Writing More Code with AI Agents

Tyler Collins
SHARCNET
tk11br@sharcnet.ca

Writing More Code with AI Agents

AI coding agents can produce what feels like an infinite amount of code, extremely quickly. In research, that is only useful if we can trust the result.

This talk presents a practical approach to using AI agents in research-code workflows without assuming their output is reliable. We will follow a worked example from raw data through cleaning, validation, analysis, and visualization, identifying where subtle errors can occur and how we might catch them. The goal is not to delegate scientific thinking or judgement to an agent. Instead, we will use agents to cheaply produce sanity checks, diagnostic plots, test cases, and other supporting work that is often expensive in time or expertise to create.

We will then extend this example into reusable patterns for notebooks, shared pipelines, SLURM jobs, and other research-computing workflows. Along the way, we will introduce practical concepts such as context, workflow memory, skills, automated checks, and human-in-the-loop review.

Basic familiarity with large language models is expected. No prior experience with AI coding agents is required.

Outline

- Today's goal is to try and **convince you to write more code**
  - … with AI
- Outline:
  - Defining what I really mean by "more code"
  - Claude demo problem with different levels of guidelines
  - Summarizing the results
  - How to extend these principles
  - Tangible suggestions on where/how to implement new principles

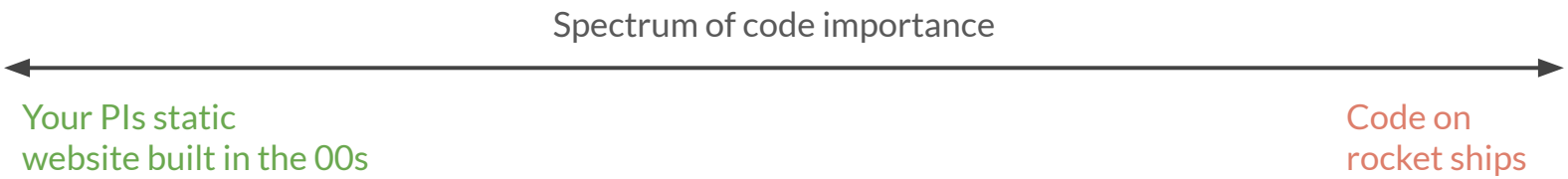
We'll wrap up with some takeaways and open questions

You are not writing enough code

- Probably a statement you're seeing on various social media platforms
- For certain definitions of code, this statement is true
- Large element of "fear of missing out" (FOMO)
- People churning out dozens of low effort publications
  - 50+!
- Some researchers have significantly accelerated their labs responsibly
  - Was this all just claude and prompting?

My interpretation: use AI to write more throwaway code, not rocket ship code!

What is “rocket ship” code?



Where is your project? Where does research fall?

Where is it okay to stop checking every single line fully?

“Every line of code is important!”

- “If my results are wrong/false I don’t get funding anymore!”
- What about all of the things that are “code” type tasks that exist outside of the meat of the project?
  - Logging, custom lint rules
  - Debuggers, stack trace parsing
  - Fancier graphics in reports
  - Automation, CI, CD, builds, releases
  - Profiling, tests, and other scaffolding
  - Porting old workflows to new tools (MATLAB -> Python)

These things can help you trust each line of code more

Research software stays “research software” because the non-coding tasks above get neglected

The title was a bit of a lie

Code is not just the scientific loop or software:

- Docs
- CI, CD, builds, releases
- Tests, debugging tools, reports

On the topic of titles:

- The title for the talk was “Writing More Code with AI Agents”
- It should really be **“Use AI agents to write slop to verify your code”**
    - This doesn’t really pull the same audience or have the same effect…

To be clear...

Some disclaimers for this point in the talk:

- Researchers (and software developers) still own the code they generate
  - If it causes problems, they own those problems
- Do not put your data into LLMs without checking you are allowed to do so
- **Never blindly trust results if they can have negative impact**

Target your “more code” generation at safe sections of your project that are tiresome to implement

Agents are very effective at these types of problems!

