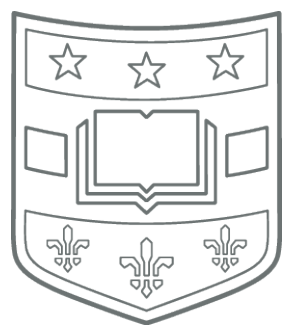
Agentic Coding for Business Research

From Idea to Paper, in Days, Not Months

Dennis Zhang

Olin Business School, Washington University in St. Louis



MODULE 0
Scope & Disclaimers

What this talk is — and what it isn't

The talk, in one sentence

A practical, no-hype guide to turning **agentic AI** — Claude Code, Codex, and the stack around them — into a **daily research collaborator**: what to set up, which workflows pay off, and the costly mistakes worth skipping.

Three honest disclaimers

1. **I'm not the expert here.** I don't build state-of-the-art auto-research platforms. This talk is about *my own learning* and pointers I've found useful — not a claim to mastery.

2. **I'm wary of fully-automated paper generation.** The goal, to me, shouldn't be "AI writes the paper." Serious mathematicians share this concern — see the **Leiden Declaration on Artificial Intelligence and Mathematics** (2026), endorsed by the International Mathematical Union; signatories include Fields Medalist Peter Scholze. (Leiden University news, June 2026)

3. **My approach is different.** I use AI as an accelerator I *steer and verify* — not the fully-automated "auto-research" that some others are pursuing.

This is deliberately not the cutting edge

What you'll see today is practical, and a little behind the frontier — on purpose. The point is reliable workflows you can adopt now.

If you want the cutting edge

Get on **X (Twitter)**. Follow the researchers at **Anthropic** and **OpenAI**, and watch how they actually use these tools day to day — that's the fastest way to track what's genuinely new.

Today: the dependable 80%. The frontier moves weekly — follow the people building it.

This isn't free — what it's actually cost me

Setting expectations up front: my real spend driving these workflows, to date.

Combined: ∼$27,967 across ∼33 billion tokens.

It buys real acceleration — but it is a real budget line. Plan for it.

Figures are illustrative. Every slide in this deck — including these tables — was generated by Claude Code.

MODULE 1

What is agentic coding?

(Claude Code) — introduction & setup

What is agentic coding? (Claude Code)

- **Agentic coding:** an AI agent that reads, writes, *and runs* code in your environment — plan → act → observe → repeat
- **Claude Code** (Anthropic): the terminal agent we'll use today
- Sees your project files, data, and dependencies
- Can script audio pipelines — e.g. transcribe meetings, analyze interviews
- Works with Python, R, Julia, LaTeX...
- The key difference from a chatbot: it runs the code and debugs it

```
* Claude Code

> Analyze the panel data
* I'll read the data, write the
  analysis, run it, generate figures.
   > Read data/panel.csv
   > Write analysis.py
   > Execute python analysis.py
 OK 3 figures saved to outputs/
```

Not just Claude Code: the same workflow runs on **OpenAI Codex** (CLI), Google's **Gemini CLI**, Cursor, and others — pick your model; the ideas transfer.

The workflow you know

If you've used ChatGPT or Gemini for coding, this loop is painfully familiar:

Ask for code → Copy code → Paste in editor → Run it → Error! → Copy error → Paste back → *Repeat...*

The problem

You become a human copy-paste machine. ChatGPT can't see your files, can't run your code, and can't see what went wrong. Every iteration requires you to be the middleman.

What makes Claude Code different

ChatGPT / Gemini	Claude Code
Lives in a browser tab	Lives in your terminal
Can't see your files	Reads your project files
Can't run your code	Writes code directly
Can't see errors	Runs it on your machine
You do all the work	Sees errors, fixes them, retries
"Here's some code, hope it works!"	*"Let me handle it end-to-end."*

The key insight: Claude Code is an **agent**, not a chatbot. It takes actions in your environment, observes results, and adapts — just like a skilled RA would.

Pricing & data security

Plan	Price/mo	Usage
Max 5x	$100	5x Pro — light research weeks
Max 20x **(rec.)**	$200	20x Pro — heavy coding, multiple projects
Team Premium	$125/seat ($100 annual)	5+ seats, central admin, SSO

What does $200/mo get you? In our experience: 3–5 full research projects per month, or ∼40–60 hours of active sessions. One complex paper implementation uses ∼$40–50.

