

FUNDAMENTALS OF AI-AUGMENTED ENGINEERING: A CONVERSATIONAL APPROACH

Taner Onsay, Ph.D. & Sebnem Onsay
Authors
August 2026 · Revision 1.0

© 2026 AI First Engineering

Problem



Model



Compute



Analyze



Validate



Improve

*The engineering workflow you will use throughout this book
by Taner & Sebnem Onsay,*

CHAPTER - 1

PART 5 · ≈ 30 MIN

Break the AI

5

Why doesn't prediction equal reality — and how do you prove an answer wrong?

Problem

›

Model

›

Compute

›

Analyze

›

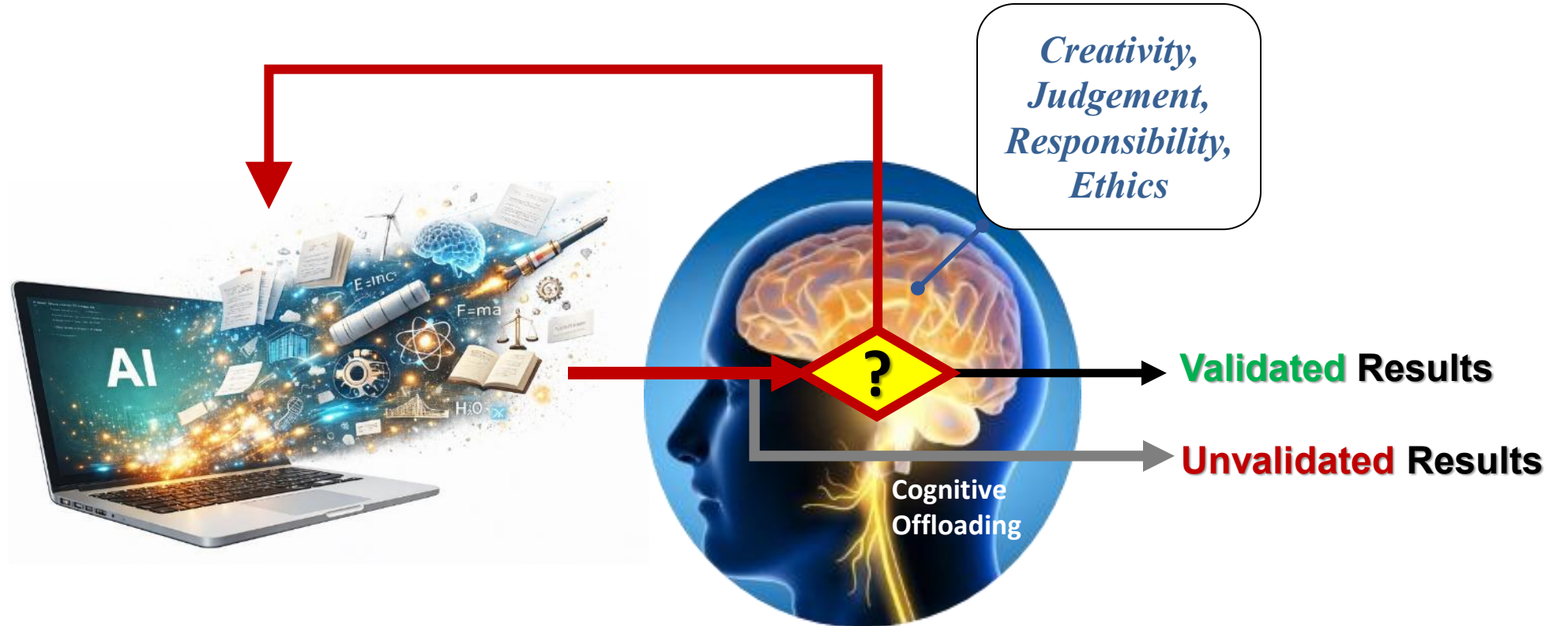
Validate

›

Improve



Critical Thinking in the AI age



***Which results would you put
your name on?***



Why doesn't prediction equal reality?

Prediction $\neq$ Reality. Why?



Physical / modeling error

The model doesn't adequately represent reality (e.g., air matters after all)



Experimental / measurement error

The measurement itself is uncertain or limited (tape, frames, reaction time)



Numerical error

Approximation, rounding, discretization, convergence — conceptual for now



Programming error

The code doesn't implement the intended algorithm (a typo that still runs)



Assumption error

An assumption is inappropriate for this situation (feather, 200 m drop)



AI error

AI supplies a wrong equation, assumption, unit, explanation, or conclusion — fluently

THE ENGINEER'S JOB, RESTATED

Anyone can notice the answer is off.
The engineer must diagnose
which kind of problem occurred.