So let's test it

The experiment for today:

- Mirror a common research pattern
- Capture Claude's output, traces, and generated artifacts
- Review each level and summarize at the end

Test at various levels inspired by:

1. "Claude, do the thing"
2. "Claude, here's how my study works and some things you should know about"
3. "Claude, I wrote the science loop. Here's what you should know about my data. Help me verify it"

Synthetic data

- Neuroscience inspired behavioural data with a 2x2 repeated measures design
- 48 participant CSV files
- 4 conditions per participant
  - Memory load: high vs low
  - Stimulus type: object vs scene
- 80 trials per condition, totalling 320 rows per participant
- A specific button press was required for each stimulus type
  - Response times and correctness

Memory load trials should be statistically significantly different

Synthetic data: it's never that simple

- Real data is never that clean
- Some "spice" thrown in to make the data feel real:
  - Subject 71 duplicates participant 17
  - Two participants did not complete the task and have incomplete files
  - Two participants have near-chance accuracy (not paying attention)
  - Two participants with implausibly slow reaction times
  - Two participants with super fast reaction times

This is much more accurate portrayal of what data looks like

Synthetic data: handling problems

- All of the spice from the previous slide are normal things that can happen
- If the problematic files are allowed to contribute, the results are necessarily wrong!
- Most of these problems have standard solutions per field
    - Often they require you to report these mistakes

What sort of output do we get from Claude at the various levels?

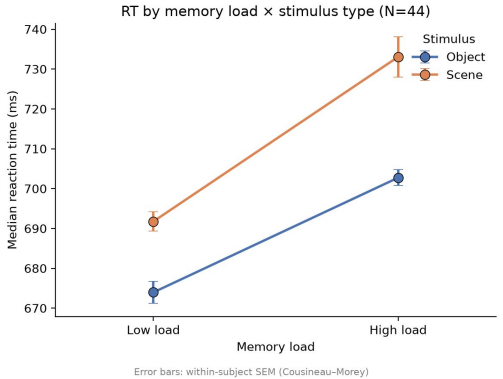
Claude experiment setup

Here's the common settings for each testing level:

- Fresh Ubuntu 24 VM
  - For security purposes
- Install Claude code
  - No plugins, no skills, no extensions, no anything
- Personal plan
- Opus 5 in auto mode

Level specific information will be in its respective section

Level 1: “Claude, make it so!”



- Extremely vague prompt:
  - “These CSV files are the results from my experiment. Do my stats for me and make a plot I can use in my paper.”
- Output looks like a publishable figure in style
- Several hidden decisions:
  - Invented its own accuracy exclusion
  - Computed median reaction times? Trimmed mean is much more common
  - Allowed comically large reaction times
  - Rough sensitivity analysis
  - Incorrect responses are left in

Level 1: “Claude, make it so!” - summary

- Don’t do this
- This is not what I am saying to do, ever

You own none of the code here, and all of the responsibilities

- In this case, Claude reported a significant effect of **p=0.040**

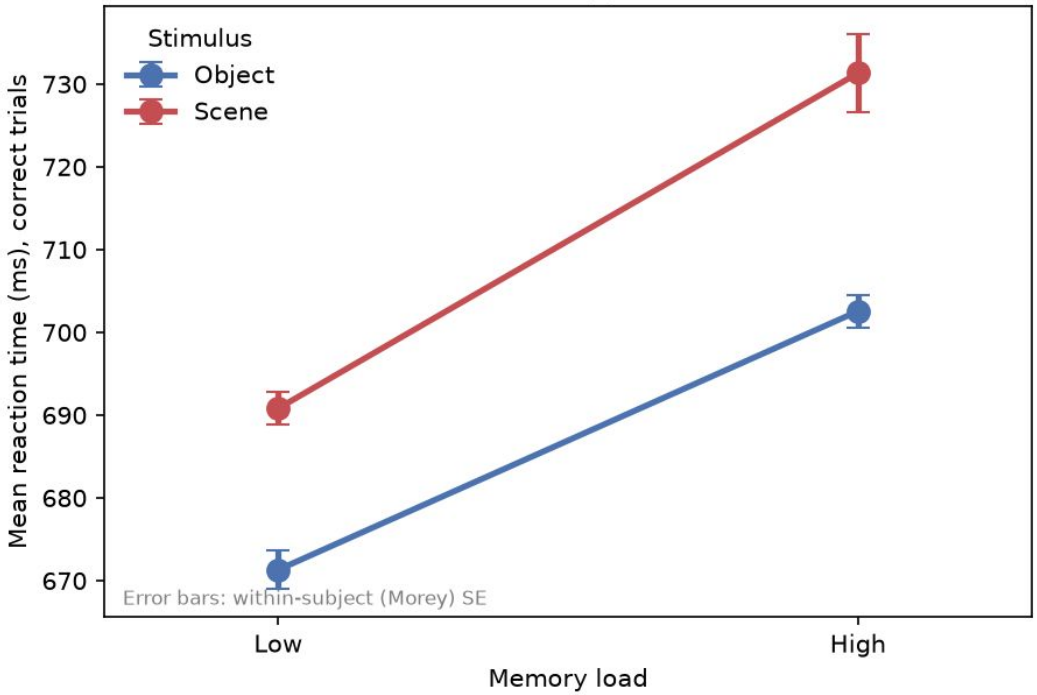
This is the worst case scenario as result is blatantly wrong