Consumer (Free, Pro, Max):

- Data may be used for training
- Opt out in Privacy Settings
- Opted out: 30-day retention
- Allowed: up to 5-year retention

Team / Enterprise:

- No training on your data by default
- Custom retention policies
- SSO, audit logs, SCIM
- HIPAA-ready options available

Why researchers should care

Your real data
Claude sees your actual CSV files, Stata datasets, or SQL databases. No more describing "I have a column called. . . "

Your real errors
When code fails, Claude sees the full traceback and fixes it. No more copying stack traces into a chat window.

Your real environment
Claude knows what packages you have installed, what Python version you're running, what OS you're on.

Your real output
Figures save directly to your project. LaTeX tables go straight to your paper folder. No manual file management.

Bottom line

Claude Code works with your actual research project, not a simplified description of it.

Real example: updating my website

One prompt. Multiple files. Automatic deployment.

The prompt

"I just posted a new working paper — here's the link. Please update my website and CV."

The paper

A new working paper — its title, coauthors, and abstract pulled straight from the link.

```
* Claude Code
 > Fetch SSRN page -> extract abstract
 > Read index.html -> papers section
 > Read cv.tex -> find publications
 > Edit index.html -> add paper entry
 > Edit cv.tex -> add "Under Review"
 > Run pdflatex cv.tex
 > Run git add, commit, push
OK Website + CV updated & deployed
```

Context + tools: Claude reads the URL, edits multiple files with correct formatting, compiles LaTeX, and deploys — all from one sentence.

Reproducibility: conversation logs

Your entire research process is documented automatically.

What gets logged

- **Prompts** — every request, verbatim
- **Reasoning** — why each approach
- **Files read** — data/code accessed
- **Files written** — full content of edits
- **Commands run** — every bash/python call
- **Output & errors** — full stdout/stderr
- **Fixes** — how errors were resolved

Why this matters for research

- **Replication:** ship the session log — reviewers see how results were produced
- **Co-author handoff:** "here's the transcript"
- **Your future self:** 6 months later, see what you asked and why
- **Debugging:** trace the exact path to a result
- **Teaching:** share logs as workflow examples

Stored locally: `~/.claude/projects/` — organized by project, searchable, exportable.

The session log in practice

A complete record of your analysis workflow.

```
~/.claude/projects/minimum-wage-study/session-2024-01-15.log
Session started: 2024-01-15 14:32 | Project: minimum-wage-study

User: Load the state-level panel and run a TWFE regression of
  unemployment on minimum-wage treatment, state+year FE,
  cluster SEs at the state level.
Claude: I'll read the data first, then write and run the TWFE.
   Read data/state_panel.csv  OK  -> 350 rows x 8 cols
   Write analysis/twfe_regression.py  OK  -> 87 lines
   Bash python analysis/twfe_regression.py  X
        -> KeyError: 'unemp_rate' not in index
   (column is 'unemployment_rate' -- fixing)
   Edit analysis/twfe_regression.py:23  OK
   Bash python analysis/twfe_regression.py  OK
        -> b=0.023 (SE=0.041, p=0.58) saved twfe_results.tex
User: Now add an event study with a 5-year window...
```

Resume anytime: `claude --resume` picks up exactly where you left off — full context preserved.

Reproducibility: git versioning

Every analysis step becomes a commit. Your project history tells the full story.

Native git integration

- Commit after each step
- Automatic descriptive messages
- Branches for exploratory analysis
- Compare results across specifications
- Roll back instantly if something breaks

```
git log --oneline
f4a2c1e Add placebo test with leads
a3f2d1e Event study, 5-year window
b7c4e2a TWFE, state-level clustering
c9d5f3b Clean panel, impute missing
d8e6g4h Initial import from BLS
```

Research scenarios

- **Referee R2:** "try industry-level clustering" → branch, modify, compare, report both
- **Replication package:** share the repo → anyone reproduces every step
- **Co-author review:** "what changed?" → `git diff` shows exactly what
- **6 months later:** commit history + session logs

Your files stay local unless the agent reads them

Claude Code runs on your machine. Only what Claude reads gets sent to the API.

How it works

- Runs locally in your terminal
- Only files Claude explicitly reads are sent to the API
- Other project files stay on your machine
- All data encrypted in transit (TLS)

Commercial / API users
Anthropic does not train on your data. Zero-data-retention (ZDR) available for enterprise.