Model? Measurement? Numerics? Code? Assumption? AI? Different diagnoses demand different fixes — and misdiagnosis wastes weeks.



A recurring course ritual

**AI-generated solutions can look polished, confident,
and completely reasonable —
while still being wrong.**

Every few weeks, we will hunt for the flaw in a plausible AI answer. Today: your first hunt.



An AI answered our exact question...

AI-generated solution

```
import math

h = 2.0      # height in meters
g = 9.81     # gravity, m/s^2

# time for the ball to reach the floor
t = math.sqrt(h / (2 * g))

print(f"The ball lands after {t:.3f} s")

The ball lands after 0.319 s
```

- ✓ Runs with no errors
- ✓ Clean, commented, professional
- ✓ Plausible number (under a second)
- ✓ Units check out — it is a time
- ✓ Confident, fluent explanation

“This uses the standard free-fall relationship to compute the landing time.” — the AI



Is this correct?

How could you prove it is wrong — without trusting us, the AI, or your gut?

DISCUSS

Work in pairs. Produce at least two independent ways to test it.



Your validation toolkit — six independent tests



Derive it

Start from $h = \frac{1}{2}gt^2$ and rearrange yourself. The AI wrote $t = \sqrt{h/2g}$; physics says $t = \sqrt{2h/g}$.



Hand-calculate

$\sqrt{2 \cdot 2 / 9.81} \approx 0.64$ s. The AI's 0.319 s is exactly half — The incorrect formula produces exactly half the correct fall time. Trace the algebra to find why.



Experiment

Drop a ball from 2 m. Even by eye, it falls longer than a third of a second.



Limiting cases

Check simple/extreme inputs where you know the answer (h near 0, huge h, a known textbook case).



Independent route

A genuinely different path — not the same formula retyped in Excel, not a second AI echoing the first. Agreement is evidence, not proof.



Physical reasoning

Average speed, energy, or just experience: does the magnitude make sense?

Independent is the important word. Units would NOT catch this bug — $\sqrt{h/2g}$ and $\sqrt{2h/g}$ are both times. Two checks sharing an equation, data source, or AI reasoning can share the same error.

Running code $\neq$ Correct code

Plausible AI output $\neq$ Engineering truth

The code was flawless Python. The prose was confident. The units even checked out. The physics was wrong by a factor of two. Passing one test is not proof — only multiple independent validations caught it.

Problem

›

Model

›

Compute

›

Analyze

›

Validate

›

Improve



The Engineering Computational Notebook



Problem

What are we determining?



Model

Equations & relationships



Assumptions

What are we taking for granted?



Computation

How was it calculated?



Results

What did we obtain?



Validation

Why is it credible?



AI contribution

How did AI assist?



Human verification

What did WE check?



Improve

What next?

Not paperwork — a reusable thinking template. Notice it IS the workflow, plus an honest account of the AI/human split.

Problem



Model



Compute



Analyze



Validate



Improve



Three small activities — depth over volume



1 · Falling object, modified

Take the fall_time notebook. Change the height, predict the new time BEFORE running, run, compare. Repeat for one more height. You are editing code, not writing from scratch.



2 · Your own simple calculation

Pick one familiar physical calculation (velocity, force, energy, pressure, unit conversion...). Identify inputs · outputs · equation/model · units · assumptions. Then compute it any way you like.



3 · AI as tutor

Choose one prerequisite you're rusty on. Run the protocol: AI › Investigate › Test › (Human Help). Submit a short reflection — see panel.

ACTIVITY 3 — SUBMIT BRIEFLY

- What I asked AI
- What AI explained
- How I tested or checked it
- What I understand now that I didn't before

A few honest sentences each. No giant AI transcripts, please.



Learning doesn't stop when lecture ends



LECTURE

Understand the engineering workflow — today's 3 hours.



EXPLORE / PRACTICE

Modify the computation. Run the homework activities.



PREPARE FOR LAB

Python working · Tracker installed · one practice video tracked.



READINESS CHECK

5–7 minutes at the start of lab — recall and reason.



LAB

Predict › experiment › analyze two ways › compare › ask AI › validate.



REFLECT / EXTEND

What do we actually know — and how do we know it?

The maize cards are yours before lab day. The next three slides spell them out.



