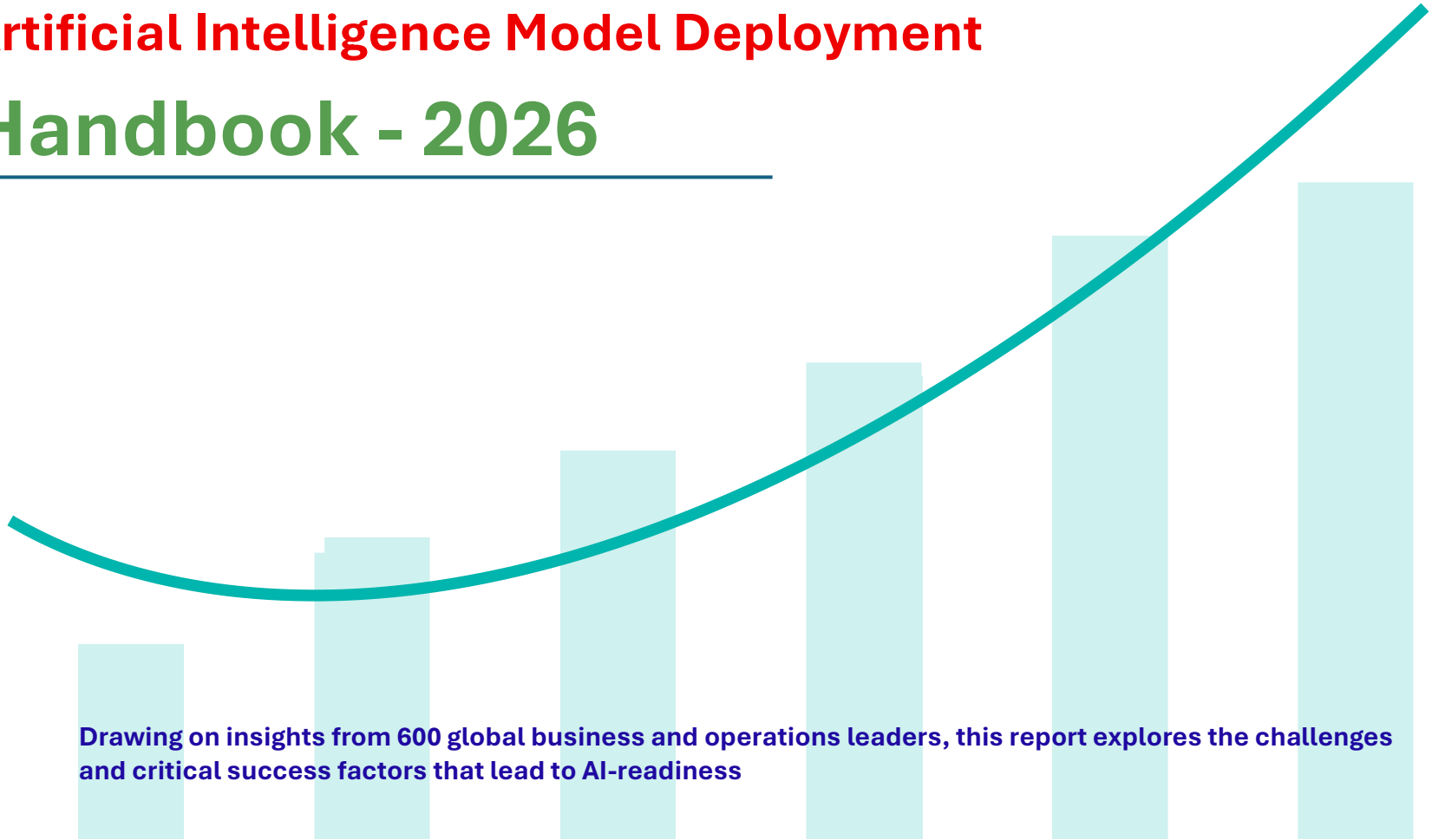
IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

IARE Industry Readiness on Artificial Intelligence Model Deployment

Handbook - 2026



Drawing on insights from 600 global business and operations leaders, this report explores the challenges and critical success factors that lead to AI-readiness

INDEX

Foreword	1
Industry Readiness	3
Program Components	8
1. Integrating generative Al into fundamental programming classes	8
2. Machine Learning Algorithms	11
3. Al Engineering	16
4. Al in Software Engineering	21
5. Al Pair Programming	31
6. Deep-Tech Al Trajectories	37
Industry Requirements	38
Key Findings	39

Foreword

The AI Engineering program provides hands-on experience in designing, developing, and deploying intelligent applications using Large Language Models (LLMs), LangChain, AI Agents, and modern software engineering practices. Built on a project-based learning approach, the program combines advanced System Design methodologies, Software Development Life Cycle (SDLC) principles, and AI-powered development techniques to prepare students for the future of intelligent software systems and digital transformation.

Students develop strong foundations in Artificial Intelligence, Machine Learning, Deep Learning, Generative AI, Agentic AI, Data Science, Robotics, and Augmented & Virtual Reality while gaining practical exposure to industry-standard tools, frameworks, and platforms. The curriculum is designed to cultivate technical expertise, analytical thinking, and problem-solving capabilities through real-world application development and innovation-driven learning.

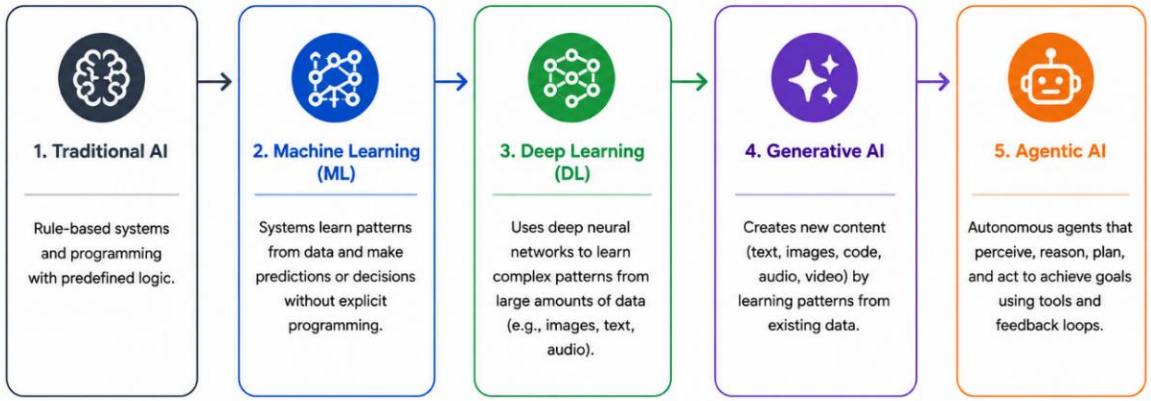


Figure 1. The Evolution of AI

Figure 1 illustrates the evolutionary progression of Artificial Intelligence across five distinct paradigms from rule-based Traditional AI through Machine Learning, Deep Learning, and Generative AI, to the emerging frontier of Agentic AI.

Figure 2 shows how Machine Learning is positioned as a major subfield of Artificial Intelligence, with supervised learning, unsupervised learning, reinforcement learning, and deep learning forming important branches. It further shows how deep learning includes neural network architectures such as ANN, RNN, CNN, and Transformers, with modern language models such as BERT, GPT, LLaMA, and Mistral emerging from transformer-based architectures.

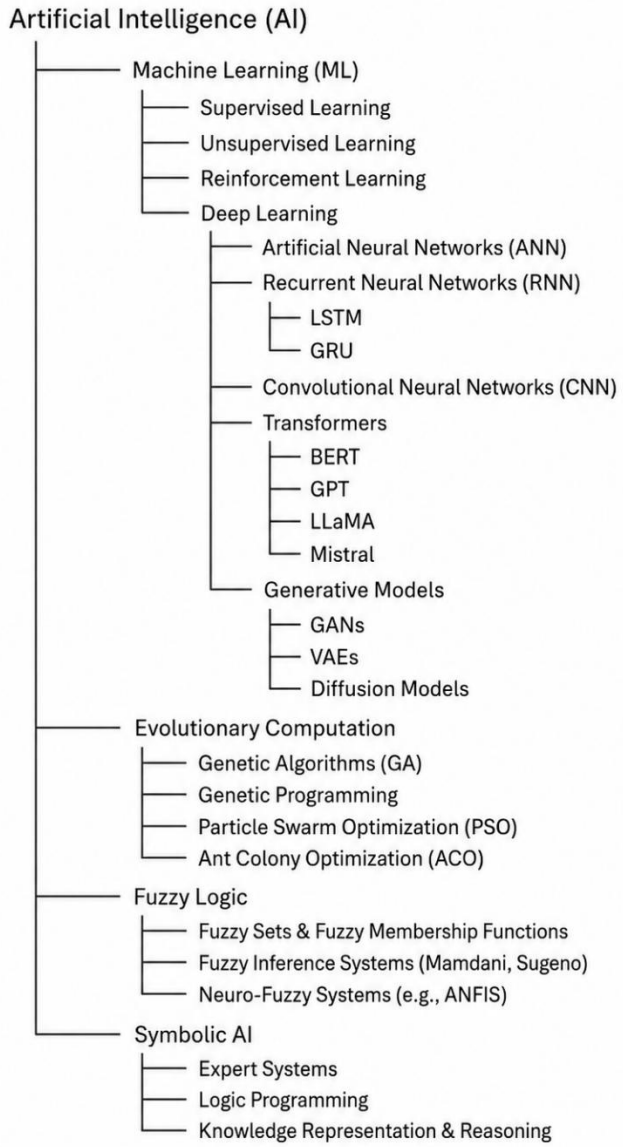


Figure 2. Hierarchical Relationship among AI, ML, Deep Learning, Neural Networks, and Transformer-Based Models

Through boot camps, workshops, internships, industry projects, hackathons, certification programs, and mentorship from AI experts and industry leaders, students gain valuable exposure to enterprise AI solutions, intelligent systems, AI copilots, and emerging technologies. The program equips learners with the skills and industry-ready competencies required to excel in high-demand roles such as AI Engineer, LLM Engineer, Machine Learning Engineer, Software Engineer, Data Scientist, and Backend Developer.