Consumer plans (Free/Pro/Max)

- Training opt-in/opt-out at claude.ai/settings
- Opted out: 30-day retention only
- Opted in: up to 5-year retention

For sensitive research
Use the API with a Team/Enterprise plan, or route through AWS Bedrock / Google Vertex for full data control.

How does it compare?

	Claude Code	**Web Claude / ChatGPT**	**GitHub Copilot**
Interface	Terminal (CLI)	Browser	IDE plugin
Executes code	Yes, on your machine	Sandboxed only	No
Reads your files	Full project access	Upload only	Open files only
Agentic loop	Run → debug → fix	No	No
Git integration	Native	No	No
Sweet spot	Full research pipeline	Quick questions	Line completion

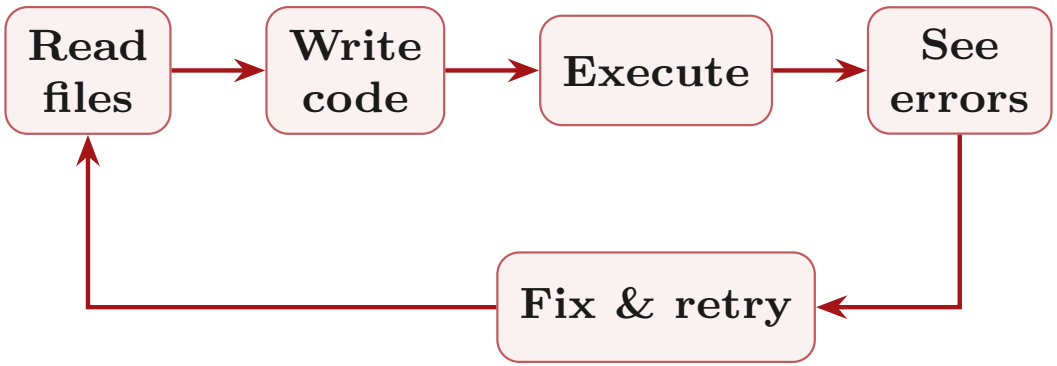
CLAUDE.md — persistent instructions

A markdown file that gives Claude project context every time it starts. Think of it as a memo to a new RA.

"Our panel is workers × weeks"
"Cluster SEs at the firm level"
"Use log(outcome) throughout"
"Save tables in LaTeX format"

```
# Project: State MW & Employment
## Data
- Panel: 25 states x 14 years
- Source: DOL + BLS (LAUS)
## Conventions
- Outcome: state unemployment rate
- Treatment: MW above federal
- Include state and year FEs
- Cluster SEs at state level
- Save to outputs/
```

The agentic loop



In practice, a typical task involves 5–15 tool calls as Claude reads, writes, runs, hits an error, fixes it, and tries again. You'll see this happen in our demos.

A few things we've learned

- **Specific prompts matter a lot.** "Run TWFE with state FEs, cluster at state level" beats "analyze the data."
- **`CLAUDE.md` pays for itself quickly.** Even 5 lines of context saves repeated prompting.
- **Git is your safety net.** Commit before asking Claude to try something risky.
- **Always verify.** Claude is impressively capable but not infallible — we've caught mistakes, and you will too.
- **Context fills up.** Use `/compact` in long sessions, or break work into smaller tasks.

Other concepts worth knowing

`/init`
Auto-generates a `CLAUDE.md` by scanning your project. Good starting point.

Git integration
Claude can commit, branch, even open PRs. Every analysis step gets a descriptive commit.

`/skills`
Define reusable research workflows as code. (More on this later.)

`/compact`
Summarizes the conversation to free up the context window. Useful for long research sessions.

MODULE 2
Setup

Part A — Hardware (software comes later)

Laptop or a dedicated server?

Where you run Claude Code matters more than people expect.

Local laptop

- Fine for single-threaded, occasional use
- Fans roar, battery drains
- Closing the lid kills long jobs
- No room for many parallel workers

Dedicated server

- Run **20–40 parallel workers** — how I actually work
- Heavy RAM + CPU always available
- Long jobs keep running 24/7
- **Reach it from your phone** — research on the fly while traveling or teaching; no laptop needed

Bottom line

If you want multiprocessing — and you will — a dedicated server isn't a luxury. It's what unlocks the workflow.