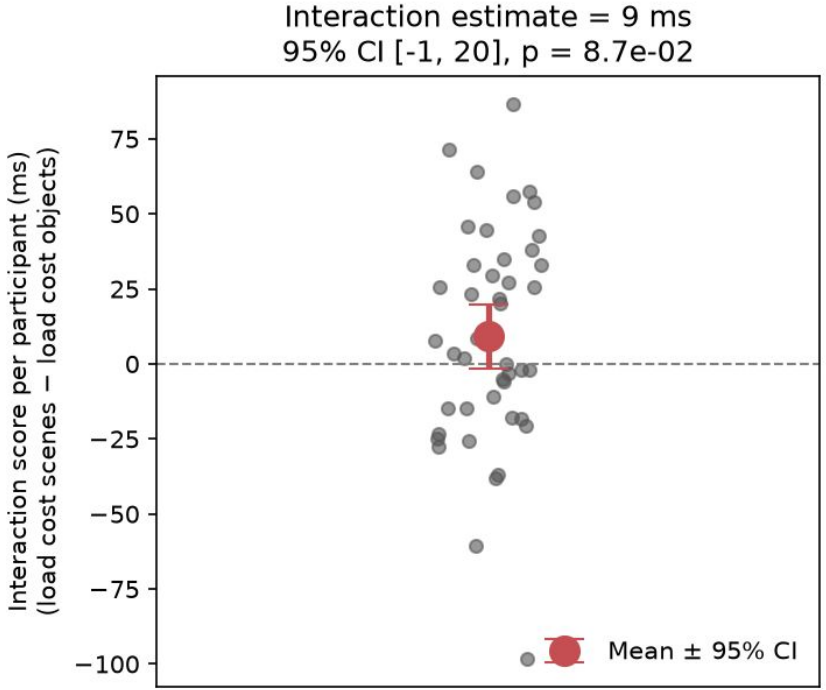
```
# ---- RM-ANOVA on accuracy ----
print("\n" + "=" * 70, "\nREPEATED-MEASURES ANOVA  —  Accuracy")
print(rm_anova(a, "accuracy", analysis_subjects).round(4).to_string())

# ---- Sensitivity: keep near-chance performers (still complete cases) ----
sens = sorted(set(cell["participant_id"]) - set(incomplete))
print("\n" + "=" * 70,
      f"\nSENSITIVITY — RT ANOVA including near-chance performers (N={len(sens)})")
print(rm_anova(cell, "rt_median", sens).round(4).to_string())
```

Level 2: Including study design

- Prompt:
  - “These CSV files contain trial-level data from a synthetic 2x2 within-subject reaction-time study. The two factors are memory load (low or high) and stimulus type (object or scene). Reaction time in milliseconds is the outcome, and the `correct` column marks correct and incorrect responses. Test the within-subject memory-load by stimulus-type interaction using an appropriate repeated-measures analysis. The hypothesis is that the effect of high memory load on reaction time is larger for scenes than for objects. Report incorrect trials from the reaction-time analysis. Report the participant count, interaction effect estimate, uncertainty, and p-value, and make a plot suitable for presenting the result.
- Analysis becomes much more conservative and reports p=0.087
- More context improved the core of the code, but the design still contains errors
  - Still bloated and not what a human would write
- Most people stop here
  - As a human running the study, I have a dozen other quality checks that the data should still go through
  - This stage is where I am suggesting AI can help the most responsibly