Industry Readiness

AI readiness as a mechanism linking educational inputs to career-related outcomes. AI readiness can be understood as the extent to which individuals are prepared to engage with AI technologies in applied contexts, reflecting not only personal competence but also the extent to which learning environments enable such engagement. Exposure to laboratory activities, group work, and experiential learning opportunities plays a central role in shaping what students ultimately gain from AI-related education.

Students who possess higher levels of AI literacy are more likely to perceive AI technologies as accessible, relevant, and useable in practice.

Table 1 presents a comparison of the key competencies required for AI-ready students from the perspectives of academics and Employers. Both groups recognize the importance of combining AI-related skills with essential human capabilities, but they prioritize these competencies differently. According to academics the highest priority is the effective use of AI tools, reflecting the growing need for students to understand and leverage AI technologies in academic and professional environments. Their second priority is the integration of human judgment with AI capabilities, emphasizing the ability to make informed decisions while collaborating with intelligent systems. Adaptability is ranked as the third priority, highlighting the need for continuous learning and flexibility in a rapidly evolving technological landscape.

Table1. Key competencies for AI-ready graduates: employer VS educator views

Market	Perspective	Priority 1	Priority 2	Priority 3
Overall	Academics	AI tool use	Human judgment + AI capabilities	Adaptability
	Employers	Communication and collaboration skills	Adaptability	Human judgment + AI capabilities

Employers, on the other hand, place the greatest emphasis on communication and collaboration skills, indicating that the ability to work effectively with teams and stakeholders remains critical in the workplace. Adaptability is identified as the second most important

competency, reflecting the demand for professionals who can quickly adjust to new technologies and changing business requirements. The combination of human judgment and AI capabilities is ranked third, underscoring the importance of balancing technological proficiency with critical thinking and decision-making skills.

The optimal AI-ready graduate- Key capabilities and skill competencies

The Figure 3 represents a consolidated vision that runs from student to industry ready graduate by building a more effective path from the classroom to the workplace.

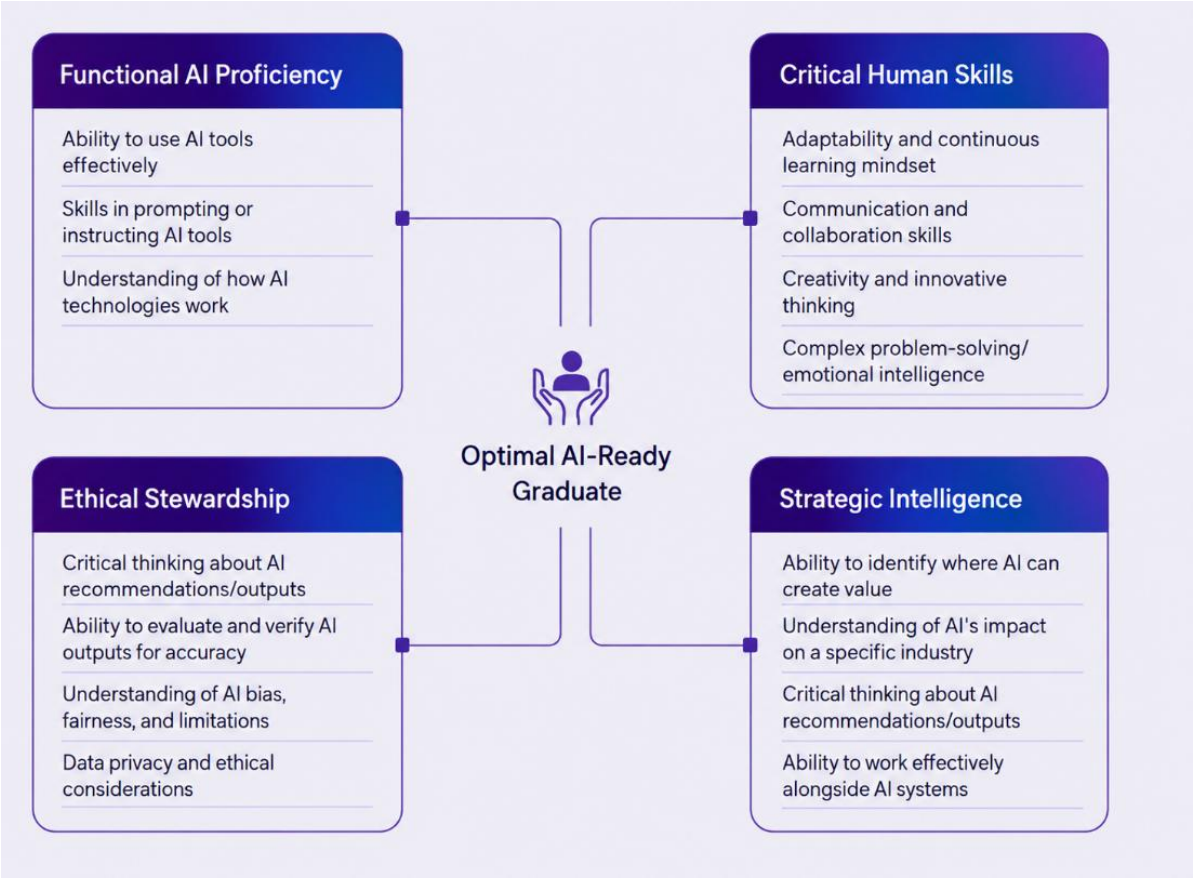


Figure 3. The optimal AI-ready graduate

For a contemporary graduate, readiness is a multifaceted construct that combines the following:

1. **Functional Proficiency**

   From day one, students must arrive functionally fluent in workplace specific tools by developing dossier of completed projects. They are able to take standard AI tools and apply them to a professional workflow. This set of skills represents the type of human-in-the-lead aptitude that employers demand. While across markets, learners prioritize the "ability to use AI tools effectively" as the top-rated requirements.

Key Competencies	Ability to use AI tools effectively; Skills in prompting or instructing; Understanding of how AI technologies work.

2. **Strategic Intelligence**

   The AI-ready graduate will have successfully cultivated the ability to identify exactly where AI adds value and where it creates risk, a skill that 1 in 3 employers today report is of high importance when hiring into their organizations. They will have an appreciation for how AI can be deployed as something more than a productivity or efficiency shortcut.

Key Competencies	Ability to identify where AI can create value; Understanding of AI's impact on a specific industry; Critical thinking about AI recommendations/outputs; Ability to work effectively alongside AI systems.

3. **Ethical Stewardship:**

   In an era of ubiquitous AI, the AI-ready student must be equipped to mitigate risk. They will understand bias, fairness, data privacy, and data integrity. They will be confident in navigating, and complying with, institutional and professional policies.

   This set of skills involves a student's capability to serve as an ethical filter and a risk mitigation manager for employers, focusing on safety, integrity, and the ethical deployment of technology.

Key Competencies	Critical thinking about AI recommendations/ outputs; Ability to evaluate and verify AI outputs for accuracy; Understanding of AI bias, fairness, and limitations; Data privacy and ethical considerations.

4. Critical Human Skills:

The optimal graduate will possess a skillset that is valued for the things that AI cannot replicate. Aware that current models have a finite shelf-life, they will bring an adaptable, agile mindset and value opportunities to learn and ensure they remain relevant as the pace of change accelerates.

It is required; critical human skills are equally important to functional AI proficiency and half of all employers' rank communication and collaboration. This set of skills represents a graduate's competency to provide what no AI model can: human judgement, creative thinking, collaborative and emotional intelligence. Whereas AI automates execution, the AI-ready graduate possesses the relational and cognitive capabilities to bring purpose and direction.

Key Competencies	Adaptability and continuous learning mindset; Communication and collaboration skills; Creativity and innovative thinking; Complex problem-solving / Emotional intelligence.

Gen AI capabilities have evolved rapidly over the past two years

Over the past two years, AI has advanced in leaps and bounds, and enterprise-level adoption has accelerated due to lower costs and greater access to capabilities. Many notable AI innovations have emerged (Exhibit 1). For example, we have seen a rapid expansion of context windows, or the short-term memory of LLMs. The larger a context window, the more information an LLM can process at once. To

illustrate, Google's Gemini 1.5 could process one million tokens in February 2024, while its Gemini 1.5 Pro could process two million tokens by June of that same year. Overall, we see five big innovations for business that are driving the next wave of impact: enhanced intelligence and reasoning capabilities, agentic AI, multimodality, improved hardware innovation and computational power, and increased transparency.

Agentic AI is acting autonomously

The ability to reason is growing more and more, allowing models to autonomously take actions and complete complex tasks across workflows. This is a profound step forward. As an example, in 2023, an AI bot could support call center representatives by synthesizing and summarizing large volumes of data including voice messages, text, and technical specifications to suggest responses to customer queries. In 2026, an AI agent can converse with a customer and plan the actions it will take afterward for example, processing a payment, checking for fraud, and completing a shipping action.

Exhibit 1: Illustrative capabilities of gen AI platforms from select frontier labs, non-exhaustive

Organization	2022–23[1]	Jan 2025[2]
Anthropic	**Claude** → **Claude 3.5** • Not multimodal (text only) • Limited contextual understanding (difficulty with complex conversations) • No tool usage	• Multimodal (text, audio, and images) • Enhanced contextual understanding and coherence during long interactions • Experimental computer usage capability for some users
Google Gemini	**Google Bard** → **Gemini 2.0 Flash** • Not multimodal (text only) • Fair reasoning • Limited contextual understanding (difficulty with complex conversations) • Limited real-time data integration • Low personalization (limited adaptability)	• Multimodal (text, audio, and images) • Advanced reasoning (capable of multistep problem-solving and nuanced analysis) • Enhanced contextual understanding (maintains coherence in long dialogues) • Real-time data integration (from Google Search) • Advanced personalization (user context)
Meta	**Llama 1** → **Llama 3.3** • Not multimodal (text only • Fair reasoning • Limited contextual understanding (difficulty with complex conversations) • No API access	• Text-based (earlier versions were multimodal, LLaMa 3.2) • Advanced reasoning (capable of multistep problem-solving and nuanced analysis) • Enhanced contextual understanding (maintains coherence in long dialogues) • API access (tools for model and agent development)
Microsoft	**Phi-1** → **Phi-4** • Not multimodal (text only) • Fair reasoning (ie, limited to coding tasks) • Focused training (smaller, coding-focused data set)	• Multimodal (text, audio, and images) • Advanced reasoning (capable of multistep problem-solving and nuanced analysis) • Comprehensive training (diverse data)
OpenAI	**GPT-3.5** → **OpenAI o1** • Not multimodal (text only) • Fair reasoning ability (eg, scored high on SAT, but bottom 10% on bar examination) • Limited contextual understanding (difficulty with coherence in complex conversation) • Standard API access (for text generation)	• Multimodal (text and images) • Advanced reasoning (eg, top 10% on bar examination) • Enhanced contextual understanding (maintains coherence in long dialogues) • Advanced API access (supports multimodal inputs)

Multimodality is bringing together text, audio, and video

Today's AI models are evolving toward more advanced and diverse data processing capabilities across text, audio, and video. Over the last two years, we have seen improvements in the quality of each modality. For example, Google's Gemini Live has improved audio quality and latency and can now deliver a human-like conversation with emotional nuance and expressiveness. Also, demonstrations of Sora by OpenAI show its ability to translate text to video.