What to optimize for

Two things dominate everything else:

- **Maximum RAM.** Most of what I do is matrix multiplication and large-scale text processing — both memory-hungry. More RAM = more parallel workers and bigger models before you hit a wall.
- **Speed & a fast interconnect.** Lots of CPU and GPU cores, *and* a fast RAM↔GPU / CPU↔GPU link. Bandwidth between memory and the GPU is often the real bottleneck — not raw core count.

RAM lets you run more at once; interconnect speed decides how fast each one goes.

Option 1 — Dedicated Windows server

A classic workstation build, maxed for parallel research:

- **120 GB+ RAM**
- **Dedicated GPU** (e.g. RTX 5090, 32 GB)
- **20–24-core CPU**
- Fast NVMe storage

Ballpark cost (2026)

~**$7,300** prebuilt
(24-core, 128 GB DDR5, RTX 5090)

DIY: ~$5,000–6,000, GPU-dependent.

Strong, upgradeable, runs anything — but discrete GPU memory (e.g. 32 GB VRAM) caps the size of local models you can hold on the card.

Option 2 — Apple Mac Studio (M3 Ultra)

My pick if you're on the Apple platform:

- **96 GB unified memory** — Apple's only current M3 Ultra config
- CPU *and* GPU **share the same memory**
- → holds far larger models than a typical 32 GB discrete GPU
- Quiet, low power, small footprint

Ballpark cost (2026)

M3 Ultra (96 GB): **$3,999** — now the *only* config Apple sells.

Apple dropped the 128/256 GB upgrades in 2026; resellers still carry them at a markup.

Still *undercuts* a comparable Windows + high-end-GPU build.

Why unified memory wins: even at 96 GB, the Mac holds far larger models than a 32 GB discrete GPU — no splitting across CPU and GPU. (A next-gen Ultra will likely restore higher-memory options.)

No hardware? Spin up the cloud

For heavy downloading, big scripted jobs, or continuous non-disruptive tasks — you don't need to own a machine.

- Ask **Claude Code to launch an AWS or Google Cloud instance** — up in about a second.
- Pay only while it runs; spin it down when done → the lowest cost of any option.

Instance	Good for	Est. cost (on-demand, 24/7)
`t4g.micro`	light, continuous jobs	∼$6/mo
`t4g.small`	heavier scripts / downloads	∼$12/mo

Run it only a few hours a day and it's a couple of dollars a month.

MODULE 2
Setup

Part B — Software & tools

The core — plus a small support cast

You need an agent running on your machine — **Claude Code** and/or **Codex**. A handful of supporting tools make the workflow far more efficient:

- **Cloudflare** — host & visualize results on the web
- **GitHub** — collaboration + version control (your safety net)
- **Tailscale** — reach your own machines from anywhere
- **Terminals** (Termius, iTerm) — reliable remote access
- **tmux** — keep long jobs alive after you disconnect
- **Dictation** (Super Whisper / Typeless) — talk to your agent
- **Ollama** — a local model for private text
- **Browser control** — for web apps the CLI can't reach

Cloudflare — host & visualize (~$7/mo)

- Cheap, simple hosting for your project websites.
- **Why it matters — visualization.** We default to talking through ideas, but for *consuming* data and results, **seeing HTML** beats scrolling CLI output.
- Let the agent build a small web page; you read it like a dashboard, not a terminal dump.

Demo

I'll show this with my own **research website for literature reviews** — results rendered as a page you can actually read.

GitHub — collaborate & version

Essential, for two reasons:

- **Collaboration.** Agents like Claude Code are built for collaborative, repo-based work — branches, pull requests, shared history.
- **Version control = your safety net.** Agents make mistakes. Back up continuously so you can *verify, diff, and roll back.* "Run it, check git status, commit, push" becomes a single loop.

Never let an agent edit code you haven't committed first.

Tailscale — your machines, from anywhere

- Creates a secure, private tunnel between all your devices (a personal VPN mesh).
- Reach your dedicated server from your **laptop or phone**, wherever you are.
- Crucial for **travel** and **restrictive networks / checkpoints** — no open ports, no fragile port-forwarding.

Pairs perfectly with the "research from your phone" workflow from Part A.

Terminals & remote access

How you actually connect to your machines:

- I use **Termius** (great on phone *and* desktop) and **iTerm** on Mac — many good options exist.
- Claude Code and Codex have a built-in **"remote control"** mode with nicer visuals — but it can be unstable.
- **Dedicated terminal apps are more reliable** for day-to-day remote work.

tmux — never lose a running job

- A terminal multiplexer: your sessions **keep running on the server** even after you disconnect.
- Close the laptop, lose Wi-Fi, switch from desktop to phone — the download, training run, or long agent loop keeps going.
- Reattach with `tmux attach` and pick up exactly where you left off.