Python: Predict › Compute › Compare



Run the full-time notebook

The exact code from today. Confirm it runs on your machine.



Change the height — several cases

Try $h = 0.5, 1.0, 3.0$ m... your choice.



Predict BEFORE you press Run

Write the number down first. Every single time.



Compare prediction vs. output

Where were you right? Where surprised? Why?

PREDICT

If the drop height becomes **four times larger**, what happens to the fall time?

Commit to an answer — then test it in the notebook.

Hint: today's Analyze slide already told you the shape of the answer.

Ten minutes, not a programming assignment. The habit — predict, compute, compare — is the deliverable.



Tracker: turn motion into data — once

VIDEO

>

POSITION / TIME DATA

>

SIMPLE PLOT

- 1 Install Tracker (free — link posted)
- 2 Import one short video
- 3 Set the scale with an object of known length
- 4 Set a coordinate system
- 5 Track the object across several frames
- 6 View the t, x, y data table · plot y vs. time



Film anything simple & safe

- Gently toss a light object upward
- Roll something across a table or floor
- Move a visible object through the frame
- Work with a friend, or prop the phone securely

The quality of the experiment is NOT the objective. Video > data > plot is.

Safety: nothing heavy, no risky heights, no standing on furniture, keep people, pets, windows, and anything fragile well clear.

Week 1 Tracker goal: “I can turn a video of motion into position/time data.” Nothing more.



Four lists — leave with these, not with 50 slides to reread



TRY

- Run the fall-time notebook and change h
- The 4×-height prediction test
- Film one simple motion · track it in Tracker



INSTALL / PREPARE

- Python + VS Code + Jupyter working
- Tracker installed · one video imported
- Scale + axes attempted · one $y-t$ plot made



BRING

- Laptop (charged) — at least one per team
- Smartphone that records video (30 or 60 fps)
- Your willingness to commit predictions in writing



UNDERSTAND

- $t = \sqrt{2h/g}$ — and the assumptions behind it
- Running code $\neq$ correct code
- Why the prediction is recorded before the drop

Lab opens with a 5-minute readiness check on exactly these ideas — then the investigation starts immediately.



WHAT YOU CAN NOW BEGIN TO DO

Given a simple engineering problem...

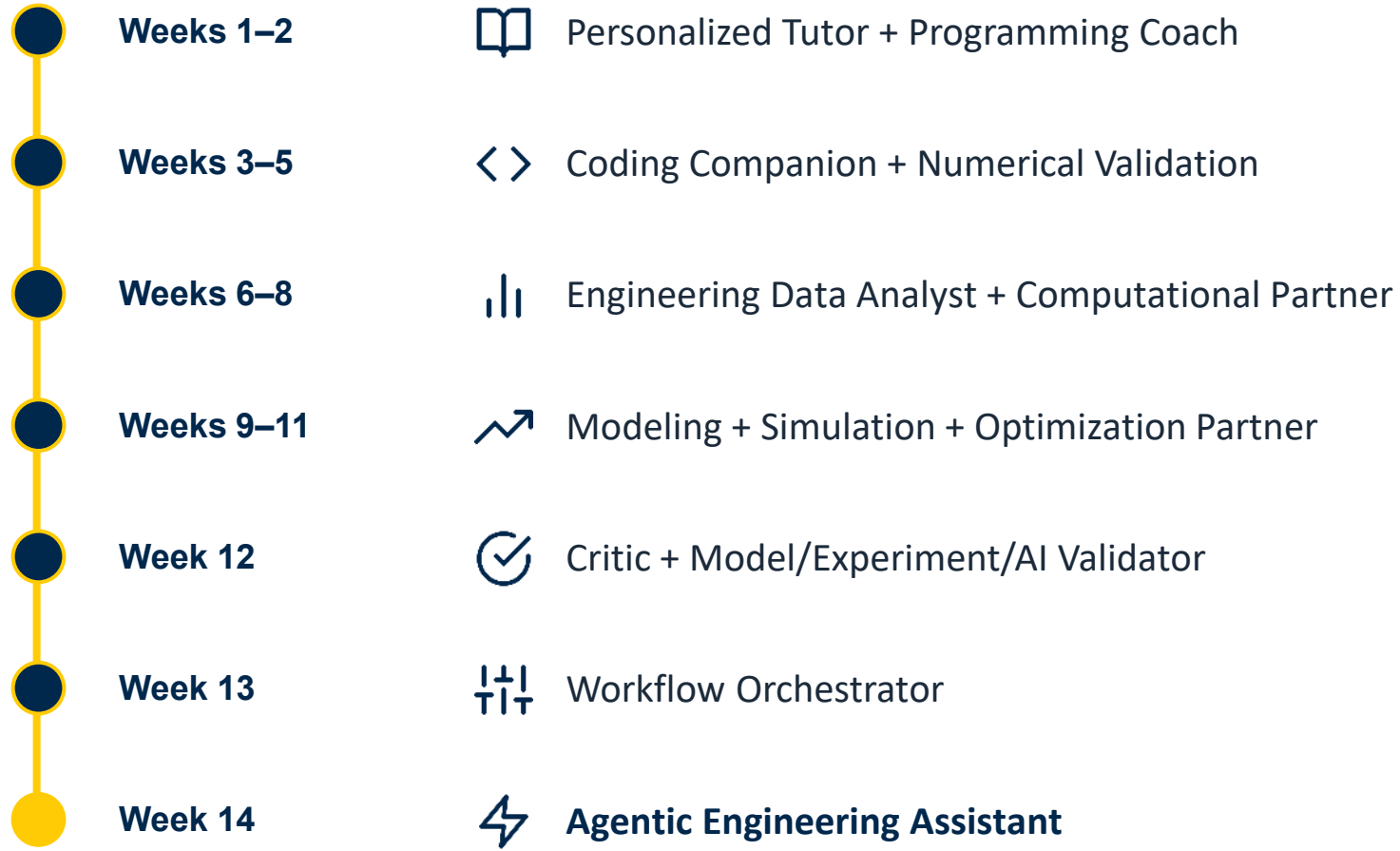
- 1 Identify the problem
- 2 Identify inputs & outputs
- 3 State assumptions
- 4 Construct / use a mathematical model
- 5 Describe an algorithm
- 6 Execute or modify a simple computation