Hardware innovation is enhancing performance

Hardware innovation and the resulting increase in compute power continue to enhance AI performance. Specialized chips allow faster, larger, and more versatile models. Enterprises can now adopt AI solutions that require high processing power, enabling real-time applications and opportunities for scalability. For example, an e-commerce company could significantly improve customer service by implementing AI-driven chatbots that leverage advanced graphics processing units (GPUs) and tensor processing units (TPUs). Using distributed cloud computing, the company could ensure optimal performance during peak traffic periods. Integrating edge hardware, the company could deploy models that analyze photos of damaged products to more accurately process insurance claims.

PROGRAM COMPONENTS

The AI Model Deployment Program is structured around the following key competency areas:

1. **Integrating Generative AI into Fundamental Programming**
2. **Machine Learning Algorithms**
3. **AI Engineering**
4. **AI in Software Engineering**
5. **AI Pair Programming**
6. **Deep-Tech AI Trajectories**

1. INTEGRATING GENERATIVE AI INTO FUNDAMENTAL PROGRAMMING CLASSES

Generative AI (GenAI) tools like GitHub Copilot, Codeium and ChatGPT are rapidly changing how programming is taught and learnt. These tools can solve assignments with remarkable accuracy. GPT-4 achieved top-tier performance on several standardized exams, far surpassing earlier models such as Codex on many coding and reasoning benchmarks.

With such capabilities, researchers are shifting from asking, **"Should we teach with AI?" to "How do we teach with AI?"**

These results spearhead transformation in teaching fundamental programming courses. Integrating GenAI tools, help students tackle complex programming tasks while developing critical thinking and problem-solving skills.

Core skills for programming with Generative AI

Teaching programming with GenAI involves fostering a mix of traditional and AI-specific skills. The workflow when writing software with Copilot is:

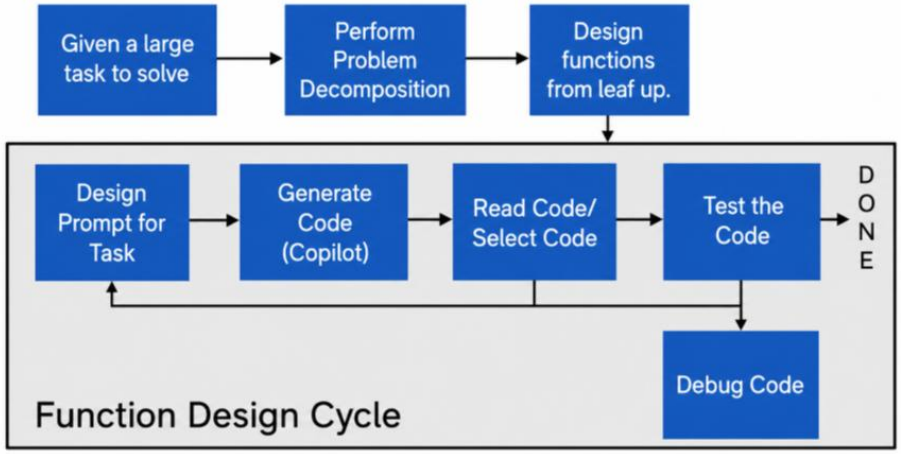


Figure 4. LLM-Integrated Function Design Cycle for Collaborative Programming

Figure 4 presents a structured Function Design Cycle that systematically guides students through the stages of problem decomposition, prompt engineering, LLM-assisted code generation, code selection, testing, and debugging. This iterative workflow operationalises the integration of Generative AI tools within foundational programming education, reinforcing both computational thinking and critical code verification competencies.

Writing software with GenAI applications, such as Copilot, needs to be approached differently to traditional programming tasks

- **Prompting and function design:** Students learn to articulate precise prompts for AI tools, honing their ability to describe a function's purpose, inputs, and outputs, for instance. This clarity improves the output from the AI tool and reinforces students' understanding of task requirements.

- **Code reading and selection:** AI tools can produce any number of solutions, and each will be different, requiring students to evaluate the options critically. Students are taught to identify which solution is most likely to solve their problem effectively.
- **Code testing and debugging:** Students practise open- and closed-box testing, learning to identify edge cases and debug code using tools like doctest and the VS Code debugger.
- **Problem decomposition:** Breaking down large projects into smaller functions is essential. For instance, when designing a text-based game, students might separate tasks into input handling, game state updates, and rendering functions.
- **Leveraging modules:** Students explore new programming domains and identify useful libraries through interactions with Copilot. This prepares them to solve problems efficiently and creatively.

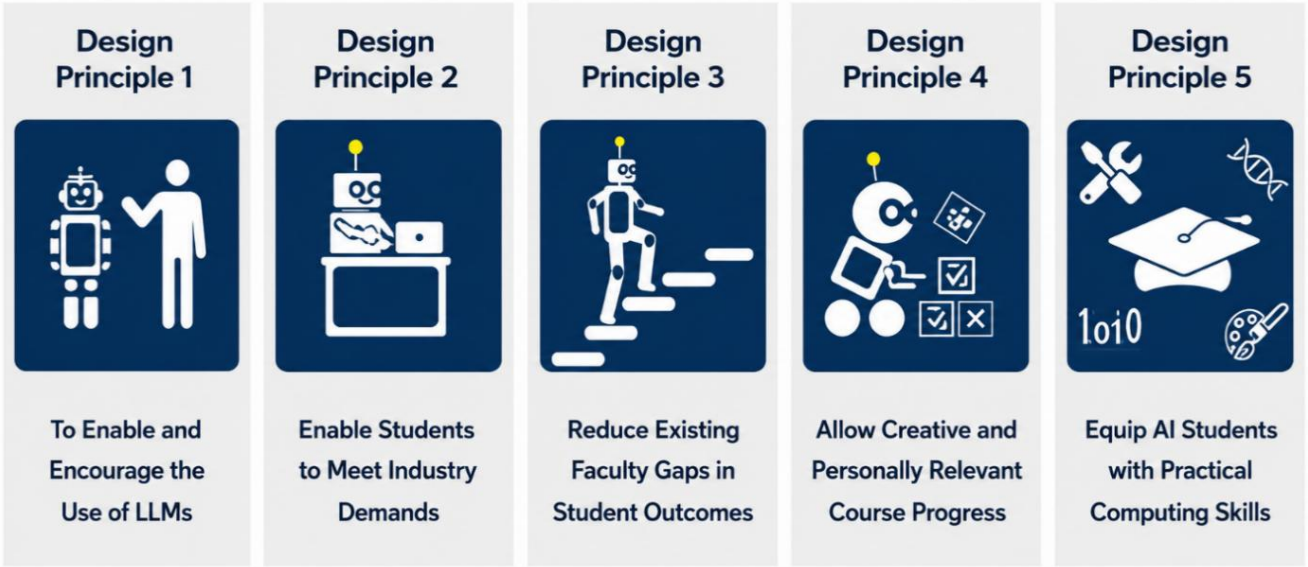


Figure 5. Design Principles for LLM-Integrated Fundamental Programming

Figure 5 describes the five design principles governing LLM-integrated fundamental programming, structured around the full spectrum of programming competencies, encompassing problem decomposition, code explanation, code writing, code tracing, code modification, code testing, and code debugging. The program systematically integrates debugging, testing, and problem decomposition as core programming competencies, ensuring that students develop a disciplined and structured approach to software development from the earliest stages of their academic training.

Practical applications in the classroom

All programming classes **use GitHub Copilot.** The course includes with complex problem-solving tasks, creative coding assignments and open-ended problems allowing students to explore their interests while applying the skills they have learnt.

The creative coding assignments covers the following areas:

- **Mini data handling:** Students use Kaggle datasets to explore questions related to their fields of study; for example, on neuroscience dataset - analyse stroke data. The projects encouraged interdisciplinary thinking and practical applications of programming.
- **Image manipulation:** Students work with the Python Imaging Library (PIL) to create to create **photo montages** and apply filters to images, showcasing their creativity and technical skills.
- **Story-based programming:** Students create interactive storytelling programs where users choose different actions, helping them practice conditional logic and program flow.
- **Puzzle and logic challenges:** Sudoku helpers, simple maze navigation, or mathematical puzzles encourage algorithmic thinking and debugging skills.
- **Game development:** A project focus on designing text-based games encouraged students to break down problems into manageable components while using Al tools to generate and debug code.

These beginner-level creative coding assignments make programming enjoyable, interactive, and application-oriented while helping students build confidence in fundamental coding concepts through hands-on learning and experimentation.

2. MACHINE LEARNING ALGORITHMS

Learning Machine Algorithms provides a comprehensive introduction to core machine learning and deep learning concepts, spanning supervised and unsupervised learning, neural network architectures, computer vision, and natural language processing. It progresses from classical algorithms to modern deep learning frameworks and state-of-the-art models. This also introduces Generative AI, equipping students with both theoretical understanding and hands-on experience using industry-standard libraries.

Key skills you gain

1. **Machine Learning Fundamentals:** Build predictive models using supervised and unsupervised learning techniques, feature engineering, model evaluation, and optimization methods.

2. **Data Analysis & Model Development:** Prepare, analyze, and visualize data using Python, NumPy, Pandas, and Scikit-learn to develop real-world machine learning solutions.
3. **Deep Learning & Neural Networks:** Design and train Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs) using TensorFlow and PyTorch.
4. **Advanced AI Applications:** Develop intelligent systems for computer vision, natural language processing, recommendation systems, and generative AI applications with industry-standard frameworks.

