IISc Centre for Continuing Education

Agentic AI Coding

From Specification to Deployment

Course Reader

Prof. Deepak Subramani
Department of Computational and Data Sciences
Indian Institute of Science, Bengaluru

Sept–Oct 2026 Cohort

Draft v0.3 | September 2026

Preface

This reader accompanies *Agentic AI Coding: From Specification to Deployment*, a six-weekend applied course run by IISc's Centre for Continuing Education. It requires no previous AI/ML experience—just a real interest in building software—and it develops a single point with progressively more detail: by 2026, coding agents have moved the scarce, high-value skill from writing code to accurately defining the desired outcomes. The overall goal is to enable you to actually build an application in your domain of interest by the end of the course.

The reader is organised in four parts.

- **Part I – Foundations** (Chapters 1–2) gives a fast, non-technical grounding in what AI and agents actually are, so that everything after this point can build on shared vocabulary rather than hype. We establish the difference between rule-based and data-driven AI, Predictive versus Generative AI, foundation models and the language/LLM/LMM distinction, and the perceive-reason-act-verify-iterate loop that turns a model into an agent.
- **Part II – Agentic Systems and Specification-Driven Development** (Chapters 3–4) is the conceptual core of the course. It's a deep exploration of what a coding harness is and the four control surfaces an engineer can truly shape—prompt, context, loop, and graph—why specification-driven development (SDD) outperforms "vibe coding" for anything intended to last, and, via five worked examples covering a full-stack web app, an AI-native app, a multi-agent review loop, a cross-disciplinary CAD copilot, and an already-deployed café ordering agent, Brews and Bytes, what it looks like in practice to take a single idea from a written spec all the way to a deployed, working system—or, in the case of Brews and Bytes, to determine after the fact whether it actually works.
- **Part III – Building, Testing and Deploying** (Chapters 5–6) follows the Trip Planner, and your own self-project, through the same spec-to-deployment arc at full scale: front-end and back-end specs and the API contract between them, converting acceptance criteria into a real test suite, CI/CD, and cloud deployment.
- **Part IV – Evaluation and System Design** (Chapters 7–8) steps back to ask how to judge agentic work rigorously, both the artifact and the process that produced it, how the classic software development lifecycle changes when an agent is writing most of the code, and finally how the same specification-first discipline scales from one application to a multi-component system.

How to use this reader. Read each chapter once before its corresponding weekend; the MAPS TO WEEKEND n banner at the top of every chapter tells you which. Attempt every CHECKPOINT yourself before reading its answer in that chapter's Checkpoint Answers section, the value is in attempting it, not in reading the solution, and work through the five End-of-Chapter Exercises in the same way. Appendix A is a pre-course checklist worth finishing before Weekend 1, since some of its items take a day or two to activate. Appendix B is a fill-in-the-blank spec template you will reuse for your own self-project from Weekend 3 onward, and the Glossary collects

every bolded term introduced across all eight chapters in one place. A companion document, the *Trip Planner Lab Guide*, gives step-by-step, copy-paste-ready instructions for building the Trip Planner alongside Chapters 5–7, with your coding agent open beside it; use it as a hands-on complement to this reader's prose, not a replacement for reading the chapters themselves. Part II's other four worked examples, the Support Bot, the CAD Copilot, the Review Loop, and Brews and Bytes, each has its own companion guide as well, carrying that example through the same spec-to-deployment arc (and, for Brews and Bytes, through evaluation) that Chapters 5–7 carry the Trip Planner through; Section 8.5 says more about which one to reach for.

Contents

Part I

Foundations: AI and Agents

1

What Is Artificial Intelligence?

Maps to Weekend 1 — What is Agentic AI?

Artificial Intelligence (AI) has changed, within a single decade, from a specialized research topic to something that shapes how students write essays, how doctors read scans, how banks detect fraud, and how companies talk to their customers. This rapid diffusion has a side effect: the word *AI* is now used so loosely that it has become almost meaningless on its own. A calculator, a spam filter, a chess engine, a chatbot, and a self-driving car are all, at different times, called "AI," yet the underlying technology, capability, and risk profile of each are completely different. Before this reader can responsibly talk about *agentic* AI, it needs a precise, shared vocabulary for AI itself.