Essential for long agentic runs and the work-from-anywhere setup.

Dictation — type or talk?

A real trade-off:

Typing
Forces *concise, targeted* prompts — which agents often respond to better.

Dictation
Lets you dump lots of *ambient context* fast, giving the agent more to decide with.

- **English: Super Whisper** — excellent (my go-to).
- **Bilingual (English / Chinese): Typeless**.

Keep sensitive text local — Ollama

For text you don't want leaving your machine (confidential data, unpublished work):

- Run a model **locally with Ollama** — nothing is sent to an external API.
- Good local choices: **Qwen 2.5 (32B)** or Google's **Gemma 2**.
- Use it for sensitive summarization/extraction; reserve the cloud agents for everything else.

  This is exactly where the big-RAM machine from Part A pays off.

Browser usage — agents that see the web

Some work lives in a browser, not a terminal — dashboards, web apps, data behind a login.

- **Claude for Chrome** and **Codex** browser control: the agent reads pages, clicks, fills forms, and pulls data.
- Complements the **desktop apps** (Claude, Codex) for what the CLI can't reach.
- Handy for: scraping a rendered page, checking a deployed site, driving a web-only tool.

Improving fast — still occasionally finicky, so keep a human watching clicks on anything irreversible.

A word on privacy

The uncomfortable default: the cloud can see what you send it.

- **How it works here:** cloud agents process your prompts — and the files they read — on the provider's servers. By design, that data leaves your machine.
- **What to worry about:** retention, training opt-in/out, and that a well-crafted prompt can surface information the model was given access to. Treat "in the cloud" as "potentially readable — and promptable."
- **Mitigations:** keep truly sensitive data local (Ollama), use ZDR / enterprise terms, and never give an agent more access than the task needs.

Deeper dive — our paper *Behavioral Transfer in AI Agents: Evidence and Privacy Implications*.

MODULE 3
Demo 1: Reddit AI Impact

Applied Difference-in-Differences

Empirical results — presented elsewhere

The full Reddit AI-impact study — design, event studies, and robustness — is covered in a separate talk.

Claude iterates on the data pipeline

Multiple iterations to optimize 2 TB of raw Reddit data:

```
* Iterations
v1: MemoryError on 80GB files
v2: Too slow -- 6 hrs per month
v3: Truncate cols 50->12 (-60%)
v4: Compression brotli/zstd/lz4
v5: orjson + streaming (3x)
v6: Final -- 20 MB/s, 2TB->85GB
```

- **Column truncation:** kept 12 of 50+ cols — raw size cut 60%
- **Compression:** zstd 4.7x faster writes than brotli
- **Parser:** orjson (3x) + streaming decompression
- **High-dim FE:** pyfixest pipeline, 98K subreddit + time FEs

6 iterations to get it right — Claude tested, failed, learned, and optimized.

Parallel Claude sessions for analysis

On Jan 19 we ran 8 Claude instances simultaneously on different tasks:

1. Parallel Trends 45 msgs; Roth tests	2. Lit Review 51 msgs; SEC frame	3. External Traffic 33 msgs; Similarweb	4. Quantile Effects 28 msgs; CiC
5. HTE Analysis 8 msgs; SEC goods	6. DiD Analysis 38 msgs; 26-mo panel	7. Treatment Date 28 msgs; Aug vs May	8. Paper Draft 53 msgs; full writeup

8 parallel sessions = 8 RAs working simultaneously. Each session keeps its own context while sharing the codebase via git.

Project timeline: 10 days, 18 sessions

18 Sessions — **537** Messages — **10** Days — **2,500** Lines of Code

Day	Phase	Sess.	Key Outputs	Lines
Jan 17	Data Processing	2	Memory optimization, streaming pipeline	∼400
Jan 18	Initial Analysis	1	Reddit deal analysis, alt explanations	∼200
Jan 19	Parallel Analysis	8	DiD, parallel trends, HTE, quantile, lit	∼1,200
Jan 20	Consolidation	1	Project digest, documentation	∼100
Jan 23–24	Paper Writing	2	Full paper draft, collaboration setup	∼400
Jan 26	Presentation	1	Public-facing presentation	∼200

From raw 2 TB data to a ready-to-submit paper in 10 days.