Memory-load × stimulus-type interaction on reaction time (N = 46)





```
print("n files:", len(files))
print("total rows:", len(df))
print("\ncolumns/dtypes:\n", df.dtypes)
print("\nmemory_load values:", df.memory_load.unique())
print("stimulus_type values:", df.stimulus_type.unique())
print("correct values:", df.correct.unique(), df.correct.dtype)
print("\nparticipant_id: n unique =", df.participant_id.nunique())
print("participant range:", sorted(df.participant_id.unique()))
print("\nacquisition_id: n unique =", df.acquisition_id.nunique())

# Multiple acquisitions per participant?
pa = df.groupby("participant_id").acquisition_id.nunique()
print("\nparticipants with >1 acquisition:\n", pa[pa > 1])

# acquisition to participant mapping — duplicated participants under diff acq?
ap = df.groupby("acquisition_id").participant_id.nunique()
print("\nacquisitions mapping to >1 participant:\n", ap[ap > 1])

# reaction time sanity
print("\nRT describe:\n", df.reaction_time_ms.describe())
print("RT n null:", df.reaction_time_ms.isna().sum())
print("RT <= 0:", (df.reaction_time_ms <= 0).sum())
print("RT negative:", (df.reaction_time_ms < 0).sum())

# correct rate
print("\ncorrect rate overall:", df.correct.mean() if df.correct.dtype==bool else df.correct.value_counts())

# cells per participant (after correct filter)
print("\n--- per participant trial counts by cell (all trials) ---")
ct = df.groupby(["participant_id","memory_load","stimulus_type"]).size().unstack(["memory_load","stimulus_type"])
print(ct.to_string())

# duplicates
print("\nfull duplicate rows:", df.duplicated().sum())
```

```
# ---- cross-check with full RM-ANOVA ---------------------------------------
long = (complete.stack(["memory_load", "stimulus_type"], future_stack=True)
        .rename("rt").reset_index())
aov = AnovaRM(long, "rt", "participant_id",
              within=["memory_load", "stimulus_type"]).fit()
print("RM-ANOVA cross-check:")
print(aov.anova_table.round(4))
inter = aov.anova_table.loc["memory_load:stimulus_type"]
print(f"\n  interaction F({int(inter['Num DF'])},{int(inter['Den DF'])}) "
      f"= {inter['F Value']:.3f}, p = {inter['Pr > F']:.3e}")
print(f"  (t^2 = {t**2:.3f} matches F; RM-ANOVA interaction == paired t on D)")
```

Very fragile implementation

Level 2: Including study design - summary

- Actual statistical portion or code was higher quality
- Hints towards the agent being capable of more “extras” than just the core of the project
- The researcher in this case still does not own enough of the code
- Strictly less errors, but still contains issues
    - All would be found by your PI discussing the results in a meeting

Good prompting does not cover for domain expertise and experience!

Level 3: Own the logic and delegate

- Wrote my own core stats logic borrowed from previous projects
- Prompt is now entirely about helping write validation scaffolding
    - Excerpt: “Report incorrect trials from the reaction-time analysis. Report a participant if they are missing any of the four conditions or completed less than 75% of the expected 80 trials per condition. Report a participant if more than 30% of their responses are incorrect. Report a participant if any reaction time falls outside 200 to 2000 ms.”
- These reports are interpreted by a human before being acted on
- Leads to the full 8 subjects being rejected and handles the duplicate case
- The “slop” code exists outside of the core portion of the project

What are the results of the statistical tests?

Level 3: Own the logic and delegate - summary

- Correct result of: N=40, p=0.037!
- Stronger effect and much more trustable results than previous levels
- Human gave expertise, direction, criteria
- Failures or inconsistencies in the data are now inspectable