CURRICULUM

Artificial Neural Networks

Machine learning has evolved rapidly, driving a surge of interest in Artificial Neural Networks (ANNs. They are computational models that draw inspiration from the structure and operations of the human brain. They comprise interconnected nodes, or artificial neurons, organized in layers. Neural networks are designed to process and examine complex data, recognize patterns, and make predictions or decisions based on their learned knowledge. They excel in solving problems that are difficult to define using traditional programming techniques or when explicit rules are unavailable.

Deep learning refers to the process of training these networks, allowing them to process raw or minimally processed data and automatically identify patterns a concept known as feature learning or representation learning.

You will learn both the theory behind these technologies and how to apply them to real problems in machine learning and data mining. Through hands-on exercises, you will also practise training and evaluating different ANN models, helping you develop a strong foundation in the field.

Topics Covered

Foundations of Neural Networks:	Organization of the Human Brain, Biological Neuron: Structure and Function, Features and Working of Biological Neural Networks (BNN), Biological vs Artificial Neurons, Artificial Neuron Models, Characteristics of Artificial Neural Networks (ANN), Comparison of Biological Neural Networks and Artificial Neural Networks, Models of Neurons.

Perceptron and Learning Principles:	Principles of Neural Network Design, McCulloch-Pitts Neuron Model, Perceptron Learning Model, Adaline and Madaline Models, Basic Learning Laws, Linear Separability, XOR Problem and Its Solution, Single Layer Neural Networks, Multilayer Perceptrons (MLP).
Backpropagation Networks:	Backpropagation Learning, Feedforward Neural Networks, Input Layer / Hidden Layer / Output Layer Computations, Backpropagation Algorithm, Error Calculation and Weight Updates, Selection of Learning Parameters, Number of Hidden Nodes, Stability and Convergence Analysis.
Self-Organizing, Counter Propagation and Associative Memory Networks:	Counter Propagation Networks, Kohonen Layer, Grossberg Layer, Self-Organizing Maps (SOM), Competitive Learning, Clustering and Visualization Applications, Hopfield Networks, Energy Functions and Network Stability, Bidirectional Associative Memory (BAM), Pattern Storage and Recall, Applications of Associative Memory Networks.
Stochastic and Adaptive Models:	Boltzmann Machines, Stochastic Learning, Energy-Based Models, Boltzmann Learning Algorithm, Applications of Boltzmann Machines, Fundamentals of ART Networks, ART Architecture and Components, Features of ART Models, Stability-Plasticity Dilemma, Character Recognition using ART Networks.
Libraries:	NumPy, SciPy, Matplotlib, Seaborn, Pandas, Scikit-learn, TensorFlow, Keras, MiniSom, PyTorch

Statistical Machine Learning

Statistical Machine Learning is a branch of AI/ML that builds predictive or descriptive models by applying statistical and probabilistic principles to data using mathematical foundations like probability distributions, hypothesis testing, information theory, and optimization to learn patterns, rather than explicit rule-based logic or deep neural architectures. These models discover meaningful, valid, and actionable knowledge from large volumes of data. It combines data preprocessing, statistical analysis, machine learning, pattern recognition, and visualization techniques to identify hidden patterns, relationships, trends, and insights that support informed decision-making.

Topics Covered

Classification algorithms:	k-Nearest Neighbors (KNN), Decision Trees (ID3, C4.5, CART), Naïve Bayes Classifier, Random Forest, and Support Vector Machines (SVM).
Clustering:	K-Means, DBSCAN, and Hierarchical Clustering
Association Mining:	Apriori and FP-Growth
Regression algorithms:	Linear Regression, Polynomial Regression, and Logistic Regression.
Ensemble Methods:	Random Forest, Bagging, Boosting - AdaBoost, XGBoost, CatBoost.
Model Evaluation Metrics:	Classification metrics - Confusion Matrix, Accuracy, Precision, Recall (Sensitivity), F1-Score, ROC-AUC.
Libraries:	NumPy, Matplotlib, Pandas, scikit-learn, Seaborn

Deep Learning

Deep learning is a subfield of machine learning. Deep learning models are characterized by having multiple hidden layers (referred to as deep neural networks) between the input and output layers. These deep neural networks can learn hierarchical representations of data by automatically discovering intricate features and patterns at different levels of abstraction. Deep learning models excel at learning from large and complex datasets, and they have achieved remarkable success in various AI tasks, such as computer vision, natural language processing, speech recognition, and more.

Deep learning is a powerful technique within the broader context of neural networks, enabling the development of highly advanced AI models.

Topics Covered

Convolutional Neural Networks (CNN):	Convolutional layers (filters, kernels, stride, padding), Pooling layers (Max / Average pooling), Flattening, Fully Connected (FC) layers, Softmax output.
Classic CNN Architectures:	LeNet-5, AlexNet, VGGNet (VGG-16/VGG-19), GoogLeNet / Inception, ResNet (Residual Networks), MobileNet.
Sequential Models:	Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU).
Libraries and Frameworks:	PyTorch, TensorFlow, Keras, torchvision, NumPy, Matplotlib, Scikit-learn, Hugging Face Transformers, OpenCV, Pandas, tqdm, ONNX Runtime.

Computer Vision Models

Develop expertise in designing intelligent vision systems capable of detecting, classifying, segmenting, and understanding visual data. Students work with state-of-the-art object detection and segmentation models used in autonomous systems, healthcare, surveillance, and industrial automation.

Topics Covered

Single-Stage Object Detection Models:	YOLO (v8/v9/v10/v11), SSD (Single Shot Multibox Detector), and RetinaNet.
Two-Stage Object Detection Models:	R-CNN (Regions with CNN), Fast R-CNN, Faster R-CNN.
Semantic Segmentation Models:	U-Net, DeepLabv3+, PSPNet (Pyramid Scene Parsing Network), SegNet, SAM2 (Segment Anything Model 2), Mask2Former.
Frameworks, Libraries & AI Model Platforms:	PyTorch, TensorFlow, Ultralytics, Detectron2, torchvision, OpenCV, segmentation-models-pytorch, Hugging Face Transformers, ONNX Runtime, Albumentations, NumPy, Matplotlib.

Natural Language Processing (NLP)

Learn how machines process, understand, and generate human language. Students explore modern NLP techniques ranging from text preprocessing and language representation to transformer-based large language models used in chatbots, search engines, virtual assistants, and generative AI applications.

Topics Covered

Natural Language Processing (NLP):	Origins, Challenges, and Applications of NLP, Text tokenization, Normalization, and Language modeling, Bag of Words (BoW).
Preprocessing and Text Representation:	Stemming, lemmatization, stop-word removal, morphological analysis, Bag-of-words, TF-IDF, word embeddings (Word2Vec, GloVe), ELMo (Embeddings from Language Models).
Syntactic Analysis and Semantics:	Part-of-Speech (POS) tagging, constituency parsing, dependency parsing, Named Entity Recognition (NER), distributional semantics, and topic modeling (LDA, NMF).
Transformer-Based Models:	BERT (Bidirectional Encoder Representations from Transformers), GPT (Generative Pre-trained Transformer), Transformer-XL, RoBERTa.
Libraries and Frameworks:	NLTK, spaCy, Gensim, Scikit-learn, Hugging Face Transformers, PyTorch, TensorFlow, Keras, NumPy, Pandas, ONNX Runtime, Matplotlib.

3. AI ENGINEERING

This program on modern Artificial Intelligence systems, covers from the foundations of Transformer architectures to the development of advanced AI agents and autonomous systems. Learn how Large Language Models (LLMs) such as GPT, LLaMA, Mistral, and T5 are designed, trained, fine-tuned, and deployed for real-world applications. The curriculum provides hands-on experience in building intelligent applications using Retrieval-Augmented Generation (RAG), vector databases, prompt engineering, and multimodal AI technologies.

Students will develop the ability to create AI-powered chatbots, knowledge assistants, content generation systems, coding assistants, and enterprise AI solutions using industry-standard frameworks such as Hugging Face, LangChain, LlamaIndex, Ollama, ChromaDB, and FAISS. They will also learn to design, implement, and manage autonomous AI agents capable of reasoning, planning, memory management, tool usage, and decision-making.

The program further equips learners with skills in developing multi-agent systems, workflow automation, and no-code/low-code AI applications using platforms such as n8n, CrewAI, AutoGen, LangGraph, OpenAI Agents SDK, and Microsoft Copilot Studio. Through project-based learning, students gain practical expertise in deploying, monitoring, evaluating, and governing AI systems while addressing critical considerations such as ethics, safety, reliability, and scalability.

Key skills you gain

1. **Diffusion Models:** Creating Images from Noise While LLMs generate text, diffusion models create images by starting with random noise and gradually transforming it into a coherent picture.
2. **Transformer Architecture:** Self-attention mechanisms, encoder-decoder design, positional encoding, and multi-head attention for sequence modeling.
3. **Large Language Models (LLMs):** Pre-training, fine-tuning, prompt engineering, and deployment of large-scale language models such as GPT and BERT.
4. **Generative AI:** Principles of text, image, and multimodal content generation using diffusion models, GANs, and autoregressive architectures.
5. **AI Agents:** Design and implementation of goal-driven agents with tool use, memory, and reasoning capabilities in dynamic environments.
6. **Agentic AI & Autonomous Systems:** Multi-agent orchestration, autonomous decision-making pipelines, and real-world deployment of self-directed AI workflows.

Upon completion, students will be capable of building next-generation intelligent applications and autonomous systems, preparing them for roles such as AI Engineer, Generative AI Engineer, LLM Engineer, AI Agent Developer, Machine Learning Engineer, AI Solutions Architect, and AI Product Developer.

