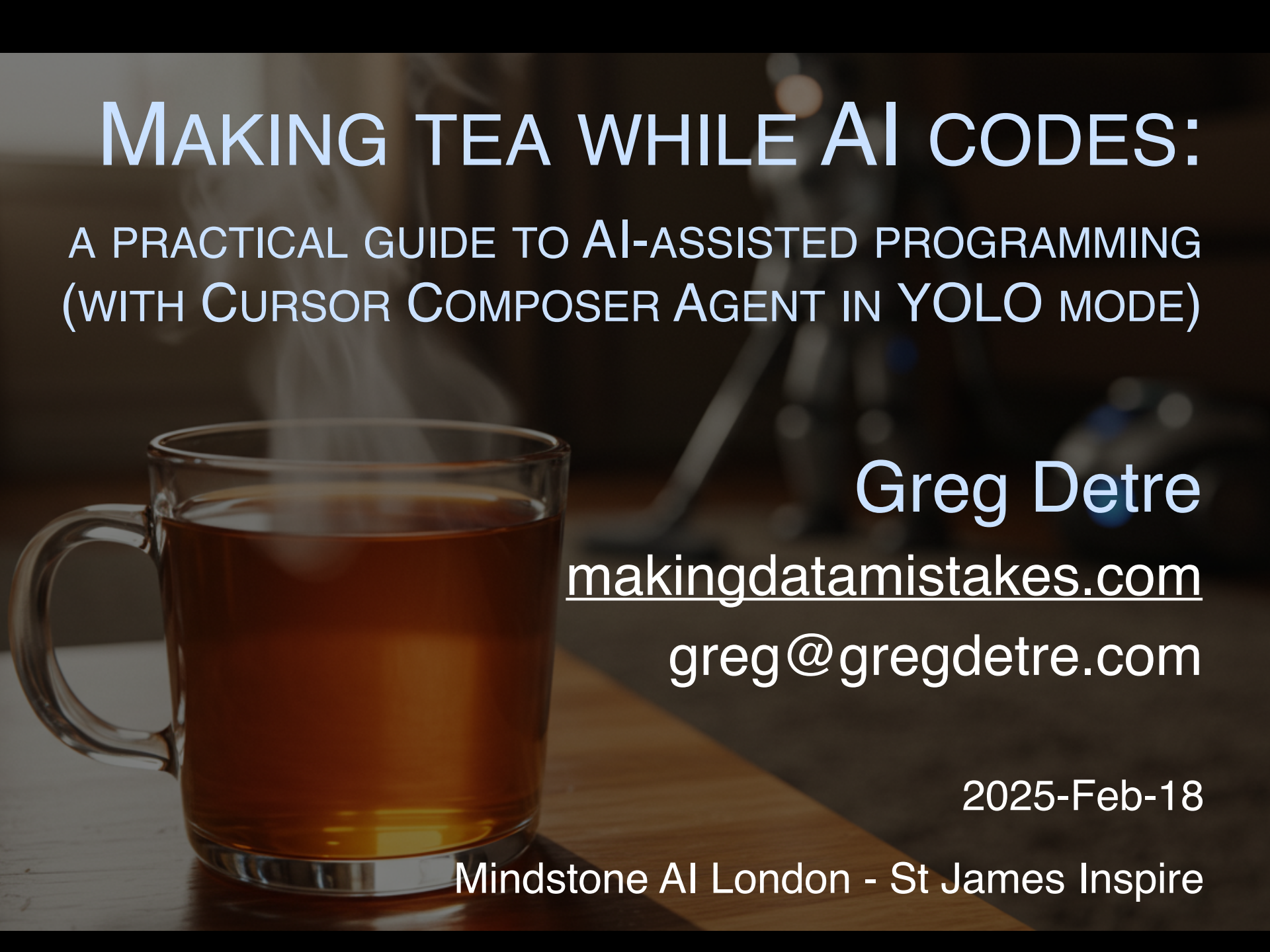


Making tea while AI codes:

A practical guide to AI-assisted programming (with Cursor Composer Agent in YOLO mode)

Greg Detre

makingdatamistakes.com

greg@gregdetre.com

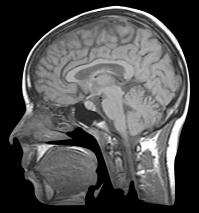
2025-Feb-18

Mindstone AI London - St James Inspire

Intro

- **machine learning expert** *Oxford, Harvard, Princeton*
- **successful startups** *Memrise + Big Health $100m+*
- **team builder** *'Best UK data & analytics team' 2019*

Dr Greg Detre



PhD
human memory,
machine learning

MEMRISE co-founder

Big Health CTO

Chief Data Scientist

Greg Detre Consulting help teams build great products and use AI in imaginative ways

Goals for today

What is Cursor? setup

We are all managers (of AI colleagues)

Cup-of-Tea criteria & TDD 2.0

Guardrails & reversibility

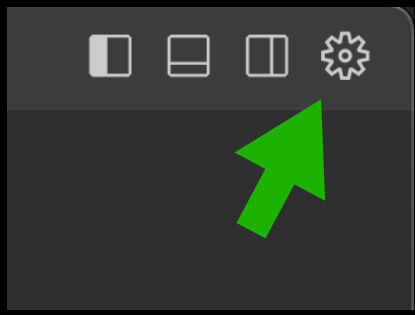
Bigger projects, conversation patterns

Getting buy-in from the rest of your team

Demoooooo...

Cursor setup

- Download from https://www.cursor.com/ (`0.45.*` at the time of writing)



- Open Cursor settings:
  - General / Privacy mode / Enabled
  - General / Rules for AI (see next page)
  - Models:
    - `Claude-3-5-sonnet-20241022`
    - and a reasoning model for tricky cases
  - Features: just turn them all on (especially YOLO mode)
    - maybe “Command denylist”, e.g. `deploy`, `git push`?
  - Beta: turn on all but “Bug finder”

Cursor settings / General / Rules for AI

```
# Safety

Never run major destructive operations (e.g. dropping tables)
first.

Don't run database changes outside of a migration without asking me first.

Don't remove comments or commented-out code unless explicitly asked to.

Don't commit to Git without asking me.

# Process

Before you make changes, first make a proposal and ask questions.

If you notice a problem or see a better way, discuss before proceeding.

If you feel you're getting stuck, stop, review, and let's discuss.

# Coding

Aim for simplicity and readability.

Aim to keep changes minimal, focused to a few areas at a time, and specific
to the current task.

Run the tests often to monitor and guide our progress, especially if you
think you've finished something. If you added a test, definitely run it.

For complex changes, start simple, and we'll add in complexity later. Aim to
break things into lots of smaller stages.

Aim to stop at the end of each stage, with working code and passing tests.

# Most importantly

Don't make irreversible changes without my permission.
```

Starting point

https://github.com/gregdetre/Mindstone-AI-London-2025-Feb-18-Cursor

- Provide your own `.env` file with OpenAI key

```
OPENAI_API_KEY=sk-blahblah
```

- empty `requirements.txt`

Create a new AI conversation

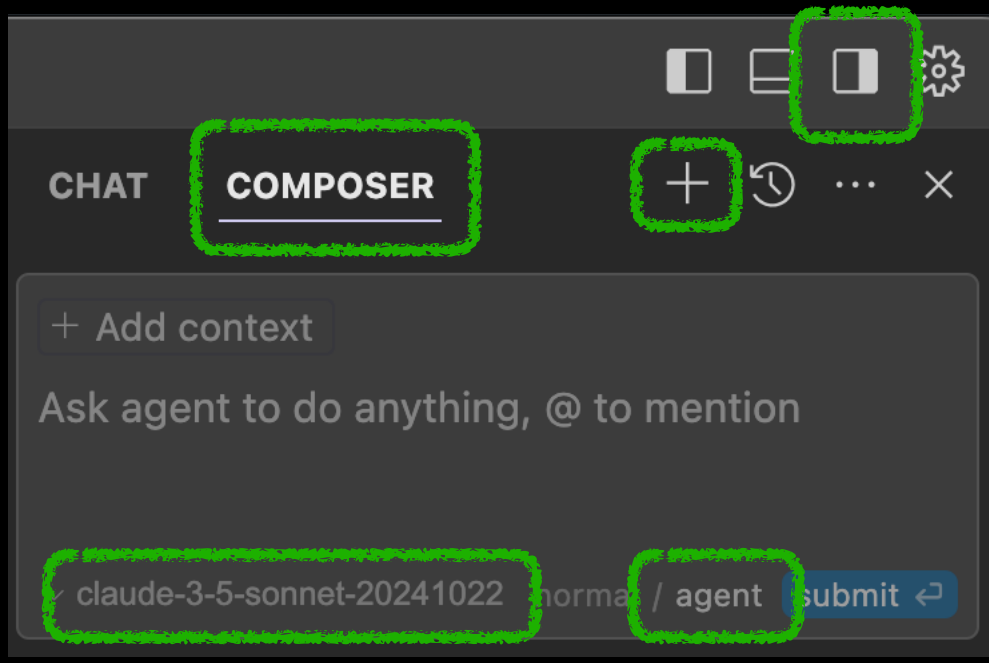
Open the AI panel in the right sidebar

Choose `Composer` (rather than `Chat`)

The `+` to the right starts a new conversation

Make sure model = `claude-3-5-sonnet-20241022`

And make sure you're in `agent` rather than `normal` mode



Provide context, as if to a brand-new employee

```
We want to build a vector
search app for a live
demo.

Keep things really simple,
get something basic
working, and then add
features incrementally.

Ask me questions, make
proposals, don't make
changes yet.
```

goal, context

operating principles

information-gathering & alignment mode

Answer questions, check alignment, refine prompt

```
1) …

2) …

Before you make changes, write
out a concise, minimal plan in
`250216_initial_planning.md`:

- goal

- context, background,
principles

- prioritised actions (marked as
TODO or DONE)

(Don’t add information beyond
what I’ve provided)
```

answer the questions

now let’s write out a planning doc

use this structure for the planning doc

Edit the plan manually, or ask the AI to add to it

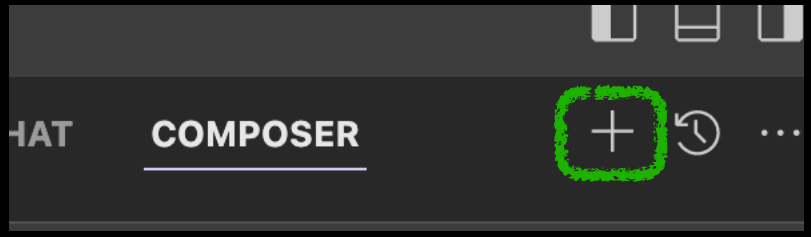
you can edit
the plan manually

```
Command-line at first.

We'll need to create a mini
dataset.

Create a single integration test
with Pytest before we start
coding

Update
@250216_initial_planning.md
```

instructions you
want the AI
to add to the plan

update doc

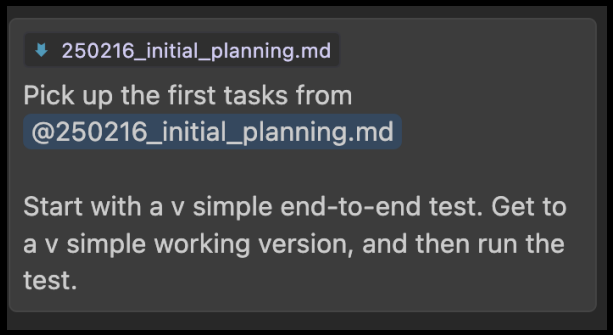
Start a new conversation



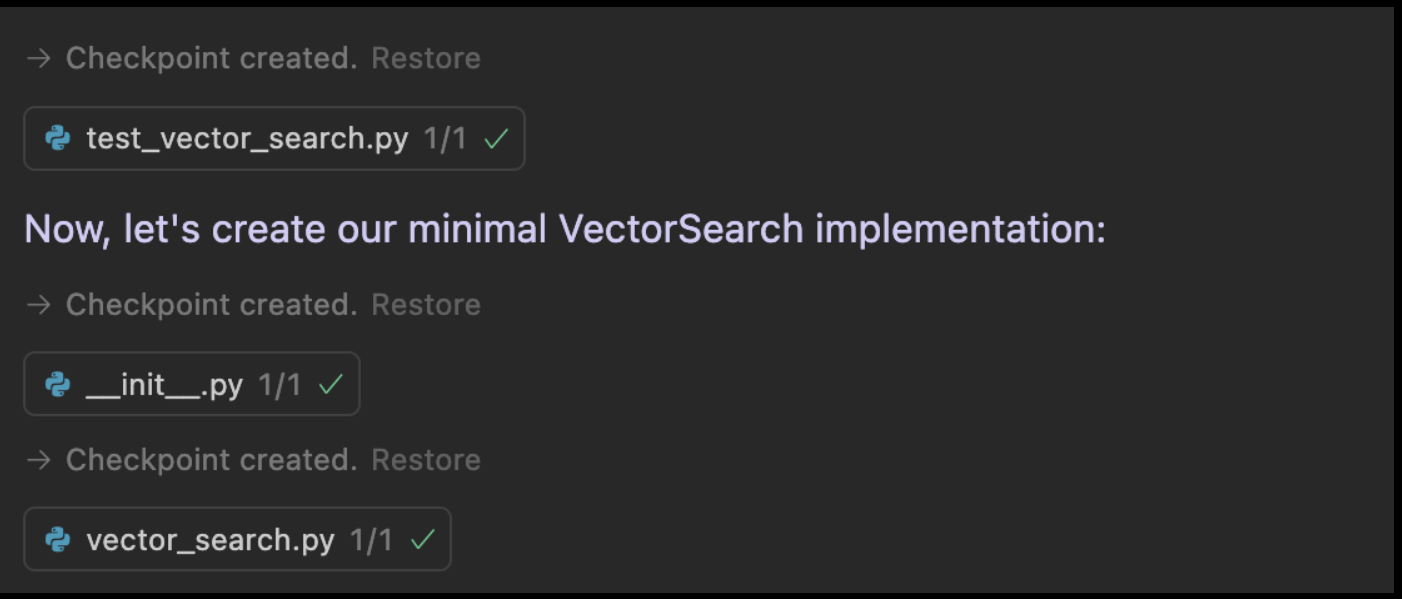

```
Pick up the first tasks from
@250216_initial_planning.md

Start with a v simple end-
to-end test. Get to a v
simple working version, and
then run the test.
```

reference the planning doc

it's time for the AI to start work!



Make a cup of tea while the AI ploughs ahead



It’s installing packages

Let's install the requirements and run the test:

```
$ pip install -r requirements.txt
```

```
Using cached iniconfig-2.0.0-py3-none-any.whl (5.9 kB)
Using cached packaging-24.2-py3-none-any.whl (65 kB)
Using cached annotated_types-0.7.0-py3-none-any.whl (13 kB)
Using cached idna-3.10-py3-none-any.whl (70 kB)
Using cached MarkupSafe-3.0.2-cp311-cp311-macosx_11_0_arm64.
Using cached certifi-2025.1.31-py3-none-any.whl (166 kB)
Using cached h11-0.14.0-py3-none-any.whl (58 kB)
Installing collected packages: typing-extensions, tqdm, snif
arkupSafe, jiter, itsdangerous, iniconfig, idna, h11, distro
kzeug, pytest, pydantic-core, Jinja2, httpcore, anyio, pydan
Successfully installed Jinja2-3.1.5 MarkupSafe-3.0.2 Werkzeu
er-1.9.0 certifi-2025.1.31 click-8.1.8 distro-1.9.0 flask-3.
-3.10 iniconfig-2.0.0 itsdangerous-2.2.0 jiter-0.8.2 numpy-2
pydantic-2.10.6 pydantic-core-2.27.2 pytest-8.3.4 python-dot
nsions-4.12.2
```

check the `requirements.txt` to be sure you like its choices

interrupt, edit, and resume whenever you need to

Pay especially close attention to the tests

tests > test_vector_search.py > ...

Accept ⌘Y Reject ⌘N

```python
import pytest
from src.vector_search import VectorSearch


def test_basic_search():
    # Initialize the search engine
    search_engine = VectorSearch()

    # Add test documents
    test_docs = {
        "doc1": "The quick brown fox jumps over the lazy dog",
        "doc2": "A lazy dog sleeps all day",
        "doc3": "The brown fox is quick and clever",
    }
    search_engine.add_documents(test_docs)

    # Perform a search
    results = search_engine.search("quick fox")

    # Verify results structure
    assert isinstance(results, list)
    assert len(results) == 3  # Should return all documents

    # Check that each result has the expected fields
    for result in results:
        assert isinstance(result, dict)
        assert "id" in result
        assert "text" in result
        assert "score" in result
        assert isinstance(result["score"], float)
        assert 0 <= result["score"] <= 1  # Cosine similarity is be

```

if they're right, then the code probably will be too

and the AI can iterate, fixing code until they pass

Pay especially close attention to the tests

```
$ python -m pytest tests/test_vector_search.py -v
```

```
git:(main) ✗  python -m pytest tests/test_vector_search.py -
========================================= test session sta
platform darwin -- Python 3