1.1 A Working Definition

Before we define *Artificial* Intelligence, it helps to clarify the term it draws on: intelligence. **Intelligence** is often described as *the ability to acquire and apply knowledge*. This description contains two distinct components: *acquiring* knowledge (learning something that was not known before) and *applying* it (using existing knowledge to respond to a new situation). Treating these components separately lets us differentiate the two ways an artificial system may exhibit intelligent behavior.

DEFINITION: Artificial Intelligence

Artificial Intelligence (AI) is the ability of a digital computer, or a computer-controlled robot, to perform tasks commonly associated with intelligent beings: that is, to acquire and apply knowledge in the course of performing a task.

DEFINITION: ANI, AGI, and ASI: The Breadth Spectrum

Artificial Narrow Intelligence (ANI) is AI that performs a specific task, or a bounded range of tasks, without the flexibility to transfer to genuinely novel task types; every AI system that exists today, including large, multi-task systems such as modern chatbots and coding agents, is ANI.
Artificial General Intelligence (AGI), the long-term and still unrealized goal of AI research, is a hypothetical system with the flexibility to perform *any* intellectual task a human can.
Artificial Superintelligence (ASI) would go a step further still, exceeding human capability across essentially every domain. There is active disagreement about how far away AGI is, and even about whether current methods can reach it at all.

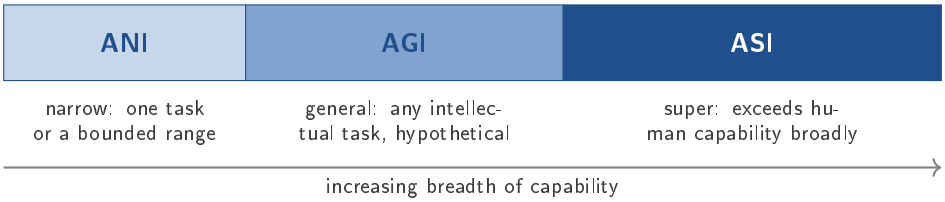

Figure 1.1: The breadth spectrum from ANI to AGI to ASI. Every AI system that exists today, including this course's coding agents, is ANI.

AI is defined by the *task* a system performs, not by the *mechanism* that produces its behavior, and that mechanism can take two very different forms.

DEFINITION: Rule-Based AI

Rule-based AI (also called **symbolic AI**) *applies* knowledge that a human expert has already acquired and hand-encoded as logic (if-then rules, decision trees, search procedures), but it never itself *acquires* new knowledge from data. A hand-coded expert system, a classical chess engine searching a game tree with hand-tuned heuristics, and a robot arm following a hand-derived control law are all rule-based AI: each performs a task "commonly associated with intelligent beings," and none of them ever adjusts its own logic in response to data.

DEFINITION: Data-Driven AI

Data-driven AI both *acquires* and *applies* knowledge: a training algorithm first **acquires** knowledge by adjusting internal, adjustable numbers (**parameters**) so that the system's outputs match a large dataset of examples as closely as possible, and the trained system then *applies* that acquired knowledge every time it produces a prediction, rather than a human writing the logic directly. **Machine Learning (ML)** is this data-driven branch of AI.

Neither definition addresses how autonomous or adaptive the system it produces is. A rule-based chess program can be just as autonomous—and just as difficult to construct—as a learned one. The rule-based vs. data-driven distinction is about *mechanism*: where the behavior originates, and it is independent of how autonomous that behavior ends up being. Keep this difference in mind: the large language models powering this course's coding agents are data-driven AI, but the harnesses that enclose them (Chapter 3) layer substantial hand-written, rule-based control flow on top of that learned core.

1.2 A Brief History of AI: Booms and Winters

Hype cycles in AI are not new. The field has been through this before, more than once, and that history is worth knowing before this reader asks you, in the next section, to be skeptical of AI capability claims.

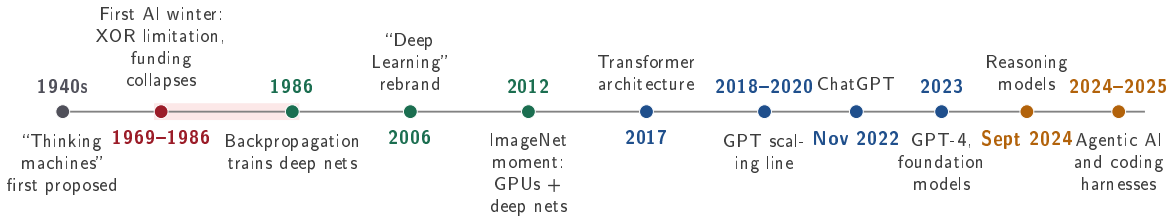

Figure 1.2: Booms and winters in AI, 1940s–2025: grey marks the field's earliest ambitions, red the first winter, green the deep-learning breakthroughs that ended it, blue the large-language-model era, and amber the agentic era this reader picks up.

