specialists – for CI/CD, versioning, and monitoring (Continuous Integration/Continuous Development)**

**Domain Experts – for interacting via AutoML and low-code/no-code tools**

(i) Typical AI Pipeline Stack: General AI Engineering Projects
This refers to end-to-end workflows commonly used in ML/AI applications such as image classification, predictive analytics, recommendation systems, etc

(ii) LLM-Specific Engineering Stack: Tools & Real-World Combos
This focuses on projects involving Large Language Models (LLMs) such as GPT-4, LLaMA, Claude, etc., especially for chatbots, retrieval-augmented generation (RAG), summarization, agents, and more.

(iii) Example: Real-World LLM Use Case – Legal Document Search Assistant


# Typical AI Pipeline Stack

NICE
MAY 2025

| Stage | Purpose | Popular Tools |
|---|---|---|
| **Data Collection & Ingestion** | Importing raw data from sources | Apache Kafka, Apache Nifi, AWS Kinesis, web scraping (BeautifulSoup, Scrapy), APIs |
| **Data Storage** | Storing raw/processed data | PostgreSQL, MongoDB, Amazon S3, Google BigQuery, Delta Lake, Hadoop HDFS |
| **Data Preprocessing** | Cleaning, transformation, feature engineering | Pandas, NumPy, PySpark, Dask, Scikit-learn preprocessing |
| **xperimentation & Modeling** | Model building, training, evaluation | Scikit-learn, TensorFlow, PyTorch, XGBoost, Keras |
| **Model Tracking & Management** | Versioning and reproducibility | MLflow, Weights & Biases, DVC (Data Version Control), Neptune.ai |
| **Model Serving** | Deploying models into production | FastAPI, TensorFlow Serving, TorchServe, BentoML, Kubernetes |
| **Monitoring & Logging** | Observability, model drift detection | Prometheus, Grafana, Evidently AI, Seldon Core, Arize, Kibana |
| **CI/CD Pipelines** | Automation of ML workflows | GitHub Actions, GitLab CI, Jenkins, Kubeflow Pipelines, Airflow, ZenML |


# LLM-Specific Engineering Stack

NICE
MAY 2025

| Layer | Purpose | Popular Tools |
|---|---|---|
| **LLM Backend** | Core model powering the application | OpenAI API (GPT-4), Claude, PaLM, LLaMA, Mistral, Cohere |
| **Orchestration/Frameworks** | Toolkits for chaining LLM prompts/ functions, agents | **LangChain**, **LlamaIndex**, Haystack, Semantic Kernel |
| **Embedding Models** | Creating vector representations of text | OpenAI Embeddings, HuggingFace Transformers, Cohere, Sentence-Transformers |
| **Vector Databases** | Storage and retrieval of semantic embeddings (for RAG, similarity search) | **Pinecone**, **Weaviate**, FAISS, Qdrant, Milvus |
| **Data Connectors / Loaders** | Ingesting and formatting data for LLM use (RAG, fine-tuning) | LlamaIndex connectors, Unstructured.io, LangChain DocumentLoaders |
| **Prompt Engineering / Templates** | Design of prompt flows and templates | LangChain PromptTemplates, Microsoft Guidance, PromptLayer |
| **Memory / Session Storage** | Managing context over multiple interactions | LangChain Memory, Redis, MongoDB, in-memory buffers |
| **Agent Interfaces** | Tools for building LLM-driven agents with tools and reasoning abilities | LangChain Agents, ReAct framework, AutoGPT, CrewAI |
| **App Interface / APIs** | Exposing the app (web or API endpoints) | Streamlit, Gradio, FastAPI, Flask, Vercel, NodeJS |
| **Monitoring / Evaluation** | Measuring LLM output quality, latency, prompt cost | LangSmith, PromptLayer, OpenTelemetry, human eval tools |

