

N9LYA TECHNICAL GUIDES

Artificial Intelligence

From Beginner to Expert

Jerome M. Kutche

N9LYA

Second Edition · September 2026

About This Book

There are three kinds of AI book on the shelf right now, and none of them is quite this one.

The first is the academic machine learning textbook. Rigorous, mathematical, and mostly about a field as it existed before the current era. You finish it able to derive backpropagation and unable to explain why the model on your desk just invented a citation.

The second is the prompt-tips book. Written fast, dated on arrival, and largely a collection of incantations that worked against one version of one product for about four months.

The third is the vendor tutorial — how to call this specific API, using this specific SDK, with this specific pricing page. Useful, narrow, and obsolete on the next release.

This book is written from a different position: **understand the machine, then run it yourself.**

The Approach

Mechanisms over model names. The specific models current as I write will be superseded, probably within months. The mechanisms — attention, tokenization, quantization, sampling, retrieval — change far more slowly, and understanding them is what lets you evaluate the next thing rather than waiting for someone to tell you what to think about it.

Where I name a model, treat it as a timestamped example rather than a recommendation.

Local first, where it's practical. A substantial part of this book is about running models on hardware you own. Not because cloud APIs are bad — they're frequently the right answer — but because running a model yourself teaches you things no amount of API usage will. You learn what VRAM actually constrains, what quantization actually costs, and what “context window” means when you're the one paying for the memory.

Honest about failure. Chapter 10 is about what these systems get wrong and why, and it's longer than most books allow. A tool you don't understand the failure modes of is a tool you can't use responsibly.

Evaluation gets a real chapter, because “it seems to work” is not a standard, and the gap between a demo and something you'd put in front of a customer is almost entirely evaluation.

Safety is treated as engineering, not as a disclaimer chapter bolted on at the end. Chapter 29 covers alignment, misuse, and the practical question of what happens when a system that's usually right is confidently wrong in a context that matters.

A Note on Timing

This field moves faster than publishing does. The state of things as I write, in the middle of 2026: open-weight models trail the best proprietary models by roughly a quarter, mixture-of-experts architectures dominate at the frontier, million-token contexts have become common, and a genuinely capable model now fits on a single consumer GPU.

Every one of those sentences will need revising. **The concepts in Parts I through III will not.** I've structured the book so that the durable material comes first and the fast-moving material is clearly marked as such.

Who This Is For

Beginners with technical aptitude. You don't need machine learning background. You need to be comfortable with a command line and willing to think carefully. Part I assumes nothing else.

Developers and engineers integrating AI into real systems. Parts III and IV are where you'll live.

Homelabbers and self-hosters. Part IV is written for you specifically. If you have a GPU and a spare machine, you can follow all of it.

IT and industrial people. Chapter 28 covers AI in physical and industrial systems, which is a domain with different constraints and much higher consequences than a chatbot.

Anyone who wants to think clearly about this rather than absorb whichever narrative is loudest this month.

How to Use It

Each chapter ends with a **Run Check** — things to verify, build, or test. Do them. The gap between understanding a concept and having seen it behave is enormous in this field, because the systems are stochastic and their behavior is genuinely surprising.

About the Author

Jerome M. Kutche holds the amateur radio callsign N9LYA, first licensed in 1991. He spent over three decades in industrial electrical and controls work — Allen-Bradley and Siemens PLC systems, motor control, plant electrical maintenance and supervision — holding NFPA 70E and thirty-nine other professional certifications, and before that built RF test and measurement equipment for the telecommunications industry.

He runs local language models on his own hardware — an HP ProLiant server with datacenter GPUs passed through to a virtual machine, feeding automation workflows that do real work — and has spent a considerable number of hours discovering exactly which things a 30-billion-parameter model quantized to four bits can and cannot be trusted with.

He serves as Indiana’s 44Net/AMPRNet coordinator and is trustee for two amateur radio clubs.

Part I — Foundations

Chapter 1: What AI Actually Is

The word “AI” has been applied to so many things that it has nearly stopped meaning anything. Let’s establish what’s actually being discussed.