- **1940s.** Early theorists first articulate the idea of a "thinking machine": a device that could reason, not merely calculate.

- **1969–1986: the first AI winter.** Early neural networks could only learn a *linear* decision boundary, which provably cannot represent even a simple nonlinear function like XOR. This limitation, combined with a string of overpromised and underdelivered projects, sharply cut research funding for two decades.

- **1986: backpropagation.** Researchers popularize an algorithm for training *multi-layer* networks with nonlinear activation functions, finally overcoming the XOR-style limitation.

- **2006: the "Deep Learning" rebrand.** Hinton, LeCun, and Bengio show that large neural networks can be pretrained layer by layer, reviving interest in the field under a new name: **Deep Learning**.

- **2012: the modern era begins.** A deep neural network trained on the large, carefully labeled ImageNet dataset cuts the field's best image-recognition error rate from 26% to 16% in a single year, a result usually credited to three things arriving together: faster parallel computing hardware (GPUs), improved training algorithms, and better-behaved activation functions and regularization techniques.

- **2017: the Transformer.** Google researchers publish "Attention Is All You Need," introducing the **Transformer** architecture, which processes an entire sequence at once using an attention mechanism instead of reading it one step at a time. It becomes the architecture behind essentially every major language model that follows.

- **2018–2020: the GPT scaling line.** OpenAI applies the Transformer to language modeling and finds that simply making the same architecture bigger and training it on more text predictably makes it more capable: GPT-1 (2018, 117 million parameters), GPT-2 (2019, 1.5 billion parameters), and GPT-3 (2020, 175 billion parameters) trace out what later becomes known as a **scaling law**.

- **November 2022: ChatGPT.** OpenAI wraps a GPT-3.5-class model in a free, conversational chat interface. It reaches 100 million users within two months and brings large language models into mainstream daily use almost overnight.

- **2023: GPT-4 and the foundation-model era.** GPT-4 launches alongside a wave of competing foundation models (Anthropic's Claude, Google's Gemini, Meta's open-weight Llama, and others), each usable across a huge range of tasks through prompting alone.

- **September 2024: reasoning models.** OpenAI releases o1, the first widely used model trained to spend extra computation *thinking before answering*, measurably improving performance on hard math, coding, and science problems.

- **2024–2025: agentic AI and agent harnesses.** Models are wrapped in an iterative loop, a **harness**, that lets them plan a multi-step task, call external tools, observe the results, and decide what to do next, largely without a human approving each step. Cognition's Devin (March 2024) and Anthropic's Claude Code (February 2025) are early examples applied to software engineering. This is where **Agentic AI** begins, and it is where the rest of this reader picks up the story.

IMPORTANT: Why This History Matters

Every one of AI's booms was followed by a winter, driven by a gap between what was promised and what the technology could actually deliver at the time. That history is a large part of why the next section insists on skepticism: it is not cynicism about AI, it is a lesson learned the hard way, more than once.

1.3 The AI "Snake Oil" Problem

Because labeling a product "AI-powered" reliably increases sales, vendors are motivated to apply the label whether or not it is technically accurate, an incentive sometimes called **AI hype**. Narayanan and Kapoor's *AI Snake Oil*[1] proposes three diagnostic questions to place any system precisely on a spectrum of genuine capability and autonomy:

Q1. Does the task require effort or training for a human to perform? Sorting a list of ten numbers requires no training; recognizing a malignant tumor in an X-ray does. Tasks that are effortless for humans are often surprisingly hard to automate, while tasks that feel hard for humans are often trivial to automate — **Moravec's paradox**.

Q2. Was the system's behavior directly specified in code, or did it emerge from learning? This is the rule-based/data-driven question of Section 1.1 again, asked about one specific product rather than about AI in the abstract.

Q3. Can the system make autonomous decisions with some flexibility to its environment? A system that always does exactly the same thing regardless of context is closer to fixed automation than to the adaptive systems this reader is about.

