Celium

ASAS-NANP SYMPOSIUM ON MATHEMATICAL MODELING IN NUTRITION

From Model to Product

Using AI to optimize animal nutrition modeling processes

Reid Hensen

A hands-on session · 90 minutes · No prior software experience required

BACKGROUND

Reid Hensen

Managing Partner · Celium Group

THE SHORT VERSION

M.S. Agricultural Economics
Colorado State University

Turning models into working software
A career spent translating complex financial and geospatial models into practical web applications

Four large-scale MMRV platforms
Measurement, monitoring, reporting and verification, built and shipped

Celium

Consulting and software for agricultural and conservation organizations.

SELECTED CLIENTS

- Soil Health Institute
- The Nature Conservancy
- South Dakota State University
- Dairy Management Inc.

Why I'm here: our team now uses AI to ship products faster and cheaper than we ever could before — and I'd like to put those same tools in the hands of the researchers doing the next wave of modeling work.

THE LAST-MILE PROBLEM

Great science, without programing expertise

Research teams have invested heavily in mechanistic, statistical, and simulation models that capture complex biological relationships. Yet most stay locked in spreadsheets, scripts, and publications.

Spreadsheets

Powerful logic, trapped in cells only the author can navigate

Scripts

R and Python that never leaves the modeler's laptop

Publications

Equations on a page a producer will never run

The traditional barrier

Turning a working model into a deployable product has required software development expertise most research teams simply don't have.

What changes today

LLMs and prompt engineering let you generate the scaffolding of a real tool without deep programming experience.

SPEAKING THE LANGUAGE

A short AI vocabulary

Nine words you'll hear today. None of them are as complicated as they sound.

Model

The AI itself — Claude, GPT, and so on. Careful: not your nutrition model. Same word, two meanings.

Prompt

What you type to the AI. The instruction, plus whatever material you hand it to work from.

Context

Everything the AI can see right now — your files, the conversation so far, whatever you pasted in.

Agent

An AI that acts, not just answers: reads your files, runs the code, reads the result, and tries again.

Terminal

The text window where you type commands. RStudio has one — the tab next to the Console.

Repository

A project folder tracked by Git: your code, data, and full history in one place. Often just “repo.”

Token

The chunks text gets split into. Usage limits are counted in tokens, roughly ¾ of a word each.

Hallucination

A confident, wrong answer. The reason every output gets checked against a source you trust.

CLAUDE.md

A plain-text file in your repo that tells the agent about the project before you ask anything.

Highlighted: the one that matters most today, and the two worth watching.

THE LANDSCAPE

Claude Code is one of many

as of July 2026

https://code.claude.com/docs/en/quickstart

TERMINAL AGENTS

Claude Code · Codex CLI · Antigravity CLI

Run beside whatever editor you already use, and work at the level of the whole project. This is the category we're using today.

AI-NATIVE IDEs

Cursor · Google Antigravity

The editor itself is built around the agent. Strongest when the work is visual and you want to see every change as it lands.

EDITOR PLUG-INS

GitHub Copilot

Adds an agent to the editor you already have. Broadest coverage across editors, and usually the cheapest way in.

OPEN SOURCE · BRING YOUR OWN KEY

OpenCode · Aider · Cline

Free to install — you pay only for model tokens. Choose your own model, keep control of data, avoid lock-in.

Take the workflow, not the tool. MCP and `SKILL.md` work across nearly all of these, so the habits you build today travel. And treat published benchmark scores carefully — most are vendor self-reported.

SESSION OUTCOMES

What you'll walk out with

01

New ideas over AI use cases at every stage

A repeatable model → product workflow you can apply to your own workflows

02

Examples of building with claude code

You'll translate real model logic into working application components in the room.

03

A realistic sense of the ceiling

An honest read on what current AI tools can and can't do on their own.

THE MAP FOR TODAY

Every modeling project has similar stages

You already live in this pipeline. What's new is an agent that runs code so it can help at every stage, not just the last one.