Initial draft: The Impact of AI Search on the Online Content Ecosystem: Evidence from Google and Reddit

When Claude Code gets it wrong

Claude Code is powerful, but it makes mistakes. Three issues from this project required manual fixes:

1. Data balancing
Balanced within months, not across months.

2. Fixed effects
Used week/month/year instead of true day FE.

3. Non-deterministic hash
Author IDs can't match across months.

Lesson: Always verify Claude's output yourself. Trust but verify.

Mistake 1: data balancing

The problem: we process data monthly due to space constraints (2 TB total). Claude balanced data *within* each month, not *across* months.

What Claude did wrong

- Subreddit A in Dec 2023 but not Jan 2024 → no zeros assigned for A in January
- Subreddit B appears some days in Jan, not others → does fill zeros for missing days

The fix

- Define baseline cohort (Dec 2023)
- Require all cohort members present in every month
- Zero-fill across the entire 547-day panel
- Result: perfectly rectangular panel

Mistake 2: two-way fixed effects

The problem: Claude claimed to run a two-way FE model, but used proxies instead of true day fixed effects.

What Claude did

```
Y ~ treat_post | subreddit +
    week_of_day + month + year
```

Not a true day FE — a combination of time dummies that doesn't absorb daily shocks.

The fix

```
Y ~ treat_post | subreddit +
    date_str
```

True calendar-day FE (547 dummies) absorbs all common daily shocks: weekends, holidays, news, outages.

Mistake 3: non-deterministic hashing

The problem: to shrink dataset sizes, Claude hashed author IDs — but Python's `hash()` is non-deterministic across processes.

The consequences

- Each month processed in a separate Python process
- Same author → different hash in different months
- Cannot match author IDs across months
- Unique-author counts overinflated (∼12–14%)

The fix

- Use `hashlib.md5()` instead of `hash()`
- Deterministic across all processes
- DiD bias negligible (∼1.2%) — overcount absorbed by FEs
- Requires reprocessing from raw `.zst` files

The takeaway

Claude Code excels at

- Rapid prototyping
- Boilerplate code generation
- Exploring unfamiliar libraries
- Iterating on errors quickly
- Parallelizing research tasks

You must verify

- Econometric specifications
- Data pipeline correctness
- Cross-file consistency
- Subtle methodological choices
- Anything that affects identification

Claude Code is a 10x accelerator, not a replacement for expertise.
Use it to move faster, but always validate the output.

Generative Augmented Inference: idea & result

AI can label data at near-zero cost — but its labels are often *biased* (e.g. a 30% predicted purchase rate vs. 44% actual).

GAI in one line

- Treat AI outputs as features, not labels
- Learn $g(X, z) = E[y \mid X, z]$ via cross-fitted ML
- Valid inference *even when the AI is biased*

Pricing study (TWIN-2K-500)

Method	MAPE (%)	Coverage
Primary	522	96.6
Naive	307	89.9
PPI++	475	97.1
GAI	**190**	**99.3**

Lu, Wang, Zhang & Zhang, "Generative Augmented Inference."

Lowest error *and* best coverage — without ever needing unbiased AI.

What Claude Code did

A hard methods project — complex data, a novel estimator, model selection, and proofs:

- **Parsed** a messy HuggingFace dataset (nested JSON) into a clean feature matrix.
- **Implemented** a novel estimator from the theory (cross-entropy loss with pseudo-outcomes), then debugged wide CIs (PCA to 95% variance).
- **Ran** 4,500 model-selection experiments to pick hyperparameters — near-oracle.
- **Wrote** a variance-decomposition section with matched notation and proofs.

~3 active days, 0 RAs — calendar time was set by coauthor meetings, not the code.

Takeaway — and where it went wrong

Claude accelerates

- Complex data parsing
- Novel estimator implementation
- Exhaustive model selection
- Mathematical writing

You still own

- Research direction & framing
- Methodological rigor
- Every claim in the paper

Two real failures:

- Hyperparameter cherry-picking (made GAI look better)
- Overclaiming in writing without proof

Claude is eager to help — and that eagerness can quietly violate scientific rigor.

MODULE 4

Demo 2: Research Mining

Idea Generation at Scale

The core insight

What makes a landmark empirical paper?

Central Question × Dominant Players × Public Data at Scale × Clean Exogenous Shock

The problem
Finding good natural experiments is ad hoc — we stumble upon shocks, then look for data. What if we could systematically enumerate all possible combinations?