The Nested Terms

Artificial intelligence is the broad field: getting machines to do things that seem to require intelligence. It includes approaches that have nothing to do with learning — expert systems, search algorithms, constraint solvers, the code that plays chess by examining millions of positions.

Machine learning is a subset: systems that improve at a task by processing data, rather than by being explicitly programmed with rules. This is where nearly everything current lives.

Deep learning is a subset of that: machine learning using neural networks with many layers. It’s what made the last decade’s progress possible.

Generative AI is a category of application: systems that produce new content — text, images, audio, code — rather than classifying or predicting a number.

Large language models are the specific technology behind most of what people currently mean when they say AI.

Foundation models describes the pattern of training one very large general model and adapting it to many tasks, rather than training a specific model per task. That shift is arguably the most consequential change in the field’s practice.

What Changed

Neural networks are not new. The core ideas date to the 1940s through 1980s. The reasons they suddenly worked:

The Artificial Intelligence Map

Separate the field, the methods, and the products

ARTIFICIAL INTELLIGENCE

Goal-directed systems
Broad umbrella

MACHINE LEARNING

Learn patterns
from data

DEEP LEARNING

Layered neural
representations

GENERATIVE AI

Create or transform
content

N9LYA TECHNICAL GUIDES

Figure 1: The artificial intelligence map

Data. The internet produced text and image corpora of a scale that didn't previously exist.

Compute. GPUs, built for graphics, turned out to be nearly ideal for the matrix arithmetic neural networks require. Then hardware built specifically for it.

Architecture. The transformer, in 2017, made it possible to train much larger models efficiently on much more data. Chapter 5.

Scale as a strategy. The discovery that performance improved predictably with more parameters, more data, and more compute — and that this continued far past where people expected it to stop.

None of these alone was sufficient. The combination produced a discontinuity in what was possible.

Types of Machine Learning

Supervised learning — learning from labeled examples. Show it ten thousand images labeled cat or dog; it learns to label new ones. Most classical applications: classification, regression, fraud detection, medical imaging.

Unsupervised learning — finding structure in unlabeled data. Clustering, anomaly detection, dimensionality reduction.

Self-supervised learning — creating labels from the data itself. **This is how language models are trained:** the label for “predict the next word” is just the next word, which the text already contains. That's what made training on internet-scale text possible without anyone labeling it.

Reinforcement learning — learning from reward signals through interaction. Game playing, robotics, and a key part of how modern language models are shaped after pretraining.

What These Systems Actually Do

This is the most important paragraph in the chapter.

A trained neural network is a very large function. It takes numbers in and produces numbers out. The “learning” was a process of adjusting several billion numeric parameters so that the function produces useful outputs.

There is no database of facts inside. There is no reasoning engine in the sense of a logic system. There is a function whose parameters encode statistical structure extracted from training data, such that running it produces outputs that are frequently correct and sometimes confidently wrong.

Whether that constitutes “understanding” is a genuinely open question and one people argue about with more heat than light. What's not in dispute is the mechanism: matrix multiplications and nonlinear functions, applied in sequence, many times.

Holding both facts simultaneously — that it's “just” a function, and that it does things nobody expected a function to do — is the correct posture. People who hold only the first underestimate these systems badly. People who hold only the second anthropomorphize and get surprised.

Where AI Actually Gets Used

Beyond chatbots, which dominate the discourse:

Classification and prediction — fraud detection, credit scoring, medical triage, quality inspection, predictive maintenance. Frequently using classical methods rather than deep learning, because they're more interpretable and work on tabular data.

Vision — defect detection on production lines, medical imaging, autonomous systems, security analytics.

Speech — transcription, voice interfaces, translation.

Recommendation — what you're shown on every platform.

Generation — text, images, code, audio, video.

Control — robotics, process optimization, energy management.

A great deal of the highest-value AI in the world is unglamorous classification running on tabular data, and it was working well before the current era. Chapter 26 covers it, because a surprising number of problems people reach for a language model to solve are better handled by a decision tree.

The Vocabulary

Terms you'll need immediately:

Model — the trained artifact. A file containing parameters, plus the architecture that describes how to use them.

Parameters / weights — the numbers adjusted during training. “7B” means seven billion of them.

Training — adjusting parameters using data.

Inference — running a trained model to get output. This is what you do; training is what someone else usually did.