CURRICULUM

Diffusion Models

Learn the foundational architecture behind modern generative AI systems, where models create images, video, audio, and other content by gradually transforming random noise into meaningful outputs. Students gain hands-on experience with forward diffusion, reverse denoising, noise scheduling, U-Net architectures, latent representations, text conditioning, and sampling techniques, along with model development, fine-tuning, and performance evaluation using industry-standard deep learning frameworks such as PyTorch and Hugging Face Diffusers

Topics Covered

Foundations & Diffusion Processes:	Fundamentals of Generative AI, limitations of GANs and VAEs, Gaussian Noise, Forward Diffusion Process, Reverse Denoising Process, Markov Chains, Noise Prediction, Timestep Embeddings, and Noise Schedules.
Diffusion Architecture Design:	U-Net Architecture, Encoder–Decoder Structure, Residual Blocks, Skip Connections, Self-Attention, Cross-Attention, Conditional Denoising, and Latent Diffusion Architecture.
Modern Diffusion Models:	DDPM, DDIM, Score-Based Diffusion Models, Stable Diffusion, SDXL, DALL·E, Imagen, and Diffusion Transformer (DiT) Architectures.
Applied Diffusion & Fine-Tuning:	Text-to-Image Generation, Image-to-Image Generation, Inpainting, Outpainting, Super-Resolution, ControlNet, Classifier-Free Guidance, Prompt Engineering, DreamBooth, Textual Inversion, and LoRA Fine-Tuning.
Libraries & Frameworks:	• PyTorch • TensorFlow / Keras • Hugging Face Diffusers • Hugging Face Transformers • PEFT & LoRA • ControlNet / ComfyUI

Transformer Architecture

Learn the foundational architecture behind modern AI systems, where self-attention mechanisms enable models to process and relate information across entire sequences simultaneously. Students gain hands-on experience with encoder-decoder structures, multi-head attention, positional encoding, and feed-forward networks, along with model building and performance tuning using industry-standard deep learning frameworks such as PyTorch and TensorFlow.

Topics Covered

Foundations & Attention Mechanisms:	Limitations of RNNs and LSTMs, Positional Embeddings, Self-Attention Mechanism, Building a single attention head in PyTorch, Multi-Head Attention, Residual Connections & LayerNorm, Stacking into a Full Transformer Block.
Transformer Architecture Design:	Encoder Architecture, Decoder Architecture, Encoder-Decoder Transformers.
Modern Transformer Models:	BERT, GPT, T5, LLaMA, Mistral Architectures.
Applied Transformers & Fine-Tuning:	Hugging Face Transformers, Fine-Tuning Transformers, Transformer-based NLP and Vision Models.
Libraries & Frameworks:	• PyTorch • TensorFlow / Keras • Hugging Face Transformers • PEFT & LoRA • LangChain / LlamaIndex

Large Language Models (LLMs)

Learn the principles and inner workings of Large Language Models, where billions of parameters are trained on massive text corpora to understand, generate, and reason across diverse language tasks. Students gain hands-on experience with pre-training strategies, prompt engineering, retrieval-augmented generation, and model fine-tuning, along with building and evaluating LLM-powered applications using industry-standard platforms such as Hugging Face, OpenAI API, and LangChain.

Topics Covered

LLM Fundamentals:	LLM Architecture, Pre-Training, Fine-Tuning.
Knowledge Augmentation & Reliability:	Retrieval-Augmented Generation (RAG), Vector Databases, Hallucination Handling.
Open-Source LLM Deployment:	Open-Source LLM Deployment, Ollama, LangChain, LlamaIndex.
LLM-Powered Application Development:	Chatbots, PDF Knowledge Assistant, Institutional AI Assistants, ChromaDB, FAISS
Libraries & Frameworks:	• HuggingFace Transformers • LangChain • Ollama • LlamaIndex • ChromaDB / FAISS

Training a Tiny LLM

Learn the end-to-end process of training a compact Large Language Model from scratch, where raw text data is transformed into a fully functional language model through tokenization, embedding, and iterative gradient-based optimization. Students gain hands-on experience with dataset preparation, vocabulary building, loss computation, and training loop design, along with building and evaluating a fully operational tiny LLM using industry-standard deep learning frameworks such as PyTorch and Hugging Face Transformers.

Topics Covered

Tokenization & Data Preparation:	Byte-Level Tokenization, Dataset Batching & Shifting for Next-Token Prediction.
Loss Computation & Training:	Cross-Entropy Loss & Label Shifting, Training Loop from Scratch (No Trainer API).
Text Sampling & Generation:	Sampling: Temperature, Top-k, Top-p.
Model Evaluation & Validation:	Evaluating Loss on Validation Set
Libraries & Frameworks:	• PyTorch • HuggingFace Transformers and Datasets • Tokenizers • NumPy • MatplotLib

Generative AI

Learn the core principles of Generative AI, where models are trained to create original content across text, image, audio, and video modalities by learning the underlying patterns and distributions of large-scale datasets. Students gain hands-on experience with diffusion models, GANs, VAEs, and multimodal generation pipelines, along with building and deploying creative AI applications using industry-standard tools such as Hugging Face Diffusers, Stable Diffusion, and OpenAI APIs.

Topics Covered

Generative AI Foundations & Architectures:	Generative AI Architectures, GPT Models, Tokenization, Embeddings.
Prompt Engineering & Multimodal AI:	Prompt Engineering, Multimodal AI Systems, AI Image Generation.
Generative AI Models & Platforms:	ChatGPT, Claude, Gemini, Hugging Face, Perplexity.
Applied Generative AI Development:	AI-Assisted Coding, Content Generation, Intelligent AI Assistants
Libraries, Frameworks, Platforms & API:	• HuggingFace Diffusers • Tokenizers • LangChain • Gradio • Streamlit • OpenAI API • Goolge Gemini API • Stable Diffusion / DALL.E

AI Agents

Learn the design and development of AI Agents, where intelligent systems are built to perceive their environment, reason over goals, and autonomously execute multi-step tasks using tools, memory, and planning capabilities. Students gain hands-on experience with agent architectures, tool integration, memory management, and ReAct-based reasoning frameworks, along with building and deploying goal-driven AI agents using industry-standard platforms such as LangChain, LangGraph, and AutoGen.

Topics Covered

Foundations of Intelligent Agents:	Introduction to Intelligent Agents, Agent Architectures, Environments, Perception, Reasoning, Planning, Decision-Making.
LLM-Powered Agent Development:	Autonomous and Multi-Agent Systems, LLM Integration, Tool Integration, Memory Management, Retrieval-Augmented Generation (RAG).
Agent Design & Real-World Applications:	Conversational Agents, Task-Oriented Agents, Autonomous Agents, Real-World Application Design and Implementation.
Evaluation, Ethics & Deployment:	Evaluation, Deployment, Monitoring, Ethics, Safety, Governance of AI Agent Systems
Libraries, Frameworks, Platforms & API:	• LangChain • LangGraph • AutoGen • Crew AI • Hugging Face Transformers

Agentic AI & Autonomous Systems

Learn the advanced principles of Agentic AI and Autonomous Systems, where AI models are designed to operate independently, coordinate across multi-agent pipelines, and execute complex real-world workflows with minimal human intervention. Students gain hands-on experience with agent orchestration, autonomous decision-making, self-reflective reasoning, and human-in-the-loop design, along with building and deploying end-to-end agentic systems using industry-standard platforms such as LangGraph, CrewAI, and AutoGen.

Topics Covered

Autonomous Agent Foundations:	Autonomous AI Agents, Planning and Reasoning, Memory Systems, Human-in-the-Loop Systems.
Multi-Agent Collaboration & Workflow Automation:	Multi-Agent Collaboration, Workflow Automation, No-Code/Low-Code AI Agent Development using n8n.
Agentic AI Frameworks & Platforms:	CrewAI, AutoGen, LangGraph, OpenAI Agents SDK, Microsoft Copilot Studio.

Enterprise Agentic Applications:	AI Scheduling Assistants, Research Agents, Enterprise Automation Systems
Libraries, Frameworks, Platforms & API:	• LangChain • LangGraph • AutoGen • Crew AI • Open AI Agents SDK • n8n • Microsoft Copilot Studio

4. AI IN SOFTWARE ENGINEERING

Artificial Intelligence (AI) in Software Engineering refers to the application of intelligent technologies and machine learning techniques to automate, optimize, and improve various activities involved in software development. AI is transforming the traditional software engineering process by assisting developers in coding, testing, debugging, maintenance, project management and decision-making.

Modern AI-powered tools such as GitHub Copilot, ChatGPT, Gemini, Tabnine, Codium, and Code Whisperer help software engineers develop high-quality applications faster and more efficiently. AI can analyze large amounts of code, predict bugs, generate programs, automate testing, and provide intelligent coding suggestions.

AI is now becoming an essential part of the Software Development Life Cycle (SDLC), supporting activities from requirement analysis to software deployment and maintenance. Organizations worldwide are adopting AI-driven software engineering practices to improve productivity, reduce development cost, and accelerate innovation.

Key skills you gain

1. **AI-Assisted Requirement Analysis**
   Ability to use AI tools for gathering, analyzing, validating, and documenting software requirements, user stories, and functional specifications.
2. **Intelligent Software Design**
   Skills in designing software architectures, UML models, workflows, and system components with AI-assisted design tools.
3. **AI-Enhanced Programming**
   Ability to develop software using AI-powered coding assistants, code generation tools, and intelligent development environments.

4. **Automated Software Testing**
   Skills in generating test cases, performing automated testing, detecting defects, and improving software reliability using AI-based testing platforms.
5. **Software Documentation Generation**
   Skills in automatically creating technical documentation, API documentation, user manuals, and project reports using AI tools.
6. **AI-Based Project Management**
   Ability to utilize AI for project planning, effort estimation, sprint management, task prioritization, and resource allocation.

CURRICULUM

Foundations of AI in Software Engineering

Develop a strong foundation in applying Artificial Intelligence (AI) concepts, techniques, and tools to modern software engineering practices. Students gain an understanding of how AI enhances different phases of the Software Development Life Cycle (SDLC), enabling intelligent requirement analysis, software design, coding, testing, maintenance, and project management. This module provides the essential knowledge required to build, evaluate, and utilize AI-powered software engineering solutions.