IMPORTANT: Read Critically

Before accepting a claim about an AI system's capability, always ask three things: does the system actually work, as measured on data it has never seen; is it even AI by the diagnostic questions above, or a simple rule-based script wearing an AI label; and what, precisely, is being claimed, with evidence that is public and reproducible.

This habit will serve you well throughout the course: an agent that passes a demo is not the same claim as an agent that reliably meets a written specification, and Chapter 4 builds an entire discipline around making that difference checkable rather than a matter of impression.

1.4 From Rules to Learning: The Machine Learning Mental Model

Before this reader moves on to generative and agentic systems, it helps to have one simple mental picture of what machine learning is trying to do: in the real world, some data can be collected, and some quantity must be predicted in order to create value. Machine learning replaces the unknown mapping from "real world" with a machine that produces a prediction directly from the collected data.

DEFINITION: Model

A **model** is a mathematical function f, with adjustable numbers called **parameters**, that maps an input to an output. **Training** (or **fitting**) a model means using data to choose values for those parameters so that f's predictions match reality as closely as possible.

[1] Arvind Narayanan and Sayash Kapoor, *AI Snake Oil: What Artificial Intelligence Can Do, What It Can't, and How to Tell the Difference* (Princeton University Press, 2024).

This deceptively simple picture, *data in, prediction out*, underlies everything from a two-parameter straight-line fit to a trillion-parameter language model. What changes as the models become more sophisticated is the *richness of* f, not the fundamental idea.

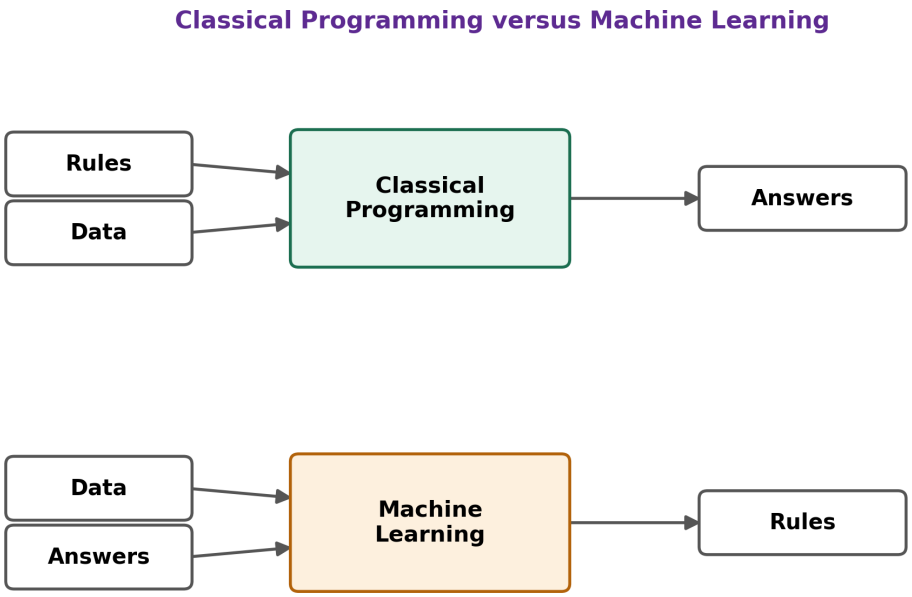

Figure 1.3: Classical programming (top): rules and data go in, and the program computes answers. Machine learning (bottom): data and answers (examples) go in, and training produces the rules — exactly the inversion the Celsius-to-Fahrenheit example below demonstrates.

EXAMPLE 1.1: Traditional Programming versus Machine Learning

Suppose we want a program that converts a temperature from Celsius to Fahrenheit. **Traditional programming** is immediate: we already know the physical relationship $F = \frac{9}{5}C+32$, so we write this formula into the code. Input: C. Logic: the equation, hand-specified by a human. Output: F. There is no data, no training, and no uncertainty — a rule-based system by Section 1.1's definitions, and most people would not call it "AI" at all.