01 Collect & Centralize

Scattered team sources → one shared, structured source of truth

WITH AN AGENT

Agent reconciles mismatched sheets, designs the schema, and stands up the datastore

02 Modeling

Fit models to data, or implement equations from the literature

WITH AN AGENT

Agent scaffolds the fit or translates equations into tested functions — the science stays yours

03 Storytelling

The last mile: build charts/graphics and tools

WITH AN AGENT

Agent builds the interface, plots, and plain-language documentation faster than ever.

↔ **VERIFICATION** Check every stage's output against something you trust

↔ **ITERATION** Every stage is a loop, iteration is cheaper than every before

WHY THIS IS DIFFERENT NOW

Software help used to mean stage 3 only

THE OLD WORLD

Centralize

by hand, or a data specialist

Model

the researcher, in scripts

Display

hire a developer or a grad student etc.

WITH AN AGENT THAT RUNS CODE

1 **Centralize** → 2 **Model** → 3 **Display**

One agent, one continuous loop — each stage's real, executed output feeds the next.

STAGE ONE

Centralize the data

The bottleneck isn't one messy file — it's that the data lives in six places, on six laptops, in six versions.

STAGE 1 · CENTRALIZE

Capture data the way the work happens

The clipboard quietly shaped what got measured. Anything awkward to write down mostly didn't get recorded.

Voice → structured notes

Dictate chute-side with your hands full and gloves on. Speech becomes text, then text becomes fields in the dataset.

Photos → scores

Body condition, bunk and forage assessments read from a phone camera — consistently, and with the image kept as evidence.

Paper → database

Decades of clipboards, scale tickets, and lab reports read and digitized instead of retyped by a summer student.