The solution — decompose the search:

1. Catalog all exogenous shocks
2. Catalog all public datasets
3. Generate all combinations
4. Score and rank systematically

Step 1: finding the shocks

Policy changes, platform decisions, regulatory actions, geopolitical events...

Platform policy
API pricing changes; content moderation; algorithm updates; feature launches/shutdowns
hundreds

Regulatory
GDPR, DSA, DMA; country-level bans; age verification laws; antitrust rulings
hundreds

AI / technology
ChatGPT launch; Copilot availability; model releases; AI product bans
hundreds

Many categories → **thousands of shocks** cataloged with dates, mechanisms, and treatment/control groups.

Step 2: finding the datasets

Micro-level, behavioral, public, 100K+ observations, panel structure.

Social media
Reddit (Arctic Shift); Twitter/X; TikTok Research API; YouTube Data API

Developer activity
GitHub (gharchive); Stack Overflow; npm/PyPI downloads; open-source commits

Platform / commerce
App Store data; Facebook Ad Library; Wikipedia pageviews; Google Trends

Many domains → **thousands of datasets** with quality scores, access methods, coverage.

Excluded: stock prices, aggregate indices, survey-only data, LLM training corpora.

Step 3: generate all combinations

Thousands of Shocks × **Thousands of** Datasets = **Millions of** Combinations

Example combinations:

Italy ChatGPT Ban	× GitHub Activity	→ Developer productivity shock
Reddit API Pricing	× Arctic Shift	→ Platform ecosystem effects
TikTok US Ban	× App Store Data	→ Platform substitution

Step 4: score and filter

Stage 1: programmatic scoring
Automated filters on millions of combinations:

- Domain relevance: shock ↔ dataset match
- Info-econ relevance: keyword matching
- Data quality: behavioral, large-scale
- Keyword match: high-value co-occurrences

Millions → a smaller pool (above a score cutoff)

Stage 2: agent evaluation
Deep qualitative assessment:

- Causal Identification
- Economic Meaning
- Data Quality
- Impact
- Novelty

thousands evaluated → a shortlist of top ideas

Millions → fewer → thousands → a shortlist

Why this isn't traditional p-hacking

P-hacking

Run many regressions on the same data, then report whichever are significant.

- Searches the space of *results*
- Data already in hand
- Specification chosen after seeing outcomes
- Inflates false-positive rate

Research mining

Evaluate whether a design is valid *before* ever touching the data.

- Searches the space of *ideas*
- No data accessed during scoring
- Design evaluated before estimation
- One pre-specified test per idea

Key distinction: we score shock quality and design plausibility — not statistical significance. The researcher picks an idea, then tests it once with a pre-registered design.

Two example ideas (14/14 score)

Meta news block, Canada
Shock: Meta blocks all news for Canadian users (Aug 2023, ongoing)
Data: Reddit news subreddit activity, Canada vs. similar markets
Design: Canada vs. Australia/UK DiD
ID 3/3; Meaning 3/3; Data 3/3; Impact 3/3; Novelty 2/2

TikTok 14-hour US blackout
Shock: TikTok offline for US users 14 hrs (Jan 2025)
Data: App Store downloads, engagement
Design: US vs. international — where did users go?
ID 3/3; Meaning 3/3; Data 3/3; Impact 3/3; Novelty 2/2

The importance of specification

How you ask matters. Checking novelty for "Canadian news bans and political views":

```
LITERAL SEARCH
> Are there papers on Canadian
  news bans?
* Searched "Canadian news ban".
  No results found.
```

Too specific — yields nothing.

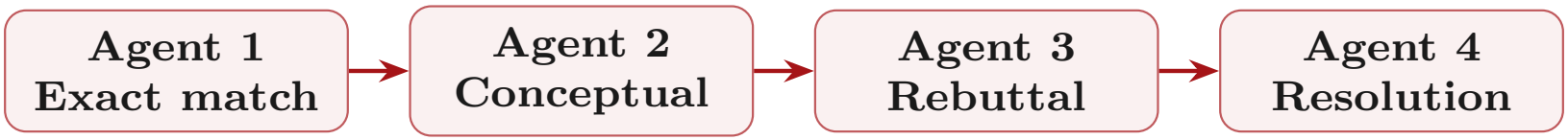
```
CONCEPTUAL SEARCH
> What papers study platform
  news restrictions conceptually?
* Found 3 in Marketing Science
  on news bans...
```

Finds Song & Manchanda (2025), Australia.