- 7 Examine units
- 8 Ask whether the answer is physically reasonable
- 9 Compare with experiment or another method
- 10 Identify possible sources of error
- 11 Improve the solution

The goal is not to become a Python programmer in one week. It is to **begin thinking like a computational engineer.**



AI's role grows every few weeks — yours grows faster



By Week 14, you operate as the lead engineer — AI coordinates tasks; you own verification and final decisions.

SEE YOU IN LAB

Lab 1 — Can We Trust the Model, the Computer, or the AI?



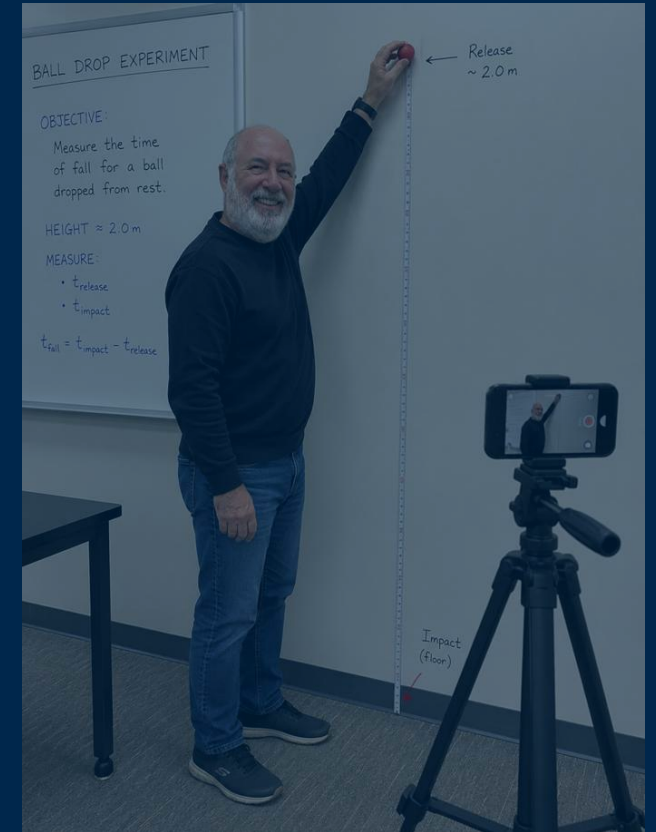
Install Python + VS Code + Jupyter + Tracker (guides posted) — before lab



Bring a phone that records video



Bring your 0.64 s prediction — we're going to test it against reality



Problem

Model

Compute

Analyze

Validate

Improve