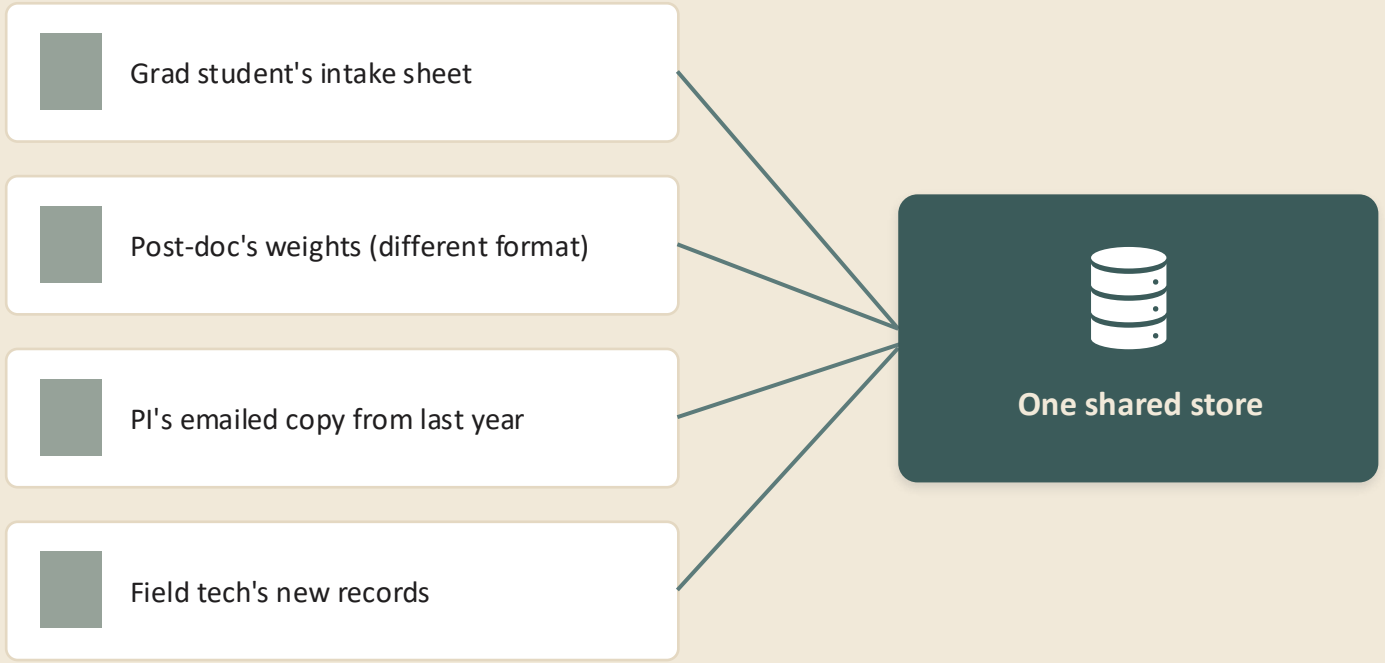
Video & audio → behavior

Time budgets, lameness, respiratory events — observed continuously instead of during a twice-daily spot check.

The catch: captured data still lands in the one shared source, and still gets validated on the way in. ***Capture you haven't verified is just faster noise.***

STAGE 1 · CENTRALIZE

Version control/centralization of disparate data



WHERE THE AGENT HELPS

1. **Reconcile**
   Reads everyone's differently-shaped sheets, writes the merge, runs it
2. **Design the schema**
   Common fields, units, and keys so IDs and dates line up
3. **Stand up the store**
   Scaffolds a simple database and the ingestion scripts
4. **Validate on the way in**
   Help with building QA mechanisms for data or track changes across versions passed around between individuals

PROJECT STRUCTURE

A place for everything

Structure isn't bookkeeping — it's what lets you (and the agent) find the one thing you need to change.

THE WORKSHOP REPO

```
nasem-workshop/
├── README.md                  how to run it
├── app.R                      the interface
├── data/                      inputs + derived
│   ├── merge_sources.R
│   └── animals.csv            ← one source of truth
├── R/
│   ├── requirements.R         ← the science, one file
│   └── score_herd.R
├── tests/
│   └── verify_requirements.R  ← proof it's right
└── docs/                      for humans
```

FIVE RULES THAT DO THE WORK

1. **One home for the science**
   Equations in a single file — never scattered
2. **Raw data is read-only**
   Derived files are regenerated, never hand-edited
3. **Tests sit next to what they check**
   Verification is part of the project, not an extra
4. **The README is the front door**
   How to run it, in the first thing anyone opens
5. **Don't commit what you can rebuild**
   Generated files go in .gitignore

Why it matters more now: "the equations are in **`R/requirements.R`**" is only a useful thing to tell an agent if there is an obvious file to point at.

STAGE TWO

Model the data

Using AI tools to help you model

STAGE 2 · MODEL

Building reproducible models, faster

FITTING

Fit a model to data

Lactation curves, growth models, mixed models

Data in → fitted parameters + diagnostics out

WITH THE AGENT

- Scaffolds the fit and picks sane starting values
- Plots residuals and flags bad convergence
- You judge whether the fit is defensible

IMPLEMENTING

Implement published equations

NASEM requirements, methane, mass balance

Equations on a page → tested functions

WITH THE AGENT

- Translates the math into clean, tested code
- Checks outputs against published values
- You confirm it matches the source, exactly

STAGE THREE

Display the results

The last mile that usually never happens — because it used to require a developer. This is where the agent's leverage is biggest and its risk is lowest.

STAGE 3 · DISPLAY

Turn a mathematical model into a tool

A user-facing tool is three layers. Each maps to a prompt — and this is the safest place to let the agent run.

Layer 1

Input interface

Forms, sliders, sensible defaults, validation

Layer 2

Logic layer

Your model runs underneath — the science

Layer 3

Output display

Tables, charts, interpreted results, plain-language docs

Why it's low-risk: *a wrong button is obvious in a way a wrong coefficient isn't, so let the agent move fast here.*

BEFORE WE BUILD

How Claude Code works

Not a chat window that hands you code, its an agent that lives in your project, runs it, reads the error, and fixes itself.

Why an agent that runs code changes everything

The chat-window version stops at “here's some code.” The agent closes the loop on real execution.