Token — the unit of text a language model processes. Roughly a word-piece. Chapter 7.

Context window — how many tokens the model can consider at once.

Prompt — the input you provide.

Fine-tuning — additional training on a specific dataset to specialize a general model.

Hallucination — confidently generating false information. Chapter 10.

Quantization — reducing the numeric precision of parameters to save memory. Chapter 17.

Embedding — a vector representation of meaning. Chapter 12.

Run Check

1. Identify three AI systems you interact with weekly. Which category is each?
2. For one of them, guess whether it's a language model, a classifier, or a recommender. Why?
3. Find the parameter count of a model you've heard of. What does that number mean physically?

Chapter 2: How Machine Learning Works

Before neural networks, the underlying idea. This chapter is the mechanism, stripped of jargon.

The Core Loop

Every supervised learning system does the same four things, repeatedly:

1. **Make a prediction** using current parameters.
2. **Measure how wrong it was** against the known answer.
3. **Determine how each parameter contributed** to the error.
4. **Adjust each parameter slightly** to reduce the error.

Repeat several million times. That's training.

Everything else — architectures, optimizers, learning rate schedules — is refinement of that loop.

How Machine Learning Learns

Training changes parameters; evaluation tests generalization



N9LYA TECHNICAL GUIDES

Figure 2: How machine learning learns

A Concrete Example

Predict house price from square footage. Start with:

$$\text{price} = w \times \text{footage} + b$$

Two parameters, w and b , initialized randomly. The model is terrible.

Loss measures wrongness. For regression, mean squared error:

$$\text{loss} = \text{average}((\text{predicted} - \text{actual})^2)$$

The gradient tells you, for each parameter, which direction increases the loss. Move the opposite way:

$$w = w - (\text{learning_rate} \times \text{gradient_w})$$

$$b = b - (\text{learning_rate} \times \text{gradient_b})$$

Repeat. The parameters converge toward values that fit the data.

That is gradient descent, and it is the entire training mechanism — for two parameters or for a trillion. The only difference at scale is that computing the gradients requires the chain rule applied through many layers, which is what backpropagation does.

Learning Rate

How large a step to take.

Too large and you overshoot the minimum, bouncing around or diverging entirely. **Too small** and training takes forever or stalls in a poor spot.

In practice it's scheduled — larger early, decaying later, often with a warmup period at the start. Getting this right is a substantial part of the practical craft of training.

Overfitting

The central problem in machine learning.

A model with enough capacity can memorize the training data perfectly and perform terribly on anything new. It learned the examples rather than the pattern.

The diagnosis: training loss keeps falling while validation loss starts rising. That divergence is the signature.

Which is why you split your data:

- **Training set** — what the model learns from
- **Validation set** — used to tune and to detect overfitting
- **Test set** — touched once, at the end, to estimate real performance

Never tune against your test set. The moment you make decisions based on test performance, it has become a validation set and your performance estimate is optimistic.

Countermeasures: more data, a smaller model, regularization (penalizing large parameters), dropout (randomly disabling neurons during training so the network can't rely on any one path), early stopping, and data augmentation.

The opposite failure, underfitting, means the model lacks the capacity to capture the pattern. High error on both training and validation data.

Features and Representation

Classical machine learning required **feature engineering** — humans deciding what aspects of the data mattered and computing them. Predicting house prices meant deciding to include square footage, bedroom count, school district, and age, and encoding each sensibly.

This was most of the work, and it was where domain expertise entered.

Deep learning's key advantage is learned representation. Given enough data, the network discovers useful features itself. A vision model's early layers learn edges, middle layers learn shapes, later layers learn objects — nobody specified that hierarchy.

This is why deep learning displaced classical methods on images, audio, and text, where good features are hard to specify by hand, and why classical methods remain competitive on tabular data, where the features are already meaningful columns.

Generalization

The whole point. A model that performs well only on data it has seen is useless.

What helps generalization: more and more diverse training data, appropriate model capacity, regularization, and — most importantly — **training data that actually resembles what you'll encounter in production.**

That last point causes more real-world failures than anything technical. A model trained on data from one hospital, one production line, or one season frequently fails on another, and the failure isn't a bug in the model. It's **distribution shift**: the world it's deployed into differs from the world it learned.