Now consider a different question: can a machine accept Celsius as input and produce Fahrenheit as output *without ever being given the rule*, only example pairs of (C, F) readings? Given six paired readings and no formula, fitting a straight line $\hat{F} = wC + b$ by ordinary least squares recovers $w \approx 1.8$ and $b \approx 32$ — the exact physical rule, discovered purely from data. This is the essential insight of machine learning: *given enough representative data, a sufficiently expressive model can discover an underlying rule without ever being told the rule explicitly.* Where traditional programming required a human to already know and encode the physics, machine learning only required example measurements.

CHECKPOINT 1.1

A spreadsheet macro sorts a column of numbers in ascending order when a button is clicked. Walk through Q1–Q3 above for this macro. Would you call it AI? Now repeat for a plugin that predicts next month's sales by fitting a trend line to two years of data. How do your answers change, and why?

1.5 Predictive AI versus Generative AI

Machine learning systems are divided into two families by the shape of things they produce.

DEFINITION: Predictive AI vs. Generative AI, Formally

Let x denote an input of any data modality. A **Predictive AI** model computes $f(x) \in \mathbb{R}^n$, where n (the number of classes, or the number of continuous targets) is *fixed at training time*. A **Generative AI** model computes $G(x, z) \in \mathcal{S}$, where $\mathcal{S}$ is a space of sequences, images, or signals whose length or resolution is *not* fixed at training time, and z is a source of randomness that lets the same x produce different valid outputs on different calls.

A spam classifier or a house-price predictor is Predictive AI: the output shape (spam/not-spam; one price) is fixed before a single example is seen. A large language model is Generative AI: it produces a token sequence of whatever length the answer needs, one token at a time, each one sampled from a probability distribution over its vocabulary and fed back as input for the next token. This **autoregressive** mechanism is why two calls to the same chatbot with an identical prompt can produce two different answers: the randomness z in the definition above means that each generated token is a draw, not a deterministic lookup. Diffusion models, the dominant mechanism behind modern image and video generators, produce an output in the same variable-size way but by a different route: refining an entire image simultaneously across many denoising steps rather than emitting it one token at a time.

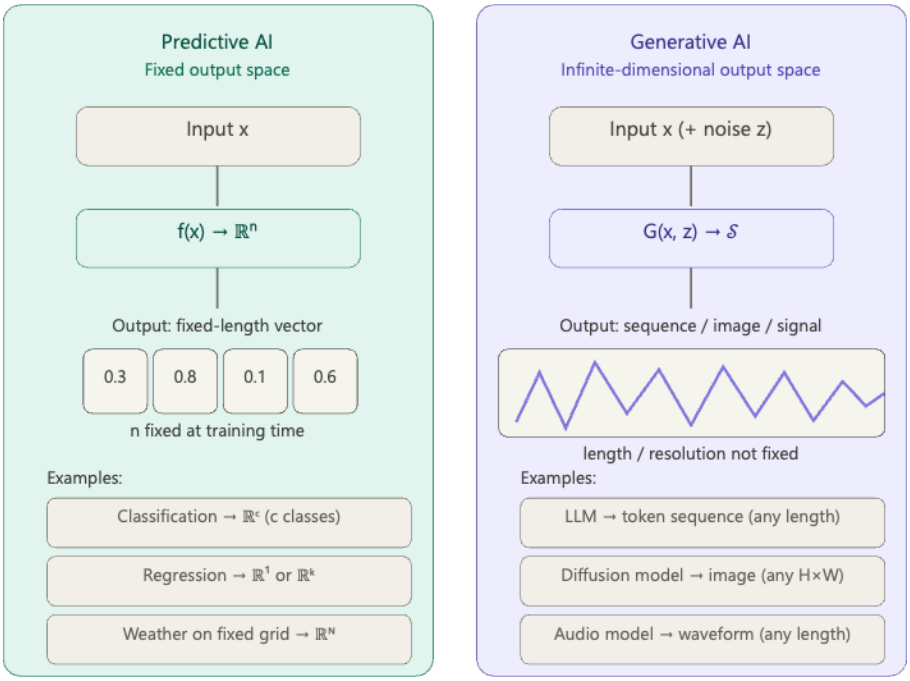

Figure 1.4: Predictive AI produces a fixed-length output vector whose dimension n is fixed at training time. Generative AI produces an output — a sequence, an image, a signal — whose length or resolution is *not* fixed at training time.

CHECKPOINT 1.2

If a commercial chatbot used *greedy* decoding (always emitting the single most likely next token, with no randomness) instead of sampling, would two calls with the identical prompt still be guaranteed to produce the same answer? What does this imply about why "the model said something different this time" is not, by itself, evidence of a bug?

1.6 Foundation Models

Predictive AI typically trains a separate model from scratch for each new task. Generative AI works differently: a single, very large model is trained once and then reused across many tasks.

DEFINITION: Foundation Model

A **foundation model** is a large-scale AI model trained on a vast, diverse dataset, intended to serve as a *base* for many downstream tasks and applications, rather than being built for one narrow purpose. Representative examples include GPT, Claude, and Gemini.

Getting to a foundation model like this takes two broad stages. **Pre-training** is where the model first learns the patterns of language and the world, by consuming an enormous amount of general text and other data; **post-training** is a later, much smaller stage, where the model is shaped into a helpful, well-behaved assistant rather than just a raw text predictor. This course stays at that level of detail throughout; the mechanics of each stage are out of scope.

Foundation models generalize to new tasks with minimal additional training, through one (or a combination) of three adaptation mechanisms, in increasing order of cost:

- **Prompt engineering**: carefully wording the instructions given to the model at inference time, with no change to the model's parameters at all. Cheapest, fastest, but limited by what the model already "knows."
- **Retrieval-Augmented Generation (RAG)**: attaching an external, up-to-date knowledge base that the model consults at inference time. Good for proprietary or rapidly changing data, without retraining the model.
- **Fine-tuning**: continuing to train (some or all of) the model's parameters on a smaller, task-specific dataset. Most expensive, but can change the model's underlying behavior, not just what it has access to.

This same three-way choice reappears, at the level of a whole coding agent's context, in Chapter 4's treatment of what a spec, a project's memory file, and a codebase's own conventions each contribute to what an agent "knows" about a task, and it is the exact adaptation decision behind Chapter 4's support-bot example, which is built specifically because prompting alone is not enough to keep its answers current.

CHECKPOINT 1.3

A hospital wants an AI assistant that answers clinicians' questions using *this week's* patient records, which cannot have been in any foundation model's training data and must never be permanently memorized into the model's weights for privacy reasons. Which of prompt engineering, RAG, or fine-tuning is appropriate here, and why do the other two fail the requirement?

1.7 Language Models, Large Language Models, and Large Multimodal Models

Section 1.6 named GPT, Claude, and Gemini as foundation models without saying what kind of model each one actually is inside.

DEFINITION: Language Model

A **language model** is a model, in Section 1.4's sense, trained specifically to work with language: given some text, it estimates what word or phrase is likely to come next. The predictive text on an ordinary phone keyboard, suggesting the next word as you type, is already a small, narrow language model. Nothing about the definition says how big the model has to be.

DEFINITION: Large Language Model (LLM)

A **Large Language Model (LLM)** is a language model built at a very large scale: trained on an enormous slice of the text available in the world (books, articles, code, conversation), and containing many billions of the adjustable parameters that Section 1.4 introduced. That scale is what turns "predict the next word" into something that can hold a fluent conversation, write an essay, or generate working code. GPT, Claude, Gemini, and Llama, all named as foundation models in Section 1.6, are LLMs at their core.

DEFINITION: Large Multimodal Model (LMM)

A **Large Multimodal Model (LMM)** extends an LLM so that it can take in, and sometimes produce, more than just text: images, audio, and video, alongside language, inside the same model. (*Multi*, more than one; *modal*, mode or type of data.) Claude and GPT can look at an uploaded photograph or a page of a PDF and describe or reason about it; Gemini is built from the ground up to handle text, images, audio, and video together; tools such as DALL-E and Midjourney take a text description and produce an image. Most of today's leading foundation models are, more precisely, LMMs rather than plain LLMs.

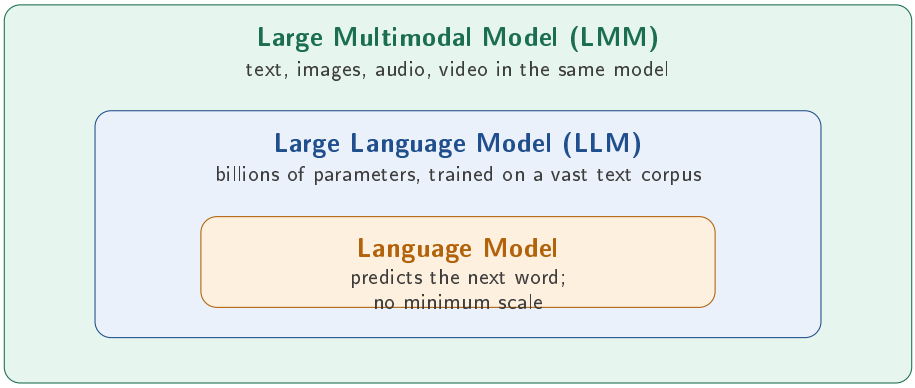

Figure 1.5: Language Model, LLM, and LMM as increasing scale and breadth, not strict containment: a phone keyboard's next-word predictor is a language model far too small to be an LLM.

EXAMPLE 1.2: Telling the Three Apart

A 2015-era phone keyboard suggesting "you" after you type "how are" is a *language model*: narrow, small, and trained on comparatively little text. Asking Claude to summarize a fifty-page report is using it as an *LLM*: the input and output are both text, but the model draws on an enormous training corpus to produce a fluent, coherent summary. Uploading a photo of a hand-drawn app sketch and asking Claude to write the matching webpage code is using it as an *LMM*: the input is an image, the output is text (code), and both are handled inside one model.

This course's toolchain leans on LMMs directly: Claude Sonnet and Opus, the models Chapter 3's coding harnesses run on, can read a screenshot of a broken user interface alongside the code that produced it, not just a text description of the bug.

CHECKPOINT 1.4

Your phone's keyboard suggests the next word as you type. Separately, you show Claude a photograph of a whiteboard sketch and ask it to write matching React code. Classify each system as a language model, an LLM, or an LMM, and justify each answer in one sentence.

1.8 From Chat to Action: Where Agentic AI Begins

Up to this point, we've been talking about a model that responds to a question one time and then ends: prompt in, prediction or generated text out. That framing doesn't account for how a system could inspect a codebase, edit or create a file, execute a test, and then choose its next step autonomously—which is exactly the focus of this course. Put simply, a **chatbot** responds to you; an **agentic system** takes actions on your behalf.

DEFINITION: Agentic AI

An **AI agent** is a system that can independently pursue a goal, make decisions, and take actions within its domain, rather than simply responding once to a single prompt. **Agentic AI** is the broader paradigm of building systems out of such agents.

Three properties distinguish an agent from a plain chatbot, and Chapter 2 develops each one in full: **state awareness** (the agent tracks what has already happened in the current task, rather than treating every message in isolation), **reactive *and* proactive behavior** (it responds to new information but can also act on its own initiative, such as asking a clarifying question before guessing), and **access to external tools** (it can call APIs, query databases, run code, or invoke other agents, rather than being limited to what is stored in its own weights).

Every idea this chapter introduced, a model as a trained function, the autoregressive loop that lets a language model generate text of any length, and RAG as a way to ground a model in knowledge beyond its training data, reappears inside the agent loop the next chapter builds. Chapter 2 takes that single-call picture and asks: what has to be added to it before a model can act?

1.9 Checkpoint Answers

Try each checkpoint yourself before reading its answer here; the value of a checkpoint is in attempting it, not in reading the solution.

Checkpoint 1.1 (spreadsheet macro versus sales-trend plugin). The sorting macro fails all three diagnostic questions in the sense that would make it interesting AI: sorting numbers takes no human training (Q1), its behavior is directly coded as a fixed procedure rather than learned (Q2), and it does exactly the same deterministic operation every time regardless of context (Q3). Most people would not call it AI at all, and by Section 1.1's mechanism split it is, at most, a trivial rule-based system. The sales-trend plugin is different: forecasting next month's sales is a skill that takes a human real effort to do well (Q1), its behavior comes from fitting a model to historical data rather than from a hand-written formula (Q2), and it adapts its output to whatever data it is given (Q3). It is data-driven AI, specifically a simple Predictive AI model, in exactly the sense Section 1.5 formalizes.

Checkpoint 1.2 (greedy decoding and repeated prompts). Yes: with greedy decoding, the model always selects the single highest-probability next token, with no random draw anywhere in the process, so an identical prompt run twice (with no other change to conversation state) produces an identical output every time. This means that when a production chatbot gives two different answers to the same prompt, the explanation is ordinary sampling randomness, the z term in Section 1.5's formal definition, not a malfunction. A difference in *wording* between two answers is expected; a difference in *substance* or correctness is a separate question worth checking on its own merits.

Checkpoint 1.3 (hospital assistant on this week's records). Retrieval-Augmented Generation is the right mechanism. It lets the assistant look up this week's patient records at the moment it answers a question, without those records ever being written into the model's parameters, satisfying both requirements at once. Prompt engineering alone cannot supply

information the model was never given access to in the first place; the current week's records still have to reach the model somehow, which is exactly the retrieval step RAG adds. Fine-tuning fails outright on the privacy requirement, since fine-tuning means continuing to train the model's own parameters on the supplied data, which is precisely the kind of permanent memorization the hospital cannot allow.

Checkpoint 1.4 (phone keyboard versus Claude reading a sketch). The phone keyboard is a *language model*, but a small, narrow one: it predicts a single next word from limited context, using far fewer parameters and a far smaller training set than a modern LLM. Claude reading an uploaded whiteboard photo and writing matching code is acting as a *Large Multimodal Model*: the input is an image rather than text, and it is the model's scale, both in parameters and in the breadth of data (text, images, and code together) it was trained on, that lets it interpret a rough sketch and produce fluent, working code rather than a single guessed word.

1.10 End-of-Chapter Exercises

Exercise 1.1. A company claims its "AI-powered" resume screener uses "advanced neural networks" to predict job success. On inspection, it turns out to check whether a resume contains a fixed list of keywords, with no training data involved anywhere. Using the three diagnostic questions of Section 1.3, is this system AI in the technical sense this chapter defines? Explain.

Exercise 1.2. Explain, in one or two sentences, why a classifier trained only on labeled cat and dog photos is Predictive AI, while a modern model that turns a text description into a brand-new photo is Generative AI. Use the formal distinction of Section 1.5.

Exercise 1.3. Name the AI winter described in Section 1.2, and explain in your own words what caused it.

Exercise 1.4. A startup fine-tunes a foundation model on last year's product-review dataset and ships it as "the assistant that knows everything about our products." Six months later the company launches five new products, and the assistant confidently gives wrong answers about them. Using Section 1.6's three adaptation mechanisms, explain what went wrong and how you would fix it.

Exercise 1.5. Is the large language model behind a modern coding assistant a Predictive AI system or a Generative AI system? Justify your answer using Section 1.5, and name one thing a pure text-only coding assistant could not do that Section 1.7's Large Multimodal Models can.

Answers

A1.1. No, or at best a very simple rule-based one, not what this chapter means by AI at its most interesting. By Q2, the screener's behavior was directly coded (a fixed keyword list) rather than learned from data, so it is rule-based, not machine learning, whatever the marketing copy says. Even granting that rule-based systems can be AI, the "advanced neural networks" claim is false, and this is exactly the AI-snake-oil pattern Section 1.3 warns about: a sophisticated-sounding label attached to a much simpler mechanism.

A1.2. The cat/dog classifier computes an output in a fixed-size space decided at training time, here, two categories, cat or dog, which is precisely Section 1.5's definition of Predictive AI. The text-to-image model instead produces an entire image whose resolution is not

fixed at training time, and includes a random ingredient, so the same prompt run twice can generate two different valid images, which is precisely the definition of Generative AI.

A1.3. The 1969–1986 first AI winter. Early neural networks could only learn a straight-line (linear) decision boundary, and some simple problems, such as XOR, provably cannot be solved by any straight line. Combined with a string of projects that had promised more than the technology of the time could deliver, funding for the field was sharply cut for close to two decades, until the 1986 popularization of backpropagation made training deeper, nonlinear networks practical.

A1.4. Fine-tuning bakes knowledge into the model's parameters at one point in time; it does not update itself as new products launch, so the assistant is stuck with a frozen, six-month-old picture of the catalog, and, worse, answers fluently and confidently even about products it has never seen. Retrieval-Augmented Generation is the better fit: connecting the assistant to a live, continuously updated product database means a newly launched product is available to it as soon as it is added, with no retraining required.

A1.5. Generative: it produces a token sequence (code) whose length is not fixed in advance, generated one token at a time, matching Section 1.5's formal definition exactly. A pure text-only model could not directly take a screenshot of a broken interface, or a hand-drawn wireframe, as input; that visual information would first have to be translated into text by something else, since a text-only model has no way to process images, audio, or video on its own.