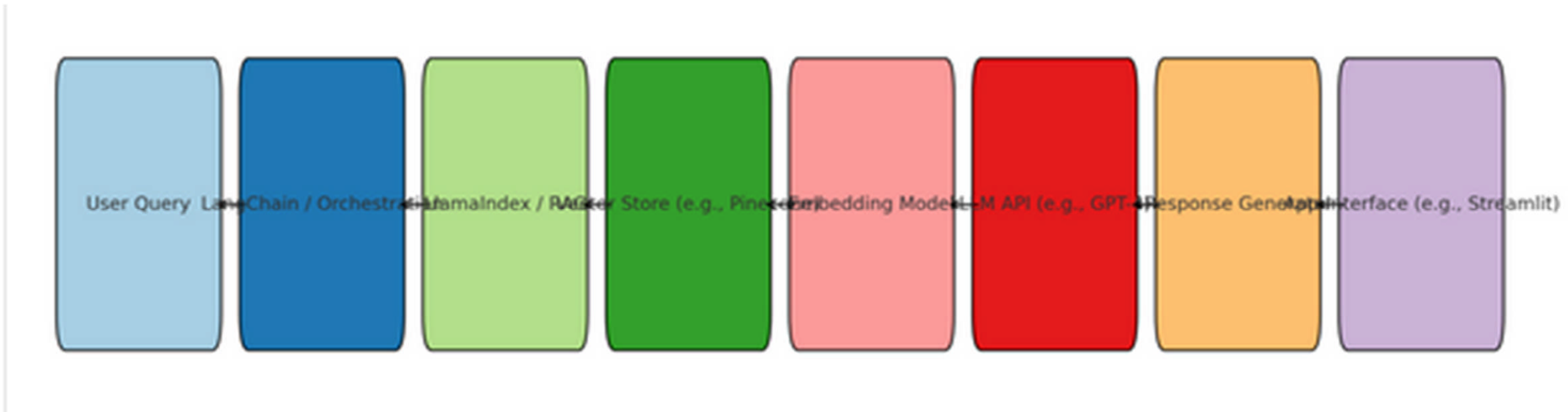
# Real-World LLM Use Case

NICE
MAY 2025

Use Case: Summarize, retrieve, and answer questions about legal documents.

Tools Used:

- LlamaIndex: To build a document index from legal PDFs
- OpenAI Embeddings + Pinecone: For storing and retrieving vectorized data
- LangChain: For chaining user queries → embedding search → LLM generation
- Streamlit: For building the interactive UI
- FastAPI + Redis: For backend API and session memory
- LangSmith: For tracking prompt chains and debugging output quality



# AI Engineer (Agent) - Virtual Entity

NICE
MAY 2025

## Core Capabilities of a Virtual "AI Engineer"

| Capability | Description |
|---|---|
| Model Architecting | Selects models based on data type (e.g., CNNs, LLMs, GNNs) |
| Hyperparameter Tuning | Uses optimization techniques (Bayesian, Grid Search) |
| Pipeline Automation | Builds end-to-end data/model training workflows |
| Deployment Automation | Pushes models into staging or production environments |
| Documentation & Reporting | Auto-generates experiment reports, code comments |
| Continual Learning Mgmt | Suggests or implements retraining schedules |
| Debugging Assistant | Identifies performance regressions, training bugs |
| Bias and Compliance Checks | Flags fairness, privacy, or explainability issues |

### Underlying Technologies

> LLMs | GPT-4, Claude, Mistral, or open-source models (fine-tuned for engineering tasks)
> Agent Frameworks | LangChain, AutoGen, CrewAI, AgentVerse
> Tool Plugins | Code runners, databases, version control, Docker, etc.
> Memory/Planning | ReAct, Chain-of-Thought, RAG, scratchpads, vector memory
> Environment | Often containerized (Docker, Replit, VSCode in-browser)

## Examples of Platforms Creating "AI Engineer" Entities

> Devin by Cognition

A fully autonomous AI software engineer that can plan, write, debug, and test code with no human intervention (still in preview

> GitHub Copilot X (with Agents)

Goes beyond code completion; can scaffold apps, generate entire classes, and collaborate over time.

> AutoGPT / AgentGPT

Experimental open-source agents that can be instructed to achieve high-level engineering goals via tool use and iterative planning.

> OpenDevin (open-source fork)

Tries to mimic Devin's architecture — acts like a terminal-based AI engineer that uses planning + code execution.

> C> odeWhisperer (AWS) and Tabnine

Autocomplete-style assistants but heading toward semi-autonomous behavior.

> LangChain Agents / CrewAI

Build modular agent teams: one can play the "AI engineer" role in an LLM-powered workflow.


# Agentic Engineering

NICE
MAY 2025

An "**AI Engineer**" as a virtual agent (an autonomous LLM-based entity that performs AI engineering tasks),
**Agentic Engineering** as a discipline or paradigm (engineering systems of agents that plan, act, and learn over time).

## AI Engineer (as a virtual agent)

This is a specialized role or embodied skillset within a broader system.
It refers to an LLM-powered autonomous agent that:
- Writes code, designs models, debugs, deploys
- Acts like a virtual software engineer
- Is task-focused (e.g., “build me an object detector”)
- May use planning, tool use, memory, and execution environments (e.g., shell, browser, Python interpreter)

Example:

Devin, GitHub Copilot + agents, or a LangChain/CrewAI agent with the "AI Engineer" role.

## Agentic Engineering

This is a new field of engineering focused on the design, orchestration, and safety of intelligent agents, especially LLM-based ones.
It involves:
- Creating multi-agent ecosystems
- Managing goals, delegation, planning, negotiation
- Enforcing safety, alignment, and controllability
- Addressing non-determinism, long-horizon actions, and memory evolution

## Related challenges include:

- Tool integration
- Agent teaming and coordination
- Goal disambiguation and intent refinement
- Autonomy vs. oversight balancing
- Temporal abstraction (short tasks vs. lifelong learning)


# AI vs Agentic Engineering

NICE
MAY 2025

Their Relationship

| Aspect | AI Engineer (Agent) | Agentic Engineering |
|---|---|---|
| What | A software agent that performs AI tasks | A discipline to build, manage, and evaluate agents |
| Cognitive Role | Acts as a specialized skill worker | Designs systems of cognition and delegation |
| Technologies Used | LLMs, memory, RAG, tool APIs | Agent platforms, safety modules, coordination logic |
| Output | Trained models, deployed pipelines | Robust multi-agent systems, reliable interfaces |
| Scope | One agent with a fixed or growing role | Multi-agent environments, emergent behaviors |
| Example | Devin generating a neural net | CrewAI orchestrating 5 agents for research |