Topics Covered and Tools Used:

AI in Software Engineering:	AI in SDLC, AI-Assisted Requirement Engineering, AI-Based Software Design, AI-Powered Coding Assistants, AI in Software Testing, AI for Software Maintenance, Intelligent Project Management
Responsible and Ethical AI:	AI Ethics and Fairness, Bias in AI Systems, Privacy and Security Issues, Responsible AI Practices,
AI Ecosystem and Development Platforms:	AI Development Frameworks, Open-Source AI Tools, Cloud-Based AI Platforms, Model Deployment Concepts, AI Integration in Software Projects, Future Trends in AI-Driven Software Engineering
Introduction to Artificial Intelligence:	Definition and History of AI, Evolution of AI, AI vs Traditional Programming, Types of AI (Narrow AI, General AI, Generative AI), Applications of AI in Various Domains
AI Tools:	**ChatGPT, Gemini, Claude, Perplexity AI, GitHub Copilot, Cursor AI**

AI in Requirements Engineering and Software Design

Develop expertise in applying Artificial Intelligence techniques and tools to software requirements engineering and software design activities. Students learn how AI can assist in eliciting, analyzing, validating, prioritizing, documenting, and managing software requirements while supporting architectural design, UML modeling, system decomposition, design pattern selection, and software documentation. The module focuses on improving software quality, reducing development time, and enhancing decision-making through AI-assisted engineering practices.

Topics Covered and Tools Used

Introduction to AI in Requirements Engineering:	Role of AI in Requirements Engineering, AI-Assisted Requirement Gathering
AI-Based Requirement Analysis and Specification:	Requirement Classification, Functional and Non-Functional Requirement Identification, Requirement Validation, Ambiguity Detection, Requirement Documentation using AI
User Stories and Requirement Management:	AI-Generated User Stories, Requirement Prioritization, Traceability Management, Change Impact Analysis
AI in Software Design and Architecture:	AI-Assisted Software Architecture Design, Component Identification, Architectural Pattern Recommendation
AI-Assisted UML and System Modeling:	Use Case Diagram Generation, Class Diagram Generation, Sequence Diagram Generation, Activity Diagram Generation, ER Diagram Generation using AI
AI for Design Optimization and Documentation:	Design Pattern Selection, Refactoring Recommendations, Software Documentation Generation, Architecture Review, Design Quality Assessment
AI Tools:	**Claude, ClickUp AI, Notion AI, Monday AI, Lucidchart AI, Eraser AI, Visual Paradigm AI, Mermaid AI, StarUML AI, Cursor AI, SonarQube AI**

AI in Code Generation and Software Construction

Develop expertise in utilizing Artificial Intelligence tools and techniques for software construction, code generation, debugging, refactoring, code optimization, and software development automation. Students learn how AI-powered coding assistants improve productivity, software quality, maintainability, and development speed through intelligent code suggestions, automated documentation, code review, and collaborative AI pair programming practices.

The **18 AI-powered coding tasks** (shown below) enable developers to write, understand, analyze, optimize, secure, maintain, and evolve software more efficiently, making AI an essential component of modern software engineering

S.No	Programming Task	Description	Open-Source AI Tools	AI Tools (Commercial / Proprietary)	Behind the Tool – AI Technology
1	**Automatic Code Edit**	Auto-modification of code.	Continue.dev, OpenHands	GitHub Copilot, Cursor	Large Language Models (LLMs), Transformer Networks, Reinforcement Learning
2	**Code Analysis**	Evaluates structural and runtime behavior of source code.	SonarQube Community Edition, Semgrep	Snyk, DeepCode (Snyk Code)	Static Analysis, Machine Learning, Graph Neural Networks (GNNs)
3	**Code Authorship and Identification**	Utilizes neural models to attribute code to developers.	CodeBERT, PyDriller	IBM watsonx	Deep Learning, Stylometry, Transformer-based Models
4	**Code Change Detection**	Tracks commit updates in large-scale software development.	PyDriller, OpenRewrite	GitHub Advanced Security, GitLab Duo AI	NLP for Code, Sequence Models, Change Mining Algorithms
5	**Code Classification**	Categorizes code based on syntax, semantics, and algorithms.	CodeT5, GraphCodeBERT	OpenAI	Transformers, Semantic Embeddings, Deep Neural Networks
6	**Code Clone Detection**	Identifies duplicate or near-duplicate code snippets to maintain software quality.	PMD CPD, SourcererCC	Black Duck	Token-Based Matching, AST Analysis, Siamese Neural Networks
7	**Code Completion**	Enhances productivity by auto-completing blocks of codes.	Codeium, Tabby	Tabnine, Amazon Q Developer	Generative AI, Transformers, Predictive Language Modeling
8	**Code Generation**	Automates code creation using code-assistants.	Open Interpreter, Continue.dev	ChatGPT, Claude AI	Generative AI, LLMs, Transformer Architecture
9	**Code Modeling and Representation**	Uses various representations for better source code understanding.	GraphCodeBERT, Code2Vec	Amazon CodeGuru Reviewer	Embedding Models, Graph Neural Networks, Representation Learning
10	**Code Search and Retrieval**	Enables retrieval of relevant code snippets.	OpenGrok, Sourcegraph Cody Open Source	Sourcegraph Cody Enterprise	Semantic Search, Vector Databases, NLP
11	**Code Similarity Detection**	Identifies duplicate code using neural networks.	NiCad, Deckard	Moss	Deep Learning, AST Matching, Similarity Learning

S.No	Programming Task	Description	Open-Source AI Tools	AI Tools (Commercial / Proprietary)	Behind the Tool – AI Technology
12	**Code Summarization**	Generates human-readable descriptions for code.	CodeT5, PolyCoder	OpenAI Codex	NLP, Sequence-to-Sequence Learning, Transformers
13	**Code Vulnerability Detection**	Uses various neural methods to detect security flaws.	Semgrep, OWASP Dependency-Check	Checkmarx, Veracode	AI-based Security Analysis, Deep Learning, Pattern Recognition
14	**Comment Generation**	Generate descriptive comments for code.	CodeGeeX, Documatic	Mintlify Doc Writer, CodiumAI	Natural Language Generation (NLG), Transformers
15	**De compilation**	Converts machine-level code into high-level source code.	Ghidra, RetDec	IDA Pro	Reverse Engineering, Pattern Recognition, ML-assisted Binary Analysis
16	**Program Repair and Bug Fix**	Improve program debugging and efficiency.	Repairnator, SWE-agent	Sweep AI	Automated Program Repair (APR), Reinforcement Learning, LLMs
17	**Program Synthesis**	Converts NL instructions into code using neural models.	MetaGPT, Smol Developer	OpenAI Codex, Replit Ghostwriter	Neural Program Synthesis, Transformers, Seq2Seq Models
18	**Source Code Translation**	Migrates code across different versions of the same language or translates it between different languages.	TransCoder, CodeGeeX	Google AI Studio	Machine Translation, Transformer Models, Cross-Language Representation Learning

Topics Covered (Grouping based on categories):

Introduction to AI-Assisted Software Development:	Evolution of AI in Programming, AI Pair Programming, Human-AI Collaboration Models, AI-Assisted Development Workflow
AI-Assisted Code Development and Generation:	Automatic Code Edit (1), Code Completion (2), Code Generation (3), Program Synthesis (4), Comment Generation (5), Source Code Translation (6)
Code Understanding and Knowledge Extraction:	Code Analysis (7), Code Classification (8), Code Modeling & Representation (9), Code Search & Retrieval (10), Code Summarization (11), Code Authorship & Identification (12)

Software Quality Assurance and Security:	Code Clone Detection (13), Code Similarity Detection (14), Code Vulnerability Detection (15), Program Repair & Bug Fix (16)
Software Evolution and Maintenance:	Code Change Detection (17), De-compilation (18).

1. Automatic Code Edit

Automatically modifies existing source code based on developer instructions, coding standards, or bug-fix requirements. AI tools can update functions, refactor logic, and apply requested changes without manually rewriting the code. AI automatically modifies existing code based on developer instructions, reducing manual effort and improving productivity.

AI Involvement: Large Language Models (LLMs) understand developer intent and generate precise code modifications while preserving program functionality**.**

2. Code Completion

Predicts and suggests the next lines of code while a developer is programming. It improves productivity by providing context-aware code snippets, function calls, and variable suggestions. AI predicts and suggests the next lines of code while programmers write software.

AI Involvement: AI models analyze the coding context in real time and predict the most relevant code completions based on learned programming patterns**.**

3. Code Generation

Generates complete code segments, functions, classes, or applications from natural language descriptions or requirements. AI models translate user intent into executable source code. AI generates functions, classes, modules, or complete programs from natural language prompts.

AI Involvement: Generative AI converts textual prompts into syntactically correct and contextually relevant source code.

4. Program Synthesis

Automatically creates a complete program from high-level specifications, examples, or constraints. It focuses on generating correct and optimized solutions without explicit programming. AI creates executable programs automatically from specifications, examples, or requirements.

AI Involvement: AI learns mappings between problem specifications and program structures to synthesize executable software automatically.

5. Comment Generation

Produces meaningful comments and documentation for source code automatically. It helps improve code readability, maintainability, and knowledge sharing among development teams. AI generates meaningful comments and documentation to improve code readability and maintenance.

AI Involvement: Natural Language Processing (NLP) models analyze source code semantics and generate human-readable explanations.

6. Source Code Translation

Converts source code from one programming language to another while preserving functionality. Examples include translating Java code into Python or C++ into JavaScript. AI converts source code from one programming language to another while preserving functionality.

AI Involvement: AI models understand language syntax and semantics to generate equivalent implementations across programming languages.

7. Code Analysis

Examines source code to identify errors, vulnerabilities, performance issues, and coding standard violations. AI-based analysis helps developers improve software quality and reliability. AI examines code to identify errors, inefficiencies, vulnerabilities, and quality issues.

AI Involvement: Machine learning algorithms detect patterns associated with bugs, inefficiencies, and security risks.

8. Code Classification

Categorizes source code into predefined groups such as algorithms, libraries, modules, design patterns, or application domains. It supports software organization and repository management. AI categorizes source code into predefined groups such as algorithms, modules, or application domains.

AI Involvement: AI automatically learns code features and assigns categories based on functionality and structure.

9. Code Modeling & Representation

Transforms source code into machine-understandable representations such as Abstract Syntax Trees (ASTs), graphs, embeddings, or semantic models. These representations enable advanced AI-based code understanding. AI transforms code into machine-understandable representations for analysis and learning.

AI Involvement: Deep learning techniques generate meaningful code embeddings that capture syntax, semantics, and structural relationships.

10. Code Search & Retrieval

Retrieves relevant code snippets, APIs, libraries, or functions from large repositories based on keywords or natural language queries. It accelerates software reuse and development efficiency. AI helps developers find relevant code snippets, libraries, and APIs using natural language queries.

AI Involvement: AI-powered semantic search understands developer intent and finds relevant code beyond simple keyword matching.

11. Code Summarization

Generates concise descriptions of source code functionality, behavior, and purpose. It helps developers quickly understand unfamiliar or legacy codebases. AI generates concise descriptions explaining the functionality and purpose of source code.

AI Involvement: NLP-based AI models translate complex code structures into understandable natural language summaries.

12. Code Authorship & Identification

Determines the likely author, coding style, or ownership of source code using programming patterns and stylistic features. It is useful for software forensics and intellectual property analysis. AI identifies coding styles and predicts the likely author or ownership of source code.

AI Involvement: Machine learning models identify unique coding fingerprints and stylistic characteristics of developers.

13. Code Clone Detection

Identifies duplicated or near-duplicated code fragments within a software system. Detecting clones helps reduce redundancy, improve maintainability, and minimize technical debt. AI detects duplicated or highly similar code fragments within software systems

AI Involvement: AI detects both exact and semantic code similarities even when variable names and structures differ.

14. Code Similarity Detection

Measures the semantic or syntactic similarity between different code segments. It is used in plagiarism detection, software reuse, and repository analysis. AI measures functional and structural similarity between different code segments.

AI Involvement: Deep learning models compare code representations to identify functional similarities beyond textual matching.

15. Code Vulnerability Detection

Automatically detects security flaws and weaknesses in source code, such as SQL injection, buffer overflows, or authentication issues. AI assists in improving software security and compliance. AI identifies security weaknesses and vulnerabilities in software source code.

AI Involvement: AI models trained on security datasets recognize vulnerability patterns and recommend remediation strategies.

16. Program Repair & Bug Fix

Identifies software defects and automatically generates corrective code changes. AI-based repair systems can recommend or apply fixes to improve software reliability. AI automatically detects defects and suggests or generates corrective code changes.

AI Involvement: Generative AI analyzes bug reports, execution traces, and source code to propose effective fixes.

17. Code Change Detection

Detects modifications between different versions of source code and analyzes their impact on software behavior. It supports version control, maintenance, and software evolution. AI analyzes code modifications across versions and assesses their impact on software behavior.

AI Involvement: AI evaluates code differences and predicts the potential impact of changes on system functionality and quality.

18. De-compilation

Converts executable binaries or bytecode back into a human-readable source-code-like representation. It assists in reverse engineering, security analysis, and legacy software maintenance. AI assists in converting executable binaries into human-readable source-code-like representations.

AI Involvement: AI improves the accuracy of reverse engineering by reconstructing higher-level program structures and logic from compiled code.

AI in Testing and Deployment

Develop expertise in applying Artificial Intelligence techniques and tools to software testing, quality assurance, continuous integration, deployment automation, monitoring and DevOps practices. Students learn how AI improves software reliability, accelerates testing cycles, automates defect detection, enhances release management, and enables intelligent deployment pipelines for modern software systems.

Topics Covered:

Introduction to AI in Software Testing:	Role of AI in Quality Assurance, Evolution of Intelligent Testing, AI-Driven Testing Frameworks, Test Automation Fundamentals
AI-Based Test Design and Generation:	Automatic Test Case Generation, Requirement-Based Test Generation, Unit Test Creation, Integration Test Generation, Test Data Generation
AI-Assisted Test Execution and Quality Assurance:	Automated Test Execution, Intelligent Regression Testing, Defect Prediction, Bug Detection
AI in Deployment:	Future Trends in AI-Driven Deployment
AI Tools:	**Testim AI, Mabl, TestSigma AI, SonarQube AI, Testregior**

AI in Software Maintenance, Project Management, and Future Trends

Develop expertise in applying Artificial Intelligence technologies to software maintenance, project management, software evolution, resource planning, risk management, and next-generation intelligent software engineering practices.

Topics Covered:

AI in Software Maintenance:	Software Evolution, AI-Assisted Maintenance, Bug Localization, Impact Analysis
AI in Project Management:	Project Planning, Resource Allocation, Effort Estimation, Sprint Planning, Task Prioritization, Team Productivity Analytics
AI-Assisted Risk and Quality Management:	Risk Prediction, Defect Prediction, Quality Assessment, Project Monitoring, Decision Support Systems, Predictive Analytics
AI Tools:	GitHub Copilot, Cursor AI, Microsoft Copilot, Notion AI, Jira AI, ClickUp AI, Monday AI

5. AI PAIR PROGRAMMING

The software industry has recognized the potential impact of code generation tools, making it a significant topic in today's world. These tools are capable of improving the efficiency of software development, saving time and costs. Pair programming is another widely adopted technique in software engineering, popular for its ability to create high-quality software within a short time frame.

Pair Programming is a software engineering process that involves two programmers collaborating at the same workstation while using this agile software development technique. This process is a component of the extreme programming methodology, which is gaining popularity in businesses. The central concept of pair programming is that two individuals working together on the same problem can derive process improvements that result in better software.

In pair programming, one person assumes the role of the observer or navigator, who carefully observes the driver's (the second partner) work and offers advice while the other person, the driver, actively types on a computer or documents an architecture or design. The two programmers routinely switch between these two duties.

While reviewing, the observer also takes into account the direction of the work, generating suggestions for enhancements and potential issues to handle in the future. The driver can then focus only on completing the current task, using the observer as a safety net and guide.

It is found that the largest perceived benefits of pair programming were overall fewer bugs in the code, spreading code understanding among the team, and producing higher quality code. However, there may be situations, such as illness, scheduling conflicts, or efficiency concerns, that require the two individuals to work independently. Experienced pair programmers prioritize the stages of the development cycle, deciding which ones are most crucial to work on together and which ones can be completed separately. When they reunite, they must determine how to incorporate the individually created work. As a bigger group starts using pair programming as the standard method of working, the long-term continuity of a specific pair becomes less important. An individual programmer can maintain sufficient general awareness to fill in for an absent partner at a moment's notice by partnering regularly with every member of the group.

Language models

Language models is a type of deep learning model that provides probability distributions of a collection of terms in a language. While earlier language models were based on non-neural techniques, there has been a trend toward using neural networks in recent years. These models are commonly used in natural language processing to generate text. Recurrent neural networks (RNNs) have been the primary building block of language models in this shift. They take input sequences and produce a corresponding output probability distribution for the subsequent tokens in a sequence. The traditional LSTM architectures outperform more modern models when properly regularized, which was a rather unexpected result.

RNN-based language models are useful for a wide range of tasks, such as part-of-speech labeling, question-answering, and machine translation. While RNNs generally offer better performance than other models, they also suffer from a short-term memory problem, which can impact their performance, particularly when dealing with lengthy input sequences.

Code Generation Tools

The field of software development has now advanced to the point where machine learning tools and techniques can be employed in the coding process using deep learning and natural language processing. As a result, developers can now make extensive use of code completion and generation technologies offered by integrated development environments (IDEs) or as add-ons to text editors.

As NLP/DL technology develops, these code completion tools get more complex. An advanced code completion tool is Copilot from GitHub. An "AI pair programmer trained on billions of lines of public code," according to Copilot's description. Copilot, a text editor add-on for the VSCode text editor, creates potential code completions for developers by taking into account the program's context.

Code generation tools using machine learning, belongs to one of three groups.

- The first category is Description-to-code, these descriptions are often obtained from code comments written before a code snippet.
- Second is Code-to-description, this task's goal is to produce a description of the code, typically in the form of a comment. It is sometimes referred to as source code summarizing.
- The third category is Code-to-code, this paradigm's most widely used application category is automatic software repair (APR). APR receives flawed or buggy code as input, and the model try to produce identical code without the bug.

The majority of code generation studies fall in the first category with 46 %, the second category had 25 % selected studies.

Codex

Codex is a GPT language model fine-tuned on publicly available code from GitHub, specifically developed to produce code. It is claimed to produce best results for Python code-writing capabilities. A distinct production version of Codex powers GitHub Copilot. It may generate code fragments that are typically both syntactically and semantically sound when given a brief user description.

OpenAI developed Codex using a unique approach, based on their discovery the repeatedly sampling from the model can be an effective tactic for generating useful answers to challenging problems. By using this approach and sampling multiple times per problem, they were able to solve more problems. This method can increase the chances of finding a suitable solution. By utilizing Codex's ability to generate code, developers can benefit from this approach and increase their chances of finding efficient and effective solutions to their coding challenges.

Copilot

Copilot is an AI-powered tool created by Microsoft that functions as an "AI pair programmer." It is capable of generating code in multiple programming languages based on the contex provided as a prompt, including nearby code, comments, and method names.

Copilot offers three primary functionalities:

- auto-fill for repetitive code,
- suggestions for tests matching the implementation code, and
- conversion of comments into code.

The tool is using in its core Codex that have been specifically trained and fine-tuned for the purpose of generating code.

GitHub suggests that not all the code used was tested for bugs, insecure practices, or personal data, despite the glowing reviews of Copilot's productivity improvements on the website. The company writes they have put a few filters in place to prevent Copilot from generating offensive language, but it might not be perfect. The company also cautions that although this is uncommon and that the data has been discovered to be synthetic or pseudo-randomly generated by the algorithm, the model could suggest email addresses, API keys, or phone numbers. However, the majority of the Copilot-generated code is original. Only a small fraction of the generated code could be replicated exactly in the training set.

The foundation of Copilot is OpenAI's Codex**,** this is a refined version of GPT-3. This package can be used within visual studio code with different settings and was promoted as being a tool that the programmer can pair with.

AlphaCode

DeepMind Alpha Code is an innovative AI system developed by Google-owned DeepMind**.** The system relies on a powerful combination of deep learning and reinforcement learning techniques to achieve its model. It's powered by a neural network with millions of parameters that have been fine-tuned through training on simulated games. By using deep learning, AlphaCode is able to analyze and understand complex patterns in data, allowing it to make accurate predictions and decisions. And by using reinforcement learning, it is able to learn from its mistakes and continually improve its performance over time.

The potential applications of AlphaCode are wide-ranging and include different fields from robotics and self-driving cars to medical diagnosis and scientific research.

CodeWhisperer

Amazon CodeWhisperer is a new AI-powered tool developed by Amazon Web Services (AWS) that aims to assist developers in the software development process. The tool utilizes natural language processing (NLP) and machine learning techniques to provide

suggestions and auto-complete code segments based on the developer's programming language. The system is designed to learn from developers' usage patterns and improve its recommendations over time. According to AWS, CodeWhisperer has shown promising results in reducing the time and effort required to develop high-quality code.

CodeWhisperer can extract patterns and structures from the code to make suggestions to developers, including auto-completion of code segments, highlighting code errors, and providing recommendations for code improvement. The system's machine learning algorithms also enables it to learn from developers' usage patterns, making it more effective over time.

One of the features of CodeWhisperer is its ability to suggest code for a developer based on their intent. The system uses natural language processing (NLP) techniques to analyze the developer's code and provide suggestions that are aligned with their intended goals. This feature can improve the productivity of developers by reducing the time required to write code manually. CodeWhisperer can be integrated with other AWS services, such as Amazon SageMaker, to provide a complete AI-assisted development experience.

Automations with No-Code Tools

- Learn how to let software do the repetitive work without writing a single line of code.
- Usage of tools like n8n, Zapier and Hostinger Horizons to connect the apps you already use (like forms and spreadsheets) so information moves automatically.
- Create small workflows that save time, like sending summaries, tagging files, or routing messages.
- Also experiment with building basic websites or internal tools using no-code builders, and learn how to test, adjust, and improve your automations.
- Design AI-enhanced flows that can summarize, tag, or route information automatically
- Build simple websites or web apps using no-code builders (Hostinger Horizon, Firebase, Lovable.ai)
- Prototype, test, and iterate on automated workflows

The core tools would be:

- n8n
- Zapier
- Spreadsheets with AI
- Python with AI
- Firebase (for advanced workflows)
- Lovable.ai

n8n: WORKFLOW AUTOMATION PLATFORM

n8n is a workflow automation and orchestration platform that allows users to connect applications, APIs, databases, and AI models through visual workflows with little or no coding.

How n8n works with LLMs

n8n does not contain its own LLM. Instead, it can integrate with:

- OpenAI GPT models
- Llama models
- Gemini
- Claude
- Hugging Face models

For example:

User Query → n8n Workflow → GPT/Llama → Database → Email/Slack Response

The n8n can be classified under Workflow Automation & Agentic AI Platforms

- n8n
- Zapier
- Make.com

No-Code/Low-Code AI Development Tools

- n8n
- Microsoft Copilot Studio
- Zapier
- Retool

6. DEEP-TECH AI TRAJECTORIES

Agentic architecture is broader than simply using an LLM.

A practical Agentic AI system typically combines:

LLM + Reasoning + Planning + Tools + Memory + Observation + Action + Feedback

This makes **ReAct, Plan-and-Execute, Reflection/Reflexion, Memory-Augmented, Hierarchical, Multi-Agent, Supervisor–Worker, and Human-in-the-Loop architectures** particularly important topics for an AI Engineering course.

Deep-Tech topics with architectures in the sequence **Tool-Calling → ReAct → Planning → Reflection → Memory → RAG + Agents → Multi-Agent Systems → Autonomous Agents**, followed by hands-on projects.

From 2020 onward, the AI landscape has evolved rapidly from task-specific machine learning and generative models toward **autonomous, reasoning-oriented AI systems capable of perceiving context, planning multi-step actions, using external tools, maintaining memory, collaborating with other agents, and operating within real-world environments**. The emergence and rapid advancement of foundation models and LLMs has accelerated this transition toward Agentic AI. Deep knowledge in the following technology dimensions is particularly relevant to the development of modern Agentic AI systems:

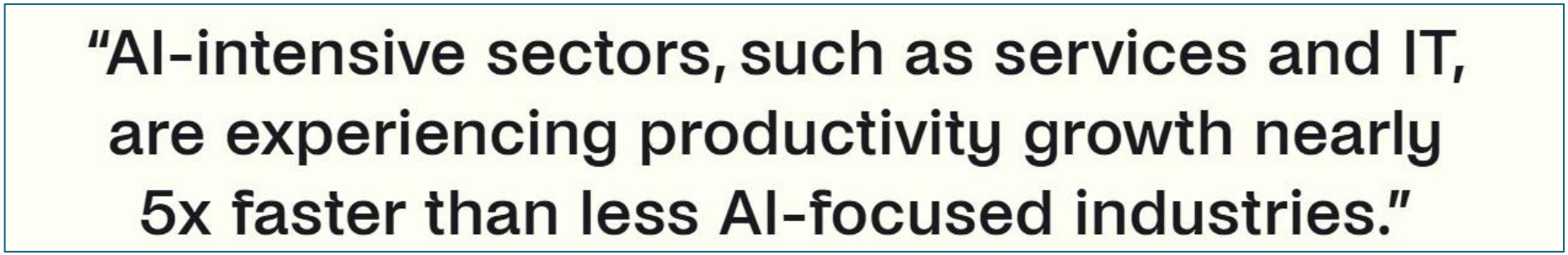
1. Mamba and Jamba Architectures
2. Multi-step reasoning
3. Memory mechanisms
4. Tooluse
5. Multi-agent collaboration
6. Infrastructure and orchestration
7. Safety and security
8. BDI frameworks
9. Agentic RAG
10. Chain-of-Agents
11. Vision-language-action models
12. **Benchmarks:** BIRD (Text-to-SQL), SWE-bench (real-world software engineering/bug-fixing), and Terminal-Bench (DevOps/live shell system administration)
13. **Evaluation methodologies:** Deterministic programmatic checks, LLM-as-a-judge trajectory analysis, and Human-in-the-loop validation

INDUSTRY REQUIREMENTS

The Table 2 presents a consolidated view of the key competencies and skill priorities expected by leading AI-focused recruiters, including Qualcomm, Eightfold.ai, Cognizant, Virtusa, and ABB. While each organization emphasizes specific technical and domain requirements aligned with its business needs, a common theme emerges: the demand for graduates who can effectively develop, deploy, and apply AI-driven solutions while demonstrating strong engineering, problem-solving, and industry-readiness capabilities.

Table 2. Table Industry Readiness Priorities Across Leading AI-Focused recruiters

Industry	Priority 1	Priority 2	Priority 3	Priority 4	Priority 5
Qualcomm	AI Deployment & Orchestration	Systems & Software Engineering	Product Development Skills	Problem Solving & Collaboration	Industry-Ready AI Engineering
Eightfold.ai	Conversational AI & Agent Design	Agentic AI & Reasoning	AI Application Development	User Experience & Adaptability	AI Product Mindset
Cognizant	AI-Native Application Development	Full-Stack & Cloud Engineering	AI Deployment & Governance	Solution Architecture & Problem Solving	Industry & Client Readiness
Virtusa	Mathematical & AI Foundations	AI Model Development	Programming & Software Engineering	Data Analytics & Visualization	Business Application Readiness
ABB	AI Agents & Intelligent Systems	Data Engineering & Analytics	Cloud & Platform Engineering	AI-Driven Problem Solving	Innovation & Industry Application



KEY FINDINGS

The Artificial Intelligence Model Deployment Program reflects the institution's commitment to preparing students for the rapidly evolving AI-driven technology landscape. The program is built around five key competency areas, integrating Generative AI into Fundamental Programming, Machine Learning Algorithms, AI Engineering, AI in Software Engineering, and AI Pair Programming which collectively provide a strong foundation for developing industry-relevant knowledge and practical skills. By combining theoretical concepts with hands-on learning experiences, the program enables students to design, develop, deploy, and manage intelligent systems capable of solving real-world problems.

The summary of Key Findings are as follows:

1. **AI is Reshaping Workforce Skill Requirements**
   The rapid advancement of AI technologies is transforming job roles and redefining the skills required across industries. Traditional task-based skills are increasingly being automated, creating demand for new competencies.
2. **Growing Demand for AI and Digital Competencies**
   Organizations require professionals with expertise in AI, Machine Learning, Data Science, AI Engineering, and AI-enabled Software Development. Foundational digital literacy and data-driven decision-making skills are becoming essential across all disciplines.
3. **Human-Centric Skills Remain Critical**
   Despite increased automation, skills such as critical thinking, creativity, problem-solving, communication, collaboration, adaptability, and ethical reasoning continue to be highly valued as complementary capabilities to AI technologies.
4. **AI Readiness Requires Institution-Wide Skill Development**
   AI preparedness extends beyond technical specialists. Faculty, students, researchers, administrators, and future managers require AI awareness and competency development to effectively participate in AI-driven environments.
5. **Continuous Upskilling is Essential**
   AI adoption necessitates ongoing learning and reskilling initiatives. Workforce readiness depends on providing structured training programs that enable individuals to adapt to evolving technologies and workplace demands.
6. **Skill Gaps Remain a Major Challenge**
   The shortage of AI-related skills continues to be a significant barrier to successful AI adoption. Organizations and educational institutions must proactively bridge these gaps through targeted curriculum enhancements and training interventions.
7. **Integration of AI into Education is Increasingly Important**

AI-powered learning platforms can personalize education, improve learner engagement, identify skill gaps, and support competency-based learning, enhancing overall educational effectiveness.

8. **AI Readiness Requires Ethical and Responsible Adoption**
   Successful AI implementation must address challenges related to data privacy, transparency, bias, security, and responsible AI governance to ensure sustainable and trustworthy deployment.
9. **Cross-Disciplinary AI Literacy is Becoming Essential**
   AI is no longer confined to computer science disciplines. Professionals across engineering, management, healthcare, finance, and other sectors require a foundational understanding of AI concepts and applications.
10. **Strategic Investment in AI Capability Development is Necessary**
    Institutions that invest in AI-focused curriculum, faculty development, industry collaborations, experiential learning, and AI infrastructure will be better positioned to produce industry-ready graduates and support future workforce needs.

IARE possesses the necessary academic environment, faculty expertise, infrastructure, and innovation ecosystem to successfully implement this program. The curriculum aligns with current industry requirements and emerging technological trends, fostering competencies in AI-assisted software development, intelligent automation, and model deployment. Through this initiative, students will be better equipped for careers in Artificial Intelligence, Machine Learning, Software Engineering, and related domains, while also contributing to research, innovation, and entrepreneurship.

Overall, the institution demonstrates a high level of readiness to establish itself as a center of excellence in AI education, application development, and intelligent system deployment.