Agent wrote tiresome extras that made the project better

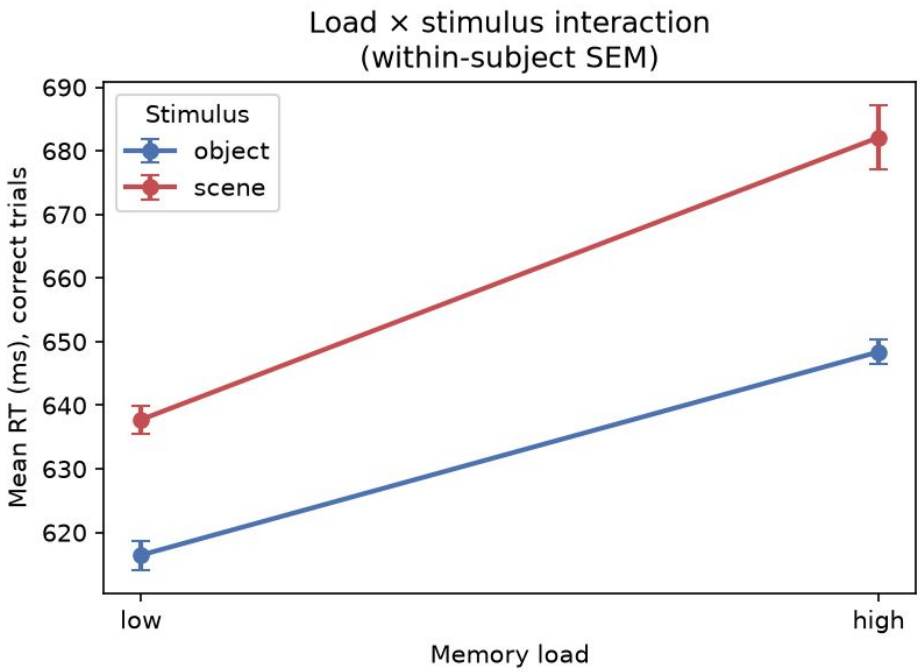
```
print("=" * 70)
print("SCHEMA CHECK")
print("=" * 70)
print("All files share expected columns:" ,
      all(list(pd.read_csv(p, nrows=0).columns) == EXPECTED_COLS for p in INPUTS))

print("\n" + "=" * 70)
print("IDENTITY: filename subject vs participant_id")
print("=" * 70)
mism = meta[meta.apply(lambda r: str(r.file_sub) != r.participant_id, axis=1)]
if len(mism):
    print("MISMATCH between filename and participant_id:")
    print(mism[["file", "file_sub", "participant_id", "acquisition_id"]].to_string(index=False))
else:
    print("None")

print("\n" + "=" * 70)
print("ACQUISITION-ID UNIQUENESS")
print("=" * 70)
dup_aid = meta.groupby("acquisition_id").filter(lambda g: len(g) > 1)
if len(dup_aid):
    print("DUPLICATE acquisition_id across files:")
    print(dup_aid[["file", "participant_id", "acquisition_id", "n", "md5"]].to_string(index=False))
else:
    print("All acquisition_ids unique")

print("\n" + "=" * 70)
print("DUPLICATE PARTICIPANT_ID across files")
print("=" * 70)
dup_pid = meta.groupby("participant_id").filter(lambda g: len(g) > 1)
print(dup_pid[["file", "participant_id", "acquisition_id", "md5"]].to_string(index=False) if len(dup_pid) else "None")

print("\n" + "=" * 70)
print("BYTE-IDENTICAL FILE CONTENT (md5 collisions)")
print("=" * 70)
dup_md5 = meta.groupby("md5").filter(lambda g: len(g) > 1)
print(dup_md5[["file", "participant_id", "acquisition_id", "md5"]].to_string(index=False) if len(dup_md5) else "None")
```

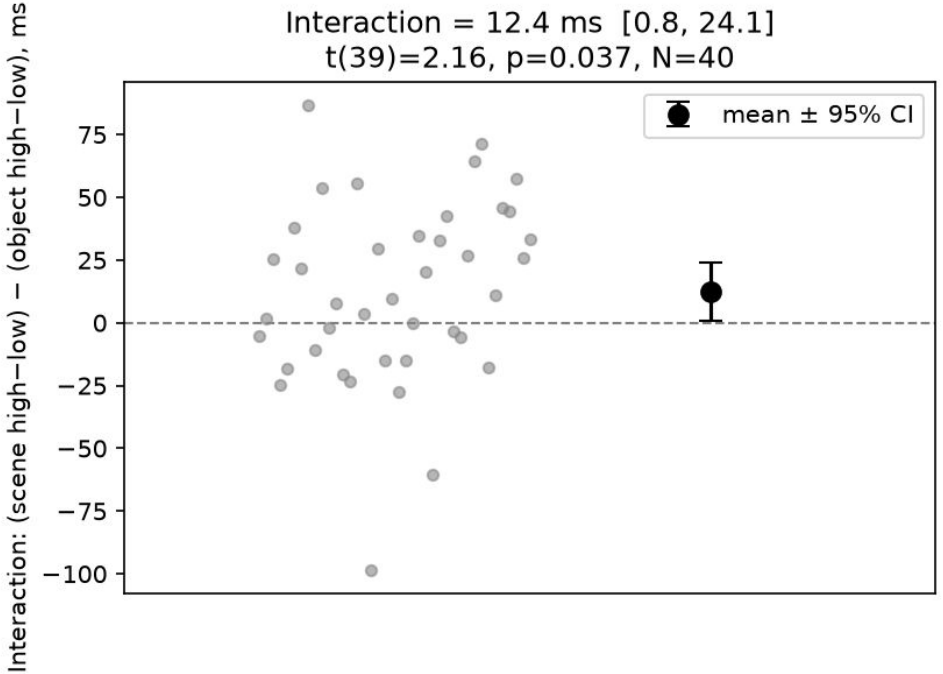
Preview | Code | Blame | 49 lines (49 loc) · 1.84 KB

Search this file

	participant_id	n_total	pct_incorrect	min_cell	n_conditions	rt_oob_correct	rt_oob_all	excluded	reasons
2	1	320	0.119	80	4	0	0	False	
3	2	320	0.094	80	4	0	0	False	
4	3	320	0.128	80	4	0	0	False	
5	4	320	0.075	80	4	0	0	False	
6	5	320	0.103	80	4	0	0	False	
7	6	320	0.075	80	4	0	0	False	
8	7	320	0.081	80	4	0	0	False	
9	8	320	0.081	80	4	0	0	False	
10	9	320	0.059	80	4	0	0	False	
11	10	320	0.078	80	4	0	0	False	
12	11	320	0.1	80	4	0	0	False	
13	12	320	0.072	80	4	0	0	False	
14	13	320	0.091	80	4	0	0	False	
15	14	320	0.053	80	4	0	0	False	
16	15	320	0.072	80	4	0	0	False	
17	16	320	0.062	80	4	0	0	False	
18	17	320	0.087	80	4	0	0	False	
19	18	320	0.075	80	4	0	0	False	

High-load RT cost is larger for scenes than objects





Test conclusions

- Claude found the duplicate participant each time
- Followed all instructions, no real hallucinations
- Taking ownership of more of the code improved results in both code quality and correctness
- Be very careful of “make it so” type prompts in research
- Involving the human in the process was what generated success

Context helps, checks are a must!

Why was level 3 the best

- Every assumption was cheaply converted to a report or a test that can fail
- Generated diagnostic summaries
  - Tables and figures
- Increased confidence in the project

Gave “free” insight into the data

How do I apply “level 3” type principles to my work if it doesn’t quite look like this?

Extending these principles

How can agents help with my…

- Notebooks: caching files, pulling out code from cells into scripts to share, swapping to pipelines
- Shared pipelines: validating inputs, generating nicer reports, tests
- SLURM scripts: profiling, checkpoints, logging

Tons more options out there, just ask the agent - you still own the core

An agent catching an error in a notebook saves minutes, catching a validation error in a SLURM workflow could save days

Concrete suggestions

- Populate your AGENTS.md or CLAUDE.md with information unique to your project
  - Make tracking sheets referenced by these documents
- Investigate Skills for complicated procedures to have an agent help
- Sandboxing lets you run agents safely without destroying your system
  - Running a VM in Nibi cloud, containers, third party solutions
- Don’t bloat your setup with plugins you never use or only use for one project
  - The more things an LLM has in its context, the worse it begins to perform
- Keep the “human in the loop” when the task could have negative outcomes

Happy to discuss more at the end of the talk!

Takeaways

- Build evidence for AI driven development, not blind confidence
- Be creative in what you ask for: custom debuggers, reports, etc
- Human in the loop strategies are powerful in research computing

Thank you for your time!

Questions?

tk11br@sharcnet.ca