**What Is Goal Generation in Agentic Systems?**

**Goal generation is the process by which an agent (like the AI Engineer) determines what it should do next. This includes:**

- Recognizing new needs or opportunities
- Transforming open-ended tasks into actionable objectives
- Aligning tasks with long-term system purpose or constraints

**1. Components of Goal Generation**

| Source | Example |
|---|---|
| Human Prompt | "Build a model to classify pneumonia in X-ray images" |
| Self-reflection | Agent detects pipeline drift and sets a goal to retrain |
| Environment State | New data availability triggers model update |
| Upstream Agent | A supervisor agent delegates "optimize hyperparameters" |


# Basis: Goals

NICE
MAY 2025

## 2. Cognitive Mechanisms Involved

Mechanism || Role

Intent Interpretation | Parsing vague instructions into well-scoped tasks
Decomposition || Breaking goals into sub-goals or tasks (e.g., Chain-of-Thought)
Prioritization & Relevance Filtering || Choosing which goals to pursue first
Goal Memory and Reuse || Recall past goals and outcomes for reuse or adjustment
Dynamic Adaptation || Modify goals in response to failures, new input, or success

## 3. Formal Representations of Goals

Goals can be represented internally as:

> Structured Task Objects: {"type": "train_model", "dataset": "lungXrays", "metric": "accuracy"}
> Planning Nodes in a hierarchy or workflow graph
> Natural Language Targets with semantic frames (via LLM embeddings)

## 4. Techniques Used for Goal Generation in Modern Systems

Technique || Use Case

> LLM-based reasoning (e.g., ReAct, Plan-and-Act) || Converts vague goals into sequenced actions
> Prompt templating + examples || Guides goal shaping via few-shot prompting
> Planning agents (e.g., BabyAGI, AutoGPT) || Create and schedule subgoals dynamically
> Graph-based Planning (RAG + Tools) || Connect facts to derive new tasks
> Goal negotiation || Collaborating with other agents to clarify or redefine goals

## Open Research Challenges

Ambiguity Resolution - Understanding under-specified goals

Alignment & Ethics = Generating goals aligned with user intent and safety constraints

Long-Horizon Goals
Managing goals that take days/weeks to complete

Interruptibility
Being able to stop, modify, or reprioritize goals mid-flight

Meta-goals
Agents improving their own goal-setting heuristics

# Basis: Goals-Intent



## 1. Formal representation

Let each goal **G** be represented as a structured tuple:

G=⟨ID,Intent,InputContext,Constraints,SuccessCriteria,Priority⟩

ID: Unique goal identifier
Intent: Natural language or formal description
InputContext: Set of facts, data, or triggers
Constraints: Temporal, logical, resource, or ethical boundaries
SuccessCriteria: Quantified metrics, logical end states
Priority: Ranking among competing goals

Example:

G1=⟨"train_fraud_model","Improve fraud detection",D,accuracy>0.9,time < 2h,P2⟩
G1=⟨"train_fraud_model","Improve fraud detection",D,accuracy>0.9,time < 2h,P2⟩

## 2. Goal Decomposition

Task Graph

> Construct a directed acyclic graph (DAG) or a hierarchical task network (HTN):
> TG=(N,E)
> TG=(N,E)
> N: nodes representing sub-tasks TiTi
> E: edges denoting dependency/order
> Each node has its own goal-spec-like tuple

Example:

T1=load datasetT1=load dataset
T2=preprocess dataT2=preprocess data
T3=train XGBoost, SVM, NeuralNetT3=train XGBoost, SVM, NeuralNet

## 3. Interaction & Conflict Detection

> Define each task with a resource claim R(Ti)R(Ti) and effect set E(Ti)E(Ti).
> Conflicts are detected by:
> Temporal Conflicts: Ti≺TjTi≺Tj but dependent inputs unavailable
> Resource Conflicts: R(Ti)∩R(Tj)≠∅R(Ti)∩R(Tj)=∅ with overlap
> Semantic Conflicts: contradictory constraints or models
> Speculative Loops: if goals regenerate themselves indefinitely

Formal predicate:

> Conflict(Ti,Tj)⇒Res(Ti)∩Res(Tj)≠∅∨¬Compatible(E(Ti),E(Tj))
> Conflict(Ti,Tj)⇒Res(Ti)∩Res(Tj)=∅∨¬Compatible(E(Ti),E(Tj))

# Main issue

## Agentic Engineering

**How to align business goals, problem solution, and agents goals?**

**Automatic Goal Generation?**

**Similarity with what an LLM produces these days, e.g., the plans they produce?**

**How would you formalize this? Ho to ensure they solve the problem?**

**What about outcome quality?**

**What about conflicting goals, the design trade-offs?**

**Establishing robust infrastructure and processes**

# Q: FORMAL APROACHES

# THE STAGE IS YOURS