1

Reads your files

Opens the actual data, equations, and code in your repo

2

Writes the code

Implements the change you asked for in plain English

3

Runs it

Executes against your real data — not a guess at the output

4

Reads the result

Sees the error or the wrong number, and fixes it — then loops

↻ This loop is the whole game — it's what lets the agent help at every stage of the pipeline, not just hand you a draft.

PART TWO · THE CRAFT

Prompting the agent

The model is a translator — not an oracle. Learn where it's strong, and where you stay in the loop.

Integrating scientific expertise with AI tools

AI HANDLES THE UNDIFFERENTIATED WORK

- ✓ Boilerplate and glue code
- ✓ Input forms and interface components
- ✓ Output tables, charts, and layout
- ✓ First-draft documentation for non-specialists
- ✓ Rapid iteration on look and structure

SCIENTISTS STILL DO THE SCIENCE

- ! Is the model logic actually correct?
- ! Do the coefficients and units check out?
- ! Are edge cases and bad inputs handled safely?
- ! Does the output mean what a producer thinks?
- ! Is this trustworthy enough to ship?

ANATOMY OF A RELIABLE PROMPT

Structure the ask and get usable code

Vague prompts give plausible junk. These six parts turn model logic into code you can verify.

01 Context & role

"You are helping build a decision-support tool for cattle producers…"

02 The model logic

Paste your equations, coefficients, and decision rules as ground truth

03 Explicit inputs

Name every input, its units, and valid ranges

04 Explicit outputs

State exactly what to return and how to display it

05 The stack

Specify the framework / interface components you want

06 Constraints & format

Edge cases, validation, and "return only the code"

Prompting Activity

Prompting Activity

SKILLS · REUSABLE EXPERTISE

Teach the agent once, not every time

A skill is a folder of instructions the agent loads when the task calls for it — your lab's conventions, packaged.

WHAT A SKILL IS

1. **A folder with a SKILL.md**
   Plain-English instructions, plus a name and a one-line description at the top
2. **The description is the trigger**
   Claude reads it every session and loads the full body only when the task matches
3. **It can bundle more**
   Reference files and scripts the agent opens on demand — not just text

SKILLS A RESEARCH TEAM WOULD WRITE

`nasem-conventions`
Canonical equations, units, rounding, citation

`lab-data-format`
How your team's files are shaped and cleaned

`figure-style`
Plots that look like your lab's, every time

`verification-checklist`
What a model change must pass to be trusted

WHERE THEY LIVE

`~/.claude/skills/` just you `.claude/skills/` in the repo, shared with the whole team via git

Skills stay dormant until they're relevant — so a big library costs you almost nothing in context.

HANDS-ON

Build the pipeline

Starting with good prompt engineering tasks

LANDING THE PLANE

Design judgment & honest limits

Running code isn't the finish line. What makes a tool or model one a producer will trust and use?

DESIGN CHOICES THAT MATTER

What makes a tool genuinely useful

In a production or extension setting, these are the difference between a demo and a tool people return to.

Sensible defaults

Pre-fill realistic values so a first-time user isn't staring at a blank form.

Guardrails

Reject impossible inputs before they produce nonsense results.

Interpretable output

Say what the number means, not just what it is.

Earned trust

Show the assumptions; let users see why they should believe it.

Verification discipline

Never ship model logic you haven't checked against the source.

Fit the setting

Match how producers and extension staff actually work.

THE TAKEAWAY

The model → product workflow

One reusable loop you can run on any model you already have.

1

Start with model logic

Your equations, coefficients, and rules

2

Prompt the layers

Interface, logic, output with structure

3

Verify against the source

Check the math; you own correctness

4

Refine iteratively

Fix, tighten, handle edge cases

5

Ship & document

Put it in producers' hands, plainly explained

↻ Repeat the loop each time you refine.

Celium

Develop models and tools faster than ever.

Thank you · Questions?

Reid Hensen · reid@celiumgroup.com