Key paper found: Song, Y. & Manchanda, P. (2025). "Does Carrying News Increase Engagement with Non-News Content on Social Media Platforms?" *Marketing Science* 44(4):748–759.

Multi-agent novelty checking

A multi-agent workflow handling both literal and conceptual matching:



- **Agent 1:** "exact shock+dataset combo?" → "No exact match"
- **Agent 2:** "conceptually similar?" → "Song & Manchanda (2025), Australia, 8 days"
- **Agent 3:** "can the author defend the idea?" → "Canada = 2.5 yrs vs. 8 days"
- **Agent 4:** "is the rebuttal valid?" → "Yes — novel angle confirmed"

Agent Skills: natural-language workflows

An open standard (Anthropic, Dec 2025) for encoding multi-agent workflows in natural language.

```
novelty-check/SKILL.md
---
name: novelty-check
description: Multi-agent
  literature novelty
---
# Novelty Checking Protocol
1. Agent 1: Exact match
2. Agent 2: Conceptual similar
3. Agent 3: Rebuttal if overlap
4. Agent 4: Validate rebuttal
Return novelty score 0-2.
```

What are Skills? "Like an onboarding guide for a new hire" — package domain expertise into composable, reusable workflows.

- YAML frontmatter + markdown body
- Progressive loading — context on demand
- Adopted by OpenAI for Codex CLI

At scale: run on tens of thousands of idea combinations; consistent, reproducible evaluation; systematic literature-review automation.

AI is already transforming research

AlphaFold & the Nobel Prize

The 2024 Nobel Prize in Chemistry went to three: David Baker (computational protein *design*), and Demis Hassabis & John Jumper (AlphaFold — protein-structure *prediction*).

- Before: years per structure, PhD expertise
- After: minutes per structure, accessible to all
- 200M+ proteins now predicted

The disruption

Structural biologists can no longer work the same way:

- Experimental validation > novel prediction
- New questions now tractable
- Competition with AI-augmented labs
- The research agenda itself changed

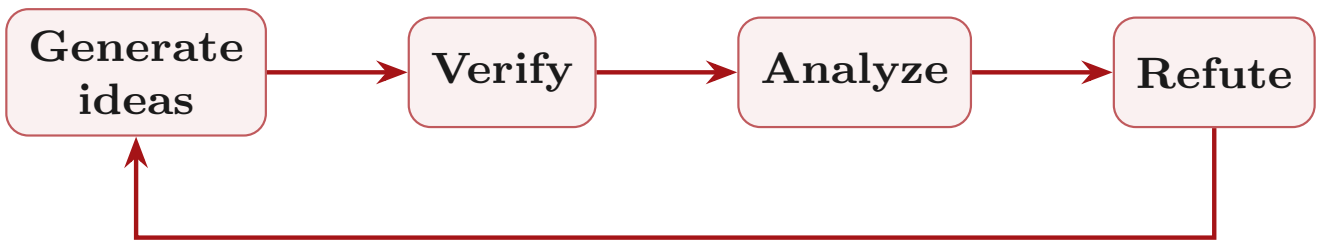
The question for business research: what does our "AlphaFold moment" look like?

MODULE 5
The Future of the Research Pipeline

Where this is all heading

The workflow: a closed loop

The research process, run as an agentic loop:



Each step is something an agent can draft and you verify — and the loop tightens over time.

Referee3.app — built for review

A tool I built specifically for the **verify / refute** parts of that loop:

- Pressure-tests a research idea *before* you commit time to it.
- Helps with novelty checks, refutation, and literature review — the review work, not the writing.
- Keeps the human in the loop: the agent surfaces the evidence; you make the call.

Demo

Referee3.app — I'll walk through how it reviews and refutes an idea live.

Continuous learning & agent autonomy

Two threads worth watching — both about agents getting better *and* more independent:

`/loop` (Claude Code)

- Time operations at set intervals
- Great for checking downloads or keeping a long proof on track
- Only as good as your verification of each pass

`/goal` (ChatGPT / Codex)

- You set the *final goal*; the agent decides the steps
- A trillion-parameter model running an MDP — still experimental, doesn't always work
- But autonomous decision-making is a genuinely interesting direction

The frontier: agents that improve over time and steer themselves — you set goals, you check results.

Thank you

Questions welcome — and let's try it on something real.

None of us has this fully figured out — we'll all learn to do this together.