Monitor for it. A model quietly degrading because its input distribution drifted is one of the most common and least noticed production failures.

Run Check

1. Explain gradient descent in your own words without using the word “gradient.”
2. Given a model with 99% training accuracy and 62% validation accuracy — what's happening, and name three fixes.
3. For a prediction task you care about, list the features you'd engineer by hand.
4. Identify a way the data you'd train on might differ from the data you'd deploy against.

Chapter 3: Neural Networks

The architecture that made everything else possible. It's simpler than it's usually made to sound.

The Neuron

One artificial neuron does three things:

1. **Multiply each input by a weight** and sum them.
2. **Add a bias.**
3. **Apply a nonlinear activation function.**

$\text{output} = \text{activation}(x_1 \times w_1 + x_2 \times w_2 + \dots + x_n \times w_n) + b$

That's it. The weights and bias are the learned parameters.

Why Nonlinearity Is Essential

This is the part people skip and shouldn't.

Stack linear operations and you get a linear operation. A hundred layers of pure matrix multiplication collapses mathematically into a single matrix multiplication. **Depth would buy you nothing at all.**

The nonlinear activation function between layers is what makes depth meaningful. It's the reason a deep network can represent something a shallow one can't.

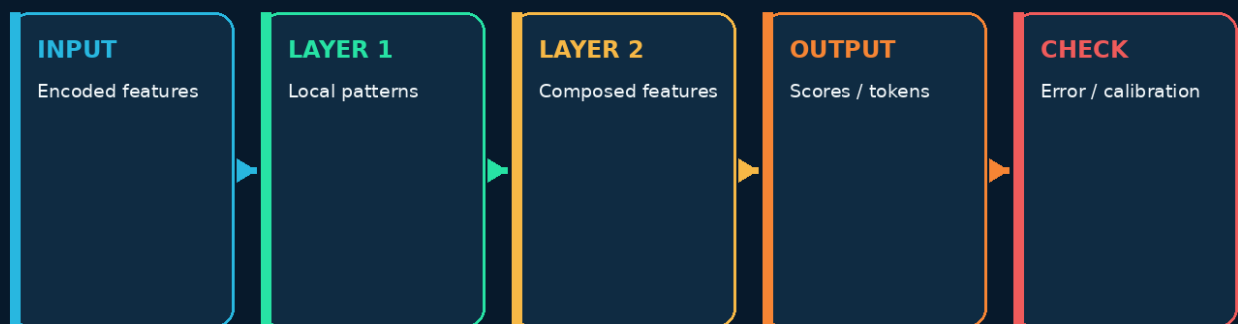
Common activations:

ReLU — output the input if positive, zero otherwise. Simple, fast, and it dominated for years. Its failure mode is “dying ReLU,” where a neuron outputs zero for all inputs and stops learning.

GELU, SiLU/Swish — smooth variants, standard in modern transformers.

A Neural Network Is Function Composition

Layers transform representations; depth alone does not create understanding



N9LYA TECHNICAL GUIDES

Figure 3: Neural networks as function composition

Sigmoid — squashes to 0-1. Used for binary output probability; problematic in hidden layers because gradients vanish at the extremes.

Softmax — converts a vector of numbers into a probability distribution summing to one. **This is what produces the token probabilities at a language model's output.**

Layers

Input layer — the data, encoded as numbers.

Hidden layers — where representation happens. Depth is the number of these.

Output layer — sized to the task. One number for regression, one per class for classification, one per vocabulary token for a language model.

A fully connected (dense) layer connects every input to every output. A layer with 1,000 inputs and 1,000 outputs has a million weights, which is why parameter counts grow quickly.

Backpropagation

How the gradients get computed through many layers.

Forward pass: data flows through, producing an output and a loss.

Backward pass: the chain rule from calculus propagates the error backward, computing each parameter's contribution to the loss.

Update: adjust every parameter against its gradient.

The insight that makes it tractable is that computing gradients layer by layer backward reuses intermediate results, making the cost roughly the same as the forward pass rather than exponential in depth. Without that, training deep networks would be computationally impossible.

Why Deep Networks Were Hard

For decades, deep networks trained poorly. The problems and their fixes are worth knowing because the fixes are in every modern architecture:

Vanishing gradients — gradients shrink as they propagate backward through many layers, so early layers barely learn. Fixed by better activations (ReLU family), normalization, and residual connections.

Exploding gradients — the opposite. Fixed by gradient clipping and careful initialization.

Residual connections — adding a layer's input to its output, so gradients have a direct path backward. **This single idea is what made networks of hundreds of layers trainable**, and it's in every transformer.

Normalization — rescaling activations between layers to keep them in a useful range. Layer normalization is standard in transformers.

Better initialization — starting weights at scales appropriate to layer size, rather than arbitrarily.

Architectures

Feedforward / MLP — the basic stack. Still used as components inside larger architectures.

Convolutional (CNN) — applies learned filters across spatial positions, sharing weights. Efficient for images because a feature detector useful in one location is useful in another. Chapter 21.

Recurrent (RNN, LSTM, GRU) — processes sequences one element at a time, carrying hidden state. Dominant for text before transformers. Their weakness: sequential processing can't be parallelized, and long-range dependencies degrade.

Transformer — processes a whole sequence in parallel using attention. Chapter 5, and it superseded RNNs for essentially everything sequential.

Mixture of Experts (MoE) — many specialized sub-networks with a router selecting a few per input. **This is how the largest current models achieve enormous parameter counts at manageable inference cost**, and it matters practically: a model with 700 billion total parameters but 40 billion active per token costs roughly what a 40-billion model costs to compute — while still requiring memory for all 700 billion.

That distinction between **total** and **active** parameters is one of the most practically important things to understand about current models, and Chapter 17 returns to it.

Run Check

1. Explain why a network without nonlinear activations gains nothing from depth.
2. Compute the parameter count of a network with layers of 784, 128, 64, and 10 neurons.
3. Find a model card for an MoE model. What are its total and active parameter counts?
4. Draw the forward and backward passes for a two-layer network.

Chapter 4: Training

Where the capability comes from, and where the cost is.

The Data

The single largest determinant of model quality.

Scale. Modern language models train on trillions of tokens — a substantial fraction of the accessible written internet, plus books, code, and curated collections.

Quality matters more than raw quantity past a point. Deduplication, filtering out low-quality and machine-generated text, and careful curation produce better models than simply adding more. Several notably capable small models got that way largely through data curation.

Composition shapes capability. More code in training improves reasoning on non-code tasks — a genuinely surprising finding. Multilingual data affects far more than just multilingual ability.

And data determines the flaws. A model trained on internet text absorbs the internet’s biases, errors, and blind spots. **This is not a bug that can be patched out later**; it’s a property of what was learned. Mitigation happens through curation and post-training, and neither fully removes it.

The provenance question is real and unsettled. Training data licensing, copyright, and consent are subjects of active litigation and legislation. If you’re building something commercially, this is a question for a lawyer, not a book.

Pretraining

The expensive part. A model learns general structure by predicting the next token across an enormous corpus.

Self-supervised, so no human labeling is needed — the text is its own label.

The cost is substantial: many thousands of GPUs running for weeks to months, and budgets in the millions to hundreds of millions of dollars for frontier models.

You will almost certainly never do this. Pretraining is done by well-resourced labs. What you’ll do is use the result, or fine-tune it.

What emerges is a model with broad knowledge and capability, and no particular inclination to be helpful. A raw pretrained model given a question will often continue it as though it were part of a document, because that’s what it learned to do. Making it useful requires post-training.

End of the free sample

Artificial Intelligence — Second Edition · by Jerome M. Kutche

That was the first 12 of 73 pages. The full book continues with the practical detail, worked examples and recovery steps that don't fit in a sample.

Get the full book (\$8.99, PDF + EPUB):

<https://booksrus.biz/product/artificial-intelligence/>

Also on Gumroad:

<https://n9lya4.gumroad.com/l/fhfqxb>



All 17 Second Edition guides in one bundle for \$89.99:

booksrus.biz/product/n9lya-technical-guides-complete-bundle

This sample may be shared freely in unmodified form. © Jerome M. Kutche. Second Edition.