

MODULE 1 · KNOWLEDGE PACK

Introduction to Artificial Intelligence

The vocabulary, mental models and ethics every later course builds on

IBM Generative AI Engineering · ML360 study pack

By Prabhat Kumar · Chief Learning Officer, ML360 Educations Pvt. Ltd.

Learning Objectives

- Define **AI, machine learning, deep learning, generative AI and LLMs**, and show how they nest
- Tell **narrow AI** apart from general and super AI, and explain why every product today is narrow
- Explain **supervised, unsupervised, reinforcement and self-supervised** learning with examples
- Describe the **training loop** and the difference between **training and inference**
- Apply IBM's **five pillars of trustworthy AI** to a real decision system
- **Build** a rule-based vs learned spam filter and a perceptron from scratch



Why This Module Matters

16
Coursera courses in the IBM GenAI Engineering certificate

C1
is the foundation course this module covers

~13 h
of Coursera video, readings and quizzes in Course 1

- Every later course assumes this vocabulary: **model, training, inference, label, bias, hallucination**
- Course 1 quizzes test **precise definitions**, so loose everyday meanings cost marks
- The two mini-projects give you a first **portfolio commit** on GitHub

Get the definitions crisp now and the rest of the certificate gets easier.

Your Path Through Module 1

1. What AI is: the nested layers from AI down to LLMs
2. How machines learn: paradigms, training loop, training vs inference
3. The AI toolbox: NLP, vision, speech, generative models, agents
4. Responsible AI: IBM's five pillars and a fairness check
5. Build: spam filter (rules vs ML) and a perceptron from scratch

Read a part, run its code, then attempt the matching Coursera quiz.



PART 1

What AI Is, and How the Pieces Nest

Five Terms You Must Define Precisely

Artificial intelligence	Machines performing tasks that normally need human intelligence: perceiving, reasoning, learning, deciding.
Machine learning	Algorithms that **learn patterns from data** instead of being explicitly programmed with rules.
Deep learning	Machine learning with **neural networks of many layers**; excels at images, audio and text.
Generative AI	Deep-learning models that **create new content** (text, images, code, audio) resembling their training data.
Large language model	A large **transformer** trained on huge text corpora to **predict the next token**; the engine behind chatbots.

The nesting: AI ⊃ machine learning ⊃ deep learning ⊃ generative AI ⊃ LLMs.

Narrow, General and Super AI

Narrow (weak) AI	General AI (AGI)	Super AI
• Built for **specific tasks** • Spam filters, face unlock, chatbots, recommenders • **All AI deployed today**	• Would match humans across **any** intellectual task • A research goal • **Not achieved**	• Would surpass humans **everywhere** • Hypothetical • A subject of debate and speculation

Exam trap: a chatbot that seems to know everything is still narrow AI.

Traditional Programming vs Machine Learning

Traditional programming	Machine learning
A human writes the **rules**	The algorithm **learns the rules** from examples
Input: **data + rules** → output: **answers**	Input: **data + answers** → output: **a model**
Breaks when the world changes	Retrain when the world changes
Easy to explain line by line	Needs good, representative data

Mini-Project A puts both approaches side by side on the same messages.

The Core Idea



Machine learning is programming with examples. You show the computer inputs and correct outputs, and it finds the pattern that connects them.



PART 2

How Machines Learn

Learning Paradigms at a Glance

Paradigm	Data	Goal	Where you meet it
Supervised	Inputs with labels	Predict a label or number	Spam detection, house prices, fine-tuning (M4, M5)
Unsupervised	Inputs without labels	Find structure: clusters, compression	K-means, PCA (M4)
Reinforcement	Rewards from an environment	Learn actions that maximise reward	Game agents; RLHF for chat models (M5)
Self-supervised	Raw data; labels made from the data	Predict a hidden or next piece	LLM pre-training (M5)

The Training Loop

1. Data: collect, clean, split into train / validation / test
2. Model: pick an architecture with adjustable weights
3. Predict: forward pass on a batch of examples
4. Loss: measure how wrong the predictions are
5. Update: backpropagation + gradient descent nudge the weights
6. Repeat until validation loss stops improving

Training vs Inference in 10 Lines

```
from sklearn.linear_model import LinearRegression

# hours studied -> exam score (labelled data)
hours  = [[1], [2], [3], [4], [5], [6]]
scores = [35, 45, 52, 61, 70, 78]

model = LinearRegression()
model.fit(hours, scores)          # TRAINING: learn weight + bias
print("weight:", round(model.coef_[0], 2),
      "bias:", round(model.intercept_, 2))

print("7 hours ->", round(model.predict([[7]])[0], 1))   # INFERENCE
```

```
OUTPUT
weight: 8.54 bias: 26.93
7 hours -> 86.7
```

- **fit()** is training: it learns a weight (slope) and a bias (intercept) from labelled pairs
- **predict()** is inference: it applies the learned weights to an input it has never seen
- Training is the expensive, occasional step; inference runs on **every user request**
- The same split holds for a 70-billion-parameter LLM

Training vs Inference

Training	Inference
Adjusts the weights using data	**Uses** the frozen weights
Needs labels or a self-made signal	Needs only the new input
Expensive: GPUs, hours to weeks	Cheap per request, but adds up at scale
Done once or occasionally	Happens every time a user asks



How LLMs Learn Without Labels

- LLMs are **self-supervised**: they make their own labels from raw text
- **GPT-style** models hide the **next** word and learn to predict it
- **BERT-style** models hide a **masked** word in the middle and predict it
- Billions of sentences become billions of free training examples
- Chat models then add **instruction tuning** and **RLHF** (reinforcement learning from human feedback) to become helpful assistants

Pre-training teaches language; fine-tuning and RLHF teach behaviour.



PART 3

The AI Toolbox

Fields of AI You Will Meet Across the Certificate

Neural networks	Layers of ‘neurons’; each computes a weighted sum plus bias, then an activation. Learning = adjusting weights (M4).
NLP	Understanding and generating language: classification, translation, summarisation, question answering (M5, M7).
Computer vision	Understanding images and video: classification, detection, captioning (M3 uses an image-captioning model).
Speech	Speech-to-text and text-to-speech; Course 6 builds a voice assistant (M3).
Generative models	GANs, VAEs, diffusion models and transformers that create new content (M2, M5).
AI agents	An LLM that plans steps and **calls tools** to reach a goal (M7).

The Modern GenAI Application Stack

Application & UI	Gradio, Flask, web and mobile front ends: where users meet the model
Orchestration	LangChain chains, prompts, agents and tools that wire the steps together
Retrieval	Embeddings and vector databases (FAISS, Chroma) that fetch relevant facts
Foundation models	LLMs served through Hugging Face, IBM watsonx.ai or other providers
Compute & data	GPUs, cloud platforms and the datasets everything is trained on

By Module 8 you will have built one working slice of every layer.

AI in the Real World

Industry	Application	AI technique
Healthcare	Reading scans; summarising clinical notes	Computer vision; LLMs
Finance	Fraud detection; credit scoring; support chatbots	Classification; anomaly detection; LLMs
Retail	Recommendations; demand forecasting	Collaborative filtering; regression
Manufacturing	Defect detection; predictive maintenance	Computer vision; time series
Customer service	Virtual agents; ticket routing	LLMs + RAG; text classification
Software	Code completion, tests, documentation	Code LLMs

Scenario questions ask: which technique fits this use case?



PART 4

Responsible AI

IBM’s Five Pillars of Trustworthy AI

Explainability	Can we explain **why** the model decided this? e.g. a loan model shows which factors drove a rejection.
Fairness	Does it treat individuals and groups **equitably**? e.g. compare approval rates across regions.
Robustness	Does it behave reliably under unusual or **adversarial** inputs? e.g. resisting prompt injection.
Transparency	Are users told **how it works** and what data it uses? e.g. disclosing that support is an AI.
Privacy	Is personal data **protected** and used with consent? e.g. masking IDs before text goes to an LLM.

A Five-Line Fairness Check

```
import pandas as pd

loans = pd.DataFrame({
    "region":   ["urban"]*6 + ["rural"]*6,
    "approved": [1, 1, 1, 0, 1, 1,   1, 0, 0, 1, 0, 0],
})
rates = loans.groupby("region")["approved"].mean()
print(rates.round(2).to_string())
ratio = rates.min() / rates.max()
print("Disparate-impact ratio:", round(ratio, 2))
print("Flag for review" if ratio < 0.8 else "Within 80% rule")
```

```
OUTPUT
region
rural    0.33
urban    0.83
Disparate-impact ratio: 0.4
Flag for review
```

- Group outcomes by a sensitive attribute and compare **approval rates**
- The **80% rule** is a common screening heuristic: the lower rate should be at least 80% of the higher
- 0.40 here means the rural group needs investigation
- A flag starts a review; it does not prove the model is unfair on its own

Four Risks to Manage

Hallucination

- Fluent but **false** output
- Ground answers in sources (RAG, M7)
- Ask for citations; verify

Bias

- Models inherit **data biases**
- Measure outcomes by group
- Fix data and thresholds

Data & IP

- Respect **copyright** and consent
- India's DPDP Act; EU AI Act
- Mask personal data

Governance

- Policies and **human oversight**
- Model documentation
- Monitor after launch



MINI-PROJECT A

Spam Filter: Hand-Written Rules vs Machine Learning

Project Brief

- **Goal**: classify short messages as **spam (1)** or **ham (0)**
- **Approach 1**: a hand-written keyword rule, the traditional-programming way
- **Approach 2**: a **Naive Bayes** model that learns word patterns from 20 labelled messages
- **Evaluate** both on 6 messages neither has seen, using accuracy
- **Tools**: Python 3, scikit-learn (runs in seconds on any laptop or Colab)

This is the whole of machine learning in miniature: data, model, training, inference, evaluation.

Step 1: Labelled Data

```
# (text, label)  1 = spam, 0 = ham (not spam)
msgs = [
 ("WIN a FREE iPhone now, click the link", 1),
 ("Congratulations! You won a cash prize", 1),
 ("Are we still meeting for lunch tomorrow?", 0),
 ("Please send the lab report by Friday", 0),
 # ... 20 training messages in total (see notes)
]
test = [
 ("FREE entry: win tickets, text now", 1),
 ("Your parcel is held, pay a small fee", 1),
 ("Is the free workshop open to juniors?", 0),
 # ... 6 unseen test messages in total
]
```

- Each example is a **(text, label)** pair: supervised learning
- **Training set**: the examples the model learns from
- **Test set**: held back to measure performance on unseen messages
- Never test on training data: that measures memory, not learning
- The full 26-message dataset is in this slide's speaker notes

Step 2: The Rule-Based Filter

```
SPAM_WORDS = {"free", "win", "won", "cash", "prize", "urgent", "offer"}

def rule_filter(text):
    words = {w.strip("!,.:?").lower() for w in text.split()}
    return int(len(words & SPAM_WORDS) > 0)    # 1 if any spam word

def accuracy(pred, gold):
    return sum(p == g for p, g in zip(pred, gold)) / len(gold)

X_test = [t for t, _ in test]
y_test = [y for _, y in test]
rule_pred = [rule_filter(t) for t in X_test]
print("Rule-based accuracy :", round(accuracy(rule_pred, y_test), 2))
```

OUTPUT

```
Rule-based accuracy : 0.67
```

- A **set intersection** finds any spam keyword in the message
- Simple, fast and fully explainable
- But a human must **guess** every keyword in advance
- Scammers change wording; the rule list never keeps up

Step 3: The Learned Filter

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB

X_train = [t for t, _ in msgs]
y_train = [y for _, y in msgs]

vec = CountVectorizer(lowercase=True)      # text -> word counts
Xtr = vec.fit_transform(X_train)           # learn the vocabulary
model = MultinomialNB().fit(Xtr, y_train)  # TRAINING

ml_pred = model.predict(vec.transform(X_test))   # INFERENCE
print("Learned (NB) accuracy:", round(accuracy(ml_pred, y_test), 2))
```

```
OUTPUT
Learned (NB) accuracy: 0.83
```

- **CountVectorizer** turns each message into word counts
- **MultinomialNB** learns how often each word appears in spam vs ham
- **fit()** = training; **predict()** = inference
- No keyword list: the model found its own evidence

Step 4: Compare Message by Message

```
for t, r, m, g in zip(X_test, rule_pred, ml_pred, y_test):
    print(f"{t[:30]:30} rule={r} ml={m} true={g}")

new = ["Claim your free cash now"]
p = model.predict_proba(vec.transform(new))[0][1]
print("P(spam) for new msg:", round(p, 2))
```

OUTPUT

```
FREE entry: win tickets, text  rule=1 ml=1 true=1
Exclusive cash bonus waiting,  rule=1 ml=1 true=1
Your parcel is held, pay a sma rule=0 ml=0 true=1
Meeting notes attached, see yo rule=0 ml=0 true=0
Can we move the lab to Thursda rule=0 ml=0 true=0
Is the free workshop open to j rule=1 ml=0 true=0
P(spam) for new msg: 0.96
```

- The rule wrongly blocks a **free workshop**; the model does not
- Both miss the **parcel scam**: it shares no words with the training spam
- **predict_proba** gives a confidence, useful for a review queue
- Lesson: models are only as good as the variety in their data

Results

0.67
accuracy of the hand-written keyword rule

0.83
accuracy of the learned Naive Bayes model

0.96
P(spam) the model gives a brand-new scam message

- Both scored on the **same 6 unseen** messages
- With 6 test messages, one error moves accuracy by 0.17: always check **sample size**
- Next step in real life: thousands of messages, plus precision and recall (M4)

Try It Yourself

- Add **five** delivery-scam messages to the training set and re-run: does the parcel scam get caught?
- Add **‘free’** to a few ham messages: watch how the model’s confidence changes
- Swap **CountVectorizer** for **TfidfVectorizer** and compare accuracy
- Print **confusion_matrix(y_test, ml_pred)** from sklearn.metrics and name each cell
- Push the notebook to GitHub with a README that states the result in one sentence

Write down your prediction before each run. Being surprised is how you learn.



MINI-PROJECT B

Build a Perceptron, the First Neural Network, From Scratch

How One Artificial Neuron Works

Inputs x	Numbers describing the example, e.g. two switches that are 0 or 1
Weights w	How much each input matters; these are what the neuron **learns**
Bias b	Shifts the decision threshold up or down
Weighted sum	$z = w \cdot x + b$: multiply, add up, add the bias
Activation	Step function: output 1 if $z > 0$, else 0
Update rule	$w \leftarrow w + lr \times error \times x$: nudge weights toward the right answer

Step 1: The Perceptron Class

```
class Perceptron:          # a single neuron (uses numpy as np)
    def __init__(self, n_inputs, lr=0.1):
        self.w = np.zeros(n_inputs)   # weights
        self.b = 0.0                  # bias
        self.lr = lr                  # learning rate

    def predict(self, x):
        z = np.dot(self.w, x) + self.b   # weighted sum
        return 1 if z > 0 else 0         # step activation

    def fit(self, X, y, epochs=10):
        for epoch in range(epochs):
            errors = 0
            for xi, target in zip(X, y):
                error = target - self.predict(xi)   # how wrong?
                self.w += self.lr * error * xi       # update rule
                self.b += self.lr * error
                errors += int(error != 0)
            print(f"epoch {epoch+1}: mistakes={errors}")
            if errors == 0: break
        return self
```

- **predict()**: weighted sum, then step activation
- **fit()**: for each example, compute the error (target − prediction)
- Correct answers give error 0, so nothing changes
- Mistakes move the weights by **lr × error × input**
- Stop early once an epoch has **zero mistakes**

Step 2: Train It on AND, Then Try XOR

```
import numpy as np

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

y_and = np.array([0, 0, 0, 1])
p = Perceptron(2).fit(X, y_and)
print("weights:", p.w, "bias:", round(p.b, 2))
print("AND:", [p.predict(x) for x in X])

y_xor = np.array([0, 1, 1, 0])
q = Perceptron(2).fit(X, y_xor, epochs=4)
print("XOR:", [q.predict(x) for x in X], "target [0, 1, 1, 0]")
```

OUTPUT

```
epoch 1: mistakes=1
epoch 2: mistakes=3
epoch 3: mistakes=3
epoch 4: mistakes=2
epoch 5: mistakes=1
epoch 6: mistakes=0
weights: [0.2 0.1] bias: -0.2
AND: [0, 0, 0, 1]
epoch 1: mistakes=2
epoch 2: mistakes=3
epoch 3: mistakes=4
epoch 4: mistakes=4
XOR: [1, 1, 0, 0] target [0, 1, 1, 0]
```

AND is learned in 6 epochs. XOR never settles: no single straight line separates its classes.

Why XOR Matters

- A single perceptron draws **one straight line**; AND and OR are linearly separable, XOR is not
- In 1969 this limit helped slow neural-network research for years
- The fix: **hidden layers** with non-linear activations, the multi-layer perceptron
- Training many layers needs **backpropagation** and gradient descent (M4)
- Stack enough layers and you reach deep learning, then transformers and LLMs (M5)

Every LLM is, at heart, billions of these weighted sums with smarter activations.

Try It Yourself

- Train the perceptron on **OR** ([0, 1, 1, 1]); how many epochs does it need?
- Change **lr** to 1.0 and 0.01: does the number of epochs change? Why?
- Print the weights after every epoch and sketch the decision line w1·x1 + w2·x2 + b = 0
- Use sklearn's **MLPClassifier(hidden_layer_sizes=(4,))** to solve XOR
- Explain in two sentences why the hidden layer helps

Where This Certificate Can Take You

Role	What they do	Key skills (module)
GenAI / LLM engineer	Builds apps on LLMs: RAG, agents, fine-tuning, evaluation	Python, LangChain, Hugging Face, PEFT (M5, M7, M8)
ML engineer	Trains, deploys and monitors models in production	PyTorch, scikit-learn, APIs, testing (M3, M4)
Data scientist	Analyses data, builds predictive models, explains insight	pandas, statistics, ML (M4, M6)
NLP engineer	Language models for search, classification, chat	Tokenisation, transformers (M5)
AI app developer	Wraps models in web apps and APIs	Flask, Gradio, deployment (M3)



WRAP-UP

Summary, Knowledge Checks and Key Terms

What You Built and Learned

- **AI ⊃ ML ⊃ deep learning ⊃ generative AI ⊃ LLMs**; all deployed AI today is narrow
- Four paradigms: **supervised, unsupervised, reinforcement, self-supervised**
- The loop: data → model → predict → loss → update; **training** adjusts weights, **inference** uses them
- IBM's pillars: **explainability, fairness, robustness, transparency, privacy**
- You built a **spam filter** two ways and saw learning beat hand-written rules
- You built a **perceptron** and saw why hidden layers are needed



Knowledge Check 1

A team trains a model on millions of emails labelled spam or not spam. Which paradigm is this?

a. Unsupervised learning

b. **Supervised learning**

c. Reinforcement learning

d. Self-supervised learning

The correct answer is b. Every training example comes with a label (spam / not spam), so the model learns a mapping from inputs to known answers.

Knowledge Check 2

A popular chatbot answers questions on almost any topic. How should it be classified?

a. General AI, because it covers many topics

b. Super AI, because it is faster than humans

c. **Narrow AI, because it works within its training data and tools**

d. Not AI, because it only predicts tokens

The correct answer is c. However broad it seems, it performs a specific task (next-token prediction) within learned patterns. AGI remains hypothetical.

Knowledge Check 3

In Mini-Project B, why can a single perceptron never learn XOR?

a. The learning rate is too small

b. **XOR is not linearly separable**

c. It needs more epochs

d. The bias is fixed at zero

The correct answer is b. One perceptron draws a single straight decision line. No straight line separates XOR's classes; hidden layers are needed.

Key Terms from This Module

Narrow AI: AI built for a specific task; all AI deployed today

Machine learning: learning patterns from data instead of hand-written rules

Deep learning: machine learning with many-layer neural networks

Generative AI: models that create new content

LLM: large transformer that predicts the next token

Supervised learning: learning from labelled examples

Self-supervised: labels created from the data itself

RLHF: reinforcement learning from human feedback

Training: adjusting weights using data

Inference: using trained weights on new inputs

Perceptron: a single neuron: weighted sum, bias, step activation

Hallucination: fluent but false model output

Disparate impact: outcome-rate ratio between groups; the 80% rule

Explainability: being able to say why a model decided



End of Module 1

Next: Module 2 · Generative AI and Prompt Engineering

Prabhat Kumar

Chief Learning Officer, ML360 Educations · Training enquiries: WhatsApp 7972642955