

Machine Learning in Engineering Science

A Colab-first course reader

Chi-Hua Jonny Yu

Fall 2026

How to use this reader

This is the Fall 2026 course reader for *Machine Learning in Engineering Science*. It replaces the 2025 model-by-model textbook structure with a learning sequence that follows the course calendar. The reader is not a catalogue of architectures. Its central question is: *what evidence is sufficient to support an engineering decision?*

Each teaching week combines about 90 minutes of lecture with 90 minutes of practice. For Week 7 these modes are distributed across three independently usable 60-minute modules: sequential data and LSTM, attention, then Transformer architecture and experiments. The website provides the matching notebooks and slide decks. This source keeps the narrative, equations, design rationale, and references in version-controlled LaTeX.

Learning outcomes

By the end of the course, students should be able to:

- formulate an ML problem with a target, features, split, metric, and decision context;
- implement and evaluate a reproducible baseline before seeking a more complex model;
- identify data leakage, overfitting, uncertainty, and failure cases;
- explain when an LLM, retrieval component, tool call, or agent loop is justified; and
- use AI coding assistance while retaining responsibility for tests, evidence, and rollback.

Semester map

Weeks	Course focus	Reader unit and tangible output
1–3	Data workflows and supervised learning	Chapters 1 and 2 ; a question, a table of features, and a baseline model.
5–6	Evaluation and representation	Chapters 3 and 4 ; a defensible split, metric, and representation.
7	Sequences, LSTM, attention, and Transformers	Chapters 5 and 6 ; three one-hour modules with reproducible experiments.
9–10	Reliability and project studio	Chapter 7 ; a reproducible midterm ML workflow.
11	Midterm assessment	Project demonstration, written examination, and oral examination.
12–14	LLMs, retrieval, tools, and agent design	Chapters 8 to 10 ; an agent plan with explicit tools and failure cases.
15–17	Vibe coding, tests, and final studio	Chapter 11 ; an evaluated engineering workflow and final demo.
18	Final assessment	Final project demonstration, written examination, and oral examination.

What is assessed

Labs are active classwork and do not create a weekly graded workload. There are four milestone-based items called *Homework*: an ML problem brief, a reproducible ML workflow, an agent plan, and demo-readiness evidence. The midterm and final each combine a project with written and oral assessment.

A note on AI tools

The course permits free or basic-credit AI tools for drafting code, explanations, and debugging. A generated answer is never sufficient evidence. Submissions must show the model or system used, the prompt or change request when relevant, the tests run, the observed result, and the student's explanation of limits. No project requires a paid API, a dedicated GPU, or a competition ranking.

Contents

How to use this reader	i
I Machine Learning Foundations	1
1 Data, Questions, and the ML Workflow	2
1.1 From an engineering question to a learning task	2
1.2 A reproducible first pass	2
1.3 Data contracts	3
2 Supervised Learning: Baselines Before Complexity	4
2.1 Classification and regression	4
2.2 Split before learning from the data	4
2.3 Algorithm choice	5
3 Evaluation, Validation, and Generalization	6
3.1 Metrics carry assumptions	6
3.2 Generalization and the bias–variance trade-off	6
3.3 Validation discipline	7
4 Unsupervised Learning and Representation	8
4.1 Clustering as a hypothesis generator	8
4.2 Representation matters	8
5 Neural Networks and Deep Learning Foundations	10
5.1 A compact mathematical view	10
5.2 Training is an experiment	10
6 Sequential Data, LSTM, Attention, and Transformers	12

6.1	Learning outcomes and evidence	12
6.2	Hour 1: ordered observations and recurrent memory	12
6.2.1	Order, sampling, and the prediction task	12
6.2.2	Chronological splits and available information	13
6.2.3	Baselines and error interpretation	13
6.2.4	RNN state and learning through time	13
6.2.5	The complete LSTM mechanism	14
6.2.6	Implementation, selection, and reproducibility	14
6.3	Hour 2: why attention is useful and how it works	14
6.3.1	The fixed-context bottleneck	14
6.3.2	Queries, keys, values, and shapes	15
6.3.3	An attention calculation with no hidden steps	15
6.3.4	Self-attention, cross-attention, and masks	16
6.3.5	Benefits, costs, and interpretation	16
6.4	Hour 3: assemble and test a Transformer	16
6.4.1	Tokens and position	16
6.4.2	Heads, nonlinear transformations, and residual paths	16
6.4.3	Objective, baseline, and generation	17
6.4.4	Behavioral checks and limits	17
6.5	Completion and explain-back	17
6.6	Primary sources	18
7	Reliability, Error Analysis, and the Midterm Workflow	19
7.1	Error analysis is not an appendix	19
7.2	The midterm ML workflow project	19
II	Language Models, Agents, and Engineering Workflows	21
8	Transformers and Large Language Models	22
8.1	Attention and next-token prediction	22
8.2	Use a bounded role	22
9	Retrieval, Tools, and Structured Knowledge	24
9.1	Retrieval-augmented generation	24

9.2 Tools are software interfaces	24
10 Agent Design for Engineering Workflows	26
10.1 A minimal agent loop	26
10.2 Agent plan Homework	26
10.3 Current agent engineering patterns	27
11 Testing, Vibe Coding, and the Final Project	28
11.1 Vibe coding with boundaries	28
11.2 Evaluation before demonstration	28
11.3 Final assessment	29
11.4 Risk-aware evaluation	29
A Colab Conventions and Homework Milestones	30
A.1 Colab conventions	30
A.2 Homework milestones	30
A.3 Submission rule	31
A.4 Runtime-aware Lab design	31
B Extension Studios: Deep Learning Architectures	32
B.1 Convolutional neural networks	32
B.2 Sequential models	32
B.3 Generative models	32

Part I

Machine Learning Foundations

Chapter 1

Data, Questions, and the ML Workflow

Weeks 1–2. A useful ML system begins before model selection. It begins with a decision, a measurable target, a representation of the available evidence, and a procedure that can be repeated. This chapter establishes the working vocabulary used throughout the course.

Learning outcomes

- Distinguish a decision problem from an ML task.
- Identify an observation, feature vector x , target y , and prediction $\hat{y}$.
- Create a small, inspectable data workflow in Python.
- State what a plot or summary statistic does *not* establish.

1.1 From an engineering question to a learning task

An engineering question such as “Which process condition is likely to produce a defect?” must be made operational. Define the unit of observation, the time at which the prediction is made, the target, the available features, and the decision that follows. A classification task predicts a discrete label; a regression task predicts a continuous value. Neither label is meaningful without a stated measurement process and error cost.

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ denote observed examples. A learning algorithm selects a function f_θ so that $\hat{y} = f_\theta(x)$ is useful on data not used to choose θ . The course treats this as an evidence problem: the data split and evaluation protocol are part of the model, not afterthoughts [8].

1.2 A reproducible first pass

Before fitting a model, inspect shape, column meaning, units, missingness, and plausible ranges. The following pattern is deliberately small enough to read line by line.

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4
5 rng = np.random.default_rng(2026)
6 df = pd.DataFrame({
7     "temperature_c": rng.normal(72, 5, 40),
8     "pressure_kpa": rng.normal(210, 18, 40),
9 })
10 df["defect_flag"] = (
11     (df["temperature_c"] > 75) & (df["pressure_kpa"] > 220)
12 ).astype(int)
13
14 print(df.shape)
15 print(df.describe())
16 plt.scatter(df["temperature_c"], df["pressure_kpa"],
17             c=df["defect_flag"], cmap="coolwarm")
18 plt.xlabel("Temperature (C)")
19 plt.ylabel("Pressure (kPa)")
20 plt.show()
```

A coloured scatter plot can reveal a possible association. It does not prove causality, establish a production threshold, or show that a future batch will behave similarly. The lab asks students to write both the supported observation and a limitation.

Colab Lab

Week 1: [Open in Google Colab](#). Inspect a measurement table, compute descriptive statistics, create one figure, and write an evidence statement. The target is not yet a high-scoring model; it is a clear workflow that another person can reproduce.

1.3 Data contracts

A data contract records the feature names, units, allowable values, target definition, split rule, and known missingness. It is lightweight documentation with a strong engineering benefit: it prevents silent changes from being mistaken for model improvement. For every course notebook, include a first cell that reports package versions, a random seed, data provenance, and the shape of the loaded table.

Checkpoint

For a proposed predictor of fatigue life, name: (1) one observation, (2) three candidate features, (3) the target and its unit, (4) a decision that uses the prediction, and (5) one feature that would be unavailable at prediction time. Explain why the last feature must be excluded.

Chapter 2

Supervised Learning: Baselines Before Complexity

Weeks 2–3. Supervised learning uses labelled examples to learn a mapping from features to a target. The practical discipline is to establish a transparent baseline, then compare alternatives under the same split and metric.

Learning outcomes

- Separate feature engineering from target definition.
- Construct train and test partitions without leakage.
- Fit a baseline regression or classification pipeline.
- Interpret a confusion matrix and a residual plot in context.

2.1 Classification and regression

For binary classification, the model estimates a score or probability $p_{\theta}(y = 1 \mid x)$. A threshold turns the score into an action. The threshold should follow the cost of false positives and false negatives, not a default convention. For regression, a common starting loss is mean squared error,

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2,$$

but an error in megapascals, minutes, or dollars must be interpreted in its application context.

2.2 Split before learning from the data

Data leakage occurs when information from the test set changes feature construction, scaling, imputation, model selection, or threshold choice. A pipeline makes the correct order explicit: fit transformations on training data only, then apply the learned transformations to held-out data.

```
1 from sklearn.datasets import load_wine
2 from sklearn.model_selection import train_test_split
3 from sklearn.pipeline import Pipeline
4 from sklearn.preprocessing import StandardScaler
5 from sklearn.linear_model import LogisticRegression
6 from sklearn.metrics import classification_report
7
8 wine = load_wine(as_frame=True)
9 X_train, X_test, y_train, y_test = train_test_split(
10     wine.data, wine.target, test_size=0.25,
11     random_state=2026, stratify=wine.target,
12 )
13 model = Pipeline([
14     ("scale", StandardScaler()),
15     ("classifier", LogisticRegression(max_iter=2000)),
16 ])
17 model.fit(X_train, y_train)
18 print(classification_report(y_test, model.predict(X_test)))
```

A logistic-regression pipeline is not “simple” in the sense of being disposable. It establishes a reproducible reference point. A more complex model earns its place only when the evaluation protocol demonstrates an improvement relevant to the decision.

2.3 Algorithm choice

Decision trees expose rule-like splits; nearest-neighbour methods compare local examples; support vector machines seek a separating margin; ensembles combine diverse predictors. These methods remain useful when the dataset is structured, moderate in size, and the evaluation contract is clear [1, 14]. The course uses them to develop comparison discipline, rather than to encourage architecture shopping.

Colab Lab

Week 2: [Open in Google Colab](#). Build a leakage-safe baseline classifier, report accuracy and a confusion matrix, then change one decision and document its effect. Your explanation is more important than the numerical score alone.

Checkpoint

A feature is calculated using the mean of every row in the dataset before the train/test split. Is this leakage? Describe the corrected order of operations and explain why it matters even if the measured score changes only slightly.

Chapter 3

Evaluation, Validation, and Generalization

Week 5. A score is evidence only when it was produced by a protocol that matches the intended use. This chapter makes error analysis, validation, and model selection explicit.

Learning outcomes

- Select metrics that expose the errors that matter.
- Explain underfitting, overfitting, and the bias–variance trade-off.
- Use cross-validation without contaminating the final test set.
- Turn model errors into a next experimental question.

3.1 Metrics carry assumptions

Accuracy is the fraction of correct predictions, but it can hide failures when classes are imbalanced. Precision asks whether positive predictions are credible; recall asks whether actual positives are found. For a binary classifier,

$$\text{precision} = \frac{TP}{TP + FP}, \quad \text{recall} = \frac{TP}{TP + FN}.$$

Choose the metric before inspecting the final score. A defect-screening model might favour recall, while an expensive destructive test may require higher precision.

3.2 Generalization and the bias–variance trade-off

Training performance can improve while test performance degrades. A model with too little capacity underfits; a model tuned too aggressively to a small training set overfits. The remedy is not always a larger model. Better measurements, fewer unstable features, regularization, or a clearer task definition may be more valuable.

3.3 Validation discipline

Use training data to fit candidate pipelines, validation or cross-validation to select among them, and a final test set for one honest estimate. Record the split seed, preprocessing, metric, and selection rule. Do not repeat test-set evaluation until it becomes an informal validation loop.

Colab Lab

Week 5: compare two pipelines using the same split and metric. Make a plot of training and validation behaviour, then write a short error analysis: which cases fail, what shared property might explain the failure, and what additional data would reduce uncertainty?

Checkpoint

A classifier has 98% accuracy but identifies only 2 of 40 safety-critical failures. Which additional metric would you inspect first, and what decision trade-off does it reveal?

Chapter 4

Unsupervised Learning and Representation

Week 6. Unsupervised methods organize data without a labelled target. They are useful for exploration, anomaly investigation, compression, and finding candidate subpopulations. They do not automatically create ground truth.

Learning outcomes

- Explain the difference between clustering and classification.
- Standardize features appropriately before distance-based analysis.
- Use a low-dimensional view as an exploratory tool rather than a final conclusion.
- State how a discovered cluster would be checked against domain knowledge.

4.1 Clustering as a hypothesis generator

Methods such as k -means partition observations by geometric similarity. The algorithm returns groups even when no meaningful groups exist. Therefore a cluster label should be treated as a hypothesis to investigate with process knowledge, additional measurements, and sensitivity analysis.

4.2 Representation matters

Distances depend on feature scale and representation. A temperature difference of 10 degrees and a pressure difference of 10 kPa are not inherently comparable. Document scaling, missing-value handling, and the variables intentionally excluded from the analysis.

Colab Lab

Week 6: standardize a multivariate dataset, compare two values of k , and inspect the stability of the result. Identify one observation that would require domain review rather than automatic labelling.

Chapter 5

Neural Networks and Deep Learning Foundations

Week 7. Deep learning learns representations through composed differentiable functions. The course focuses on the training loop, loss surface, and diagnostic signals needed to use a neural network responsibly.

Learning outcomes

- Describe a forward pass, loss, gradient, and parameter update.
- Relate activation functions and network capacity to optimization behaviour.
- Read a training curve for divergence, underfitting, and overfitting.
- Compare a neural baseline with a classical baseline under the same protocol.

5.1 A compact mathematical view

For one layer, $h = \phi(Wx + b)$, where W and b are learned parameters and ϕ is a nonlinear activation. A loss $\mathcal{L}(y, \hat{y})$ measures discrepancy. Backpropagation applies the chain rule to compute $\nabla_{\theta} \mathcal{L}$ efficiently [12]. Gradient-based optimization then updates parameters, for example $\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L}$.

5.2 Training is an experiment

Fix seeds where possible, retain the split, log loss and metric traces, and compare a neural network to a simpler baseline. A training curve is not a certificate of generalization. It is an instrument that helps identify whether the next action is to inspect data, change capacity, regularize, or stop.

Colab Lab

Week 7: train a small PyTorch network on a compact dataset using CPU or optional Colab acceleration. Keep the network deliberately small. Compare it to a classical baseline and write what the extra capacity changed.

Chapter 6

Sequential Data, LSTM, Attention, and Transformers

Week 7: Monday 19 October 2026, 14:00–17:00.

The three one-hour modules follow one question: how can a learned model use earlier information to make a defensible prediction? Hour 1 develops recurrent memory, Hour 2 develops query-dependent retrieval, and Hour 3 assembles and tests a causal Transformer. About 90 minutes of explanation and worked reasoning are combined with 90 minutes of prediction, guided computation, experimentation, and explain-back across the three hours.

6.1 Learning outcomes and evidence

Students should be able to define a forecast origin and horizon; construct windows without future leakage; trace every LSTM gate; compute attention scores, weights, and a context vector; distinguish self-attention from cross-attention; explain positional information, multi-head attention, residuals, and feed-forward layers; and test a decoder’s causal behavior. The three independent CPU notebooks compare an LSTM against baselines, verify attention arithmetic, and train and inspect a small Transformer. All data are synthetic or self-contained. No paid API or GPU is required. These are class exercises, not additional weekly graded assignments.

6.2 Hour 1: ordered observations and recurrent memory

6.2.1 Order, sampling, and the prediction task

A sequence is an ordered collection $x_1, \dots, x_T$ in which order is part of the problem definition. Sensor samples, process events, and language tokens are examples. $(1, 2, 4, 8)$ and $(8, 1, 4, 2)$ have the same multiset and mean but different transitions. A representation that discards order cannot recover those transitions from the values alone. Equally spaced samples and irregular events also require different sampling assumptions. A sequence model does not automatically infer omitted time intervals.

In the example, the forecast origin is immediately before observing x_t . With window length L ,

$$X_t = (x_{t-L}, \dots, x_{t-1}), \quad y_t = x_t.$$

The Python slice `signal[t-L:t]` excludes index t . Every input index must be strictly less than the target index. This is a one-sample horizon. A fixed-origin multi-step forecast cannot use later actual values that would not yet have been observed.

With B histories, T time steps, and F features at each step, batch-first input has shape (B, T, F) . Many-to-one predicts one value per history; many-to-many predicts outputs at multiple positions. Batch size is not sequence length, and an ID is not necessarily a physical feature.

6.2.2 Chronological splits and available information

The synthetic sensor uses 1600 samples, window length 24, training targets before index 960, validation targets 960–1279, and test targets 1280 onward. This produces 936 training, 320 validation, and 320 test windows. Target periods are disjoint. A test window may use earlier observed validation or test sensor history. This is explicitly a rolling one-step protocol, not a forecast of all future values from one initial origin.

Normalization learns statistics, so fit it on training observations only:

$$z_t = (x_t - \mu_{\text{train}}) / \sigma_{\text{train}}.$$

Global means can leak future distribution information even without labels. Random splits of heavily overlapping windows can produce nearly identical training and test histories. For multiple devices or people, group separation may be needed in addition to temporal separation.

6.2.3 Baselines and error interpretation

Persistence uses $\hat{x}_t = x_{t-1}$ and has no fitted parameters. Ridge regression uses a regularized linear map of the flattened history window. Select regularization on validation, then compare the same test targets:

$$\text{MAE} = \frac{1}{N} \sum_n |\hat{y}_n - y_n|, \quad \text{RMSE} = \sqrt{\frac{1}{N} \sum_n (\hat{y}_n - y_n)^2}.$$

Report original target units and inspect residuals against time or operating range. RMSE emphasizes large errors. Neither score alone reveals every failure. A linear model can outperform an LSTM on a small, structured signal; report the observed comparison rather than choosing the more complex name.

6.2.4 RNN state and learning through time

A basic recurrent network reuses one update at every time step:

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b).$$

Unrolling displays repeated applications of shared parameters. The state is a learned summary, not necessarily an interpretable physical variable. Backpropagation through time includes products of

Jacobians:

$$\frac{\partial h_t}{\partial h_{t-k}} = J_t J_{t-1} \cdots J_{t-k+1}.$$

Repeated contractions may shrink gradients, and repeated expansions may amplify them. A scalar example 0.9¹⁰⁰ illustrates one contraction mechanism but not a complete nonlinear Jacobian product. Gradient clipping limits large gradient magnitude; it does not itself solve vanishing gradients.

6.2.5 The complete LSTM mechanism

An LSTM separates cell memory c_t from hidden output h_t . With $u_t = [h_{t-1}; x_t]$, a standard cell uses

$$f_t = \sigma(W_f u_t + b_f), \quad i_t = \sigma(W_i u_t + b_i), \quad (6.1)$$

$$g_t = \tanh(W_g u_t + b_g), \quad o_t = \sigma(W_o u_t + b_o), \quad (6.2)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t, \quad h_t = o_t \odot \tanh(c_t). \quad (6.3)$$

Here $\sigma(z) = 1/(1 + e^{-z})$ and $\odot$ is elementwise multiplication. Each gate is a vector of coefficients, generally one per hidden dimension. The forget gate retains old cell content; the input gate scales a signed candidate write; the output gate exposes a filtered view of the updated cell. The additive memory path can improve gradient flow but does not guarantee unlimited retention or eliminate every optimization difficulty. The direct cell-state path contains products of forget gates when their other dependencies are held fixed; the full derivative is more complicated.

Choose illustrative values $c_{t-1} = 0.8$, $f_t = 0.9$, $i_t = 0.2$, $g_t = -0.5$, and $o_t = 0.7$. Then

$$c_t = 0.72 - 0.10 = 0.62, \quad h_t = 0.7 \tanh(0.62) \approx 0.386.$$

These are hand-chosen teaching values, not training outputs. If $f_t = 1$, new content can still change the cell. Changing o_t changes hidden output without changing this cell update. Predict those effects before using sliders.

6.2.6 Implementation, selection, and reproducibility

With `batch_first=True`, input/output layouts are (B, T, F) and (B, T, H) . Ordinary non-projected hidden and cell states retain layout (layers $\times$ directions, B, H). PyTorch concatenates gate chunks in input, forget, candidate, output order. The notebook reconstructs a learned cell and compares it numerically with `nn.LSTM`. Training updates use training data, validation selects a checkpoint, and test is evaluated afterward. Save raw data identity, seed, versions, window, split boundaries, normalization, model state, baselines, and measured errors. A seed alone does not guarantee bitwise agreement across environments.

6.3 Hour 2: why attention is useful and how it works

6.3.1 The fixed-context bottleneck

A final-state encoder compresses a source sequence into one fixed-width context. Different output steps may need different earlier details. Attention makes encoder states addressable and forms a

context for each query. It does not guarantee correct retrieval or unlimited context. Attention can be combined with recurrence: Bahdanau and colleagues used recurrent sequence-to-sequence modeling in a 2014 preprint presented in 2015. The Transformer architecture came later. Distinguish a retrieval mechanism from the entire architecture that uses it.

6.3.2 Queries, keys, values, and shapes

Queries encode a retrieval need. Keys encode matching information, while the corresponding value rows carry content to combine. For N_q queries, N_k keys, key width d_k , and value width d_v ,

$$Q \in \mathbb{R}^{N_q \times d_k}, \quad K \in \mathbb{R}^{N_k \times d_k}, \quad V \in \mathbb{R}^{N_k \times d_v}.$$

Values need not equal keys or have their width. A natural-language question is not the same object as the query vector produced by a learned projection.

Additive attention can use

$$e_{ij} = v_a^\top \tanh(W_q q_i + W_k k_j + b).$$

Scaled dot-product attention uses

$$S = QK^\top / \sqrt{d_k}, \quad A_{ij} = \frac{\exp(S_{ij})}{\sum_m \exp(S_{im})}, \quad C = AV.$$

Scores compare, row-wise softmax normalizes, and value multiplication retrieves. Rows are query destinations and columns are key sources. The context has shape (N_q, d_v) .

Under independent zero-mean unit-variance coordinate assumptions, the raw dot-product variance grows with d_k . Scaling by $\sqrt{d_k}$ controls that growth; it is not cosine normalization. Stable softmax subtracts the row maximum before exponentiation without changing normalized probabilities. Before later projections and residuals, the weighted context lies within the convex hull of the available values.

6.3.3 An attention calculation with no hidden steps

Choose teaching vectors

$$Q = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad K = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad V = \begin{bmatrix} 10 & 0 \\ 0 & 10 \\ 10 & 10 \end{bmatrix}.$$

Raw dot products are $(1, 0, 1)$ and scaled scores are approximately $(0.7071, 0, 0.7071)$. Softmax gives weights $(0.4011, 0.1978, 0.4011)$. The zero raw score does not give zero weight, because $\exp(0) = 1$. Combining corresponding value coordinates gives

$$\begin{aligned} C_1 &= 0.4011(10) + 0.1978(0) + 0.4011(10) \approx 8.0222, \\ C_2 &= 0.4011(0) + 0.1978(10) + 0.4011(10) \approx 5.9889. \end{aligned}$$

Use full precision in code. Change only the query and predict the new ranking. An optional positive temperature divides the scores: below one sharpens weights, above one softens them. The standard example uses one.

6.3.4 Self-attention, cross-attention, and masks

Self-attention projects one sequence X into $Q = XW_Q$, $K = XW_K$, and $V = XW_V$. Cross-attention derives queries from one stream and keys/values from another. An encoder-decoder model uses decoder queries with encoder memory. A decoder-only model normally has no separate source encoder.

In causal next-token training, input position i predicts token $i + 1$, so it may access itself and earlier inputs: $j \leq i$. Set future scores to $-\infty$ before softmax, yielding zero weights. Padding masks hide nonexistent tokens instead. Combine permissions and exclude padded query outputs from loss. An all-masked row has no valid normalization; prevent it or specify a deliberate special case. Mask conventions differ across APIs. PyTorch scaled-dot-product attention uses Boolean True for allowed entries, whereas some other attention calls use True for blocked entries.

6.3.5 Benefits, costs, and interpretation

One dense attention layer provides a direct query-to-key computational path to a distant position. Known query operations can run in parallel. Dense score interactions involve T^2 pairs per head: doubling context length quadruples pair count. Exact fused implementations can reduce intermediate memory without changing dense pair-count scaling.

Attention weights describe routing within the computation; they do not prove causal importance. Keep Q and K fixed and change V : weights stay identical but the result can change. Removing and renormalizing a source is a different intervention. Explain what a particular experiment supports rather than interpreting an attractive heatmap as a complete causal explanation.

6.4 Hour 3: assemble and test a Transformer

6.4.1 Tokens and position

Token IDs index learned vectors; their numeric magnitude has no inherent semantic distance. The teaching model has eight symbolic tokens, width 32, four heads, and context length 16. Vocabulary size and embedding width differ. Unmasked attention without positional information is permutation-equivariant: permuting input rows permutes outputs correspondingly. Order must therefore be represented deliberately. Our model adds learned absolute position vectors. The original Transformer used sinusoidal encodings; other position mechanisms also exist. A causal mask already introduces order constraints, so do not apply the unmasked equivariance claim unchanged to a masked model.

6.4.2 Heads, nonlinear transformations, and residual paths

Each head uses learned projections and its own routing weights. Concatenate head outputs and apply a learned output projection. At fixed total width D , H heads have width D/H in this implementation. More heads do not automatically multiply the total width, and heads have no assigned semantic roles. Attention mixes positions, while a shared tokenwise MLP transforms features independently at each position. The notebook expands to $4D$, applies GELU, and projects back to D .

Residual addition requires matching dimensions and preserves an identity route. The teaching decoder uses pre-normalization:

$$\begin{aligned} u &= x + \text{Attention}(\text{LN}(x)), \\ y &= u + \text{FFN}(\text{LN}(u)). \end{aligned}$$

The 2017 paper used a full encoder-decoder with post-normalization and ReLU. Our one-block decoder is an explicitly identified educational variant, not a verbatim copy of that architecture. Its causal self-attention has no encoder-decoder cross-attention sublayer.

6.4.3 Objective, baseline, and generation

Shift targets one token ahead. Logits have shape (B, T, V) , and cross-entropy minimizes

$$\mathcal{L} = -\frac{1}{BT} \sum_{b,t} \log p_{\theta}(y_{b,t} \mid x_{b,\leq t}).$$

Pass logits directly to `nn.CrossEntropyLoss`. Training computes many known positions in parallel under the causal mask. During generation, the model predicts one unknown token, appends it, and repeats. Parallel training does not imply arbitrary autoregressive generation is parallel.

Separate synthetic streams use seeds 2026, 2027, and 2028 but share one motif with small random perturbations. This is generalization to new realizations of that process, not transfer to a different engineering domain. A training-only smoothed bigram model is the reference. Uniform prediction over eight tokens has NLL $\log 8 \approx 2.079$ nats/token. Select a checkpoint by validation NLL; evaluate test afterward.

6.4.4 Behavioral checks and limits

Plot an actual trained attention head with query rows and key columns. Upper-triangle zeros are mask constraints, not learned discoveries. Change tokens only at positions 8 onward and compare logits at positions 0–7. They should remain unchanged within the declared tolerance. This tests one concrete causal behavior, not every deployment property. Separately test permutation equivariance of the unmasked operation. Removing positions from an already trained decoder is an inference intervention, not a fair separately trained architecture comparison.

The model generates symbolic states, not evidence-backed natural language. A token generator is only one possible component of an agent. An agent also needs a bounded objective, state handling, validated tools, permissions, stopping rules, and evidence review. Generation alone does not provide factual verification or controlled execution.

6.5 Completion and explain-back

H1 must produce a baseline table, residual plot, a reconstructed learned cell update, and evidence JSON. H2 must produce scores, weights, context, mask assertions, and an interpretation-limiting intervention. H3 must produce a trained-model comparison, head plot, future-token check, and configuration record. Predict the effect of one change before executing, keep the original evidence, and finish with a claim, evidence, limitation, and a test that could challenge the claim.

6.6 Primary sources

- Hochreiter and Schmidhuber (1997), *Long Short-Term Memory*: <https://doi.org/10.1162/neco.1997.9.8.1735>.
- Bahdanau, Cho, and Bengio (2014/2015), recurrent attention: <https://arxiv.org/abs/1409.0473>.
- Vaswani et al. (2017), *Attention Is All You Need*: <https://arxiv.org/abs/1706.03762>.
- Dive into Deep Learning: https://d2l.ai/chapter_recurrent-modern/lstm.html and https://d2l.ai/chapter_attention-mechanisms-and-transformers/transformer.html.
- Executable course notebooks: <https://github.com/CHJonnyYu/MLES-Fall-2026>.

Chapter 7

Reliability, Error Analysis, and the Midterm Workflow

Weeks 9–11. A model becomes useful when its limitations are visible. Reliability work connects the numerical result to the operating conditions, inputs, and risks that produced it.

Learning outcomes

- Segment error by operating condition, class, or data-quality flag.
- Separate a model result from the claim it can support.
- Create a compact model card for a small engineering workflow.
- Demonstrate a reproducible ML pipeline orally and in writing.

7.1 Error analysis is not an appendix

Start with the failures. Are they concentrated near a measurement boundary, in a rare operating condition, or in a subgroup with missing data? A confusion matrix, residual plot, and a table of representative failures provide more actionable information than an aggregate metric alone.

7.2 The midterm ML workflow project

The midterm project is a small but complete workflow: problem brief, data contract, train/test rule, baseline, metric, error analysis, and a recommendation bounded by evidence. The written examination checks conceptual understanding. The oral examination asks students to explain choices, interpret a failure case, and identify one change that would invalidate the result.

Checkpoint

A model has a strong overall score but fails mostly when a sensor is near its detection limit. State one immediate deployment restriction, one data-collection action, and one evaluation slice that must be reported.

Part II

Language Models, Agents, and Engineering Workflows

Chapter 8

Transformers and Large Language Models

Week 12. This unit provides the minimum technical frame required to use LLMs in an engineering workflow without confusing fluent output with verified knowledge.

Learning outcomes

- Describe tokenization, next-token prediction, and attention at a high level.
- Explain why a language model can produce useful text without a guaranteed factual basis.
- Distinguish prompting, retrieval, fine-tuning, and tool use.
- Identify an engineering task for which an LLM should assist rather than decide.

8.1 Attention and next-token prediction

Transformers use attention to weight relationships among tokens in a context sequence [15]. Language-model training commonly predicts the next token. This objective can produce broadly useful representations, but it does not create a built-in source of current, task-specific, or verified information.

8.2 Use a bounded role

An LLM may summarize a lab note, propose code changes, draft a test plan, or translate a structured result into prose. The workflow must state what the model may access, which claims require retrieval or calculation, and how a human checks the result. Prompting is a form of interface design, not evidence.

Colab Lab

Week 12: use a small prompt experiment to compare an unconstrained request with a request that specifies context, output format, and verification step. Record failure modes rather than ranking models by style.

Chapter 9

Retrieval, Tools, and Structured Knowledge

Week 13. Retrieval and tools reduce the need for a language model to invent information. They also introduce new interfaces that must be tested: document selection, tool arguments, return values, permissions, and error handling.

Learning outcomes

- Describe the retrieve–ground–answer pattern.
- Separate a source document, retrieved passage, model response, and citation.
- Define a simple tool contract with inputs, outputs, and failure behaviour.
- Evaluate a retrieval workflow using task-relevant test questions.

9.1 Retrieval-augmented generation

A retrieval-augmented workflow first selects relevant material, then supplies it as context for a response [7]. The model can still misunderstand or ignore the material. Therefore the output should expose which source passages were used and decline to answer when the required evidence is absent.

9.2 Tools are software interfaces

A tool call has a schema, preconditions, outputs, and errors. Treat it as a software component: validate inputs, log execution, test representative failures, and keep high-impact actions behind human confirmation. The final project may use a mock tool when a real service would require paid access or confidential data.

Colab Lab

Week 13: build a small local retrieval example from a supplied text collection. Compare an answer with and without retrieved context, then inspect one case where retrieval is insufficient.

Chapter 10

Agent Design for Engineering Workflows

Week 14. An agent is not defined by a conversational interface. In this course, an agent is a bounded loop that observes a state, selects an allowed action, records evidence, and stops under explicit conditions.

Learning outcomes

- Write a goal, state representation, action set, and stopping condition.
- Decide when a workflow needs an agent loop rather than a single prompt or script.
- Define human approvals for irreversible or high-impact actions.
- Create test cases for expected and adverse agent behaviour.

10.1 A minimal agent loop

A useful abstraction is: observe $\rightarrow$ plan $\rightarrow$ act $\rightarrow$ verify $\rightarrow$ record. The loop should be narrow enough to test. Tool use, memory, and reflection are optional components, not automatic signs of quality. Reasoning-and-action patterns such as ReAct motivate explicit tool traces, but a trace is still evidence to be inspected, not proof of correctness [16].

10.2 Agent plan Homework

Homework 3 specifies a problem boundary, inputs, permitted tools, output format, stop condition, human approval point, and at least five failure cases. The plan must be feasible with a free or basic-credit tool path and must include a non-LLM fallback for a core calculation or validation step.

Checkpoint

A proposed “research agent” can search documents, calculate a summary statistic, and send an email. Which actions require independent tests? Which action requires human confirmation? Where should the trace be stored?

10.3 Current agent engineering patterns

Contemporary agent tooling increasingly separates the decision harness from the environment in which untrusted or model-generated code runs. The practical course translation is provider-neutral: expose a small tool contract, constrain the workspace, store a trace, checkpoint useful state, and require review before a high-impact action. These boundaries make a student prototype easier to test and recover when a run fails [10]. They do not make the agent correct; they make its behaviour more observable.

Chapter 11

Testing, Vibe Coding, and the Final Project

Weeks 15–18. Vibe coding is a fast way to express an intent to an AI coding tool. It does not transfer accountability. The course uses it as an implementation technique within a testable engineering workflow.

Learning outcomes

- Turn a natural-language change request into observable acceptance criteria.
- Review an AI-generated patch for data flow, assumptions, and failure behaviour.
- Design a small test suite before or alongside an agent implementation.
- Demonstrate a final workflow with evidence, limits, and a rollback path.

11.1 Vibe coding with boundaries

A responsible request names the intended user, input/output contract, constraints, and tests. After a tool proposes code, inspect the diff, run the relevant notebook cells or tests, and explain what changed. Avoid asking a model to make broad changes to an unexamined codebase. Prefer small, reversible changes with a visible before/after result.

11.2 Evaluation before demonstration

The final project needs a compact evaluation set that includes ordinary cases, edge cases, and known failure cases. For each case, record expected behaviour, actual behaviour, evidence, and follow-up action. A polished demo without a reproducible trace does not clear the project requirement.

11.3 Final assessment

The final project demonstration explains the problem boundary, agent or workflow design, evidence from tests, remaining limitations, and a next safe step. The written examination checks conceptual transfer. The oral examination can ask students to reason about a tool contract, an AI-generated patch, an evaluation trace, a rollback, or an unexpected failure.

Colab Lab

Weeks 15–17: use the project studio to build in small steps. Preserve the baseline, record every meaningful change, and rehearse the explanation of one failure case before the final demonstration.

11.4 Risk-aware evaluation

For the final project, evaluate more than apparent task completion. Include an ordinary case, an adverse or ambiguous case, a tool failure, and a case that must escalate to a human. Record the risk, expected behaviour, observed behaviour, evidence, and mitigation. This course-level practice is consistent with the lifecycle emphasis of the NIST Generative AI Profile, while remaining small enough for a student project [9].

Appendix A

Colab Conventions and Homework Milestones

A.1 Colab conventions

Every student notebook should begin with a short environment cell, use a declared random seed where relevant, and provide a CPU-compatible path. Optional accelerator cells must not be required for assessment. Notebooks should obtain public data from a documented source or a course repository path; they must not include credentials, personal data, or hidden instructor solutions.

The course website opens each notebook through a public GitHub repository using the Colab URL pattern defined by `\CourseRepo` in `main.tex`. Replace that one configuration value when the repository is created. The notebook source remains the canonical student artifact; the rendered website is an access layer.

A.2 Homework milestones

Homework		Timing	Evidence required
ML Problem Brief		Week 5	Decision context, target, features, split proposal, metric, and one risk.
ML Workflow		Week 10	Executable notebook, baseline, evaluation, error analysis, and bounded conclusion.
Final Plan	Agent	Week 14	Goal, tools, stop condition, approval point, failure cases, and test plan.
Demo ness	Readi-	Week 16	Reproducible run, evaluation cases, trace, limitation, and rollback or safe fallback.

A.3 Submission rule

An AI tool can assist with code, prose, or debugging, but the student remains responsible for the submitted claim. Work that uses AI assistance should identify the assistance when it materially shaped the artifact, then show the validation performed by the student.

A.4 Runtime-aware Lab design

Colab can load notebooks from GitHub, but its free resource limits, hardware availability, and runtime lifetime change over time. Runtimes are ephemeral, so a notebook must install or check any required package, fetch only documented public data, and write important outputs to a student-controlled location only when the student explicitly chooses to do so [3]. Core Labs in this course are CPU-first; any GPU cell is optional and cannot be necessary for a graded result.

The current reference implementation uses scikit-learn for core ML workflows and PyTorch only after neural-network foundations. Rather than pinning the course to a fragile hosted-runtime version, each notebook records the actual package versions at execution time. This makes a future result diagnosable when managed environments change [11, 13].

Appendix B

Extension Studios: Deep Learning Architectures

These topics remain available as optional Colab studios, reading groups, or final-project extensions. They are deliberately not prerequisites for the core ML workflow.

B.1 Convolutional neural networks

CNNs use shared local filters to exploit image structure. LeNet and later residual architectures illustrate how representation depth, skip connections, and data scale change a vision workflow [6]. Use this studio when the course data are images and a classical baseline is already in place.

B.2 Sequential models

RNNs, LSTMs, and GRUs model ordered observations. Their main lesson for this course is temporal data discipline: define what is known at each prediction time and avoid future leakage [5].

B.3 Generative models

GANs formulate generation as an adversarial game [2]; diffusion models learn a reverse denoising process [4]. These are useful extension topics for representation and synthesis, but they do not replace the need for an evaluation contract.

Bibliography

- [1] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
- [2] Ian Goodfellow et al. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.
- [3] Google Colaboratory. Frequently asked questions: Resource limits and notebook runtimes. <https://research.google.com/colaboratory/faq.html>, 2026. Accessed August 2026.
- [4] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, 2020.
- [5] Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [6] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [7] Patrick Lewis et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, 2020.
- [8] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [9] National Institute of Standards and Technology. Artificial intelligence risk management framework: Generative artificial intelligence profile. Technical Report NIST AI 600-1, NIST, 2024. URL <https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf>.
- [10] OpenAI. The next evolution of the agents sdk. <https://openai.com/index/the-next-evolution-of-the-agents-sdk/>, 2026. Accessed August 2026.
- [11] PyTorch Contributors. Pytorch stable documentation. <https://docs.pytorch.org/docs/stable/>, 2026. Accessed August 2026.
- [12] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.
- [13] scikit-learn developers. scikit-learn 1.9 release history. https://scikit-learn.org/stable/whats_new.html, 2026. Accessed August 2026.
- [14] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.
- [15] Ashish Vaswani et al. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

- [16] Shunyu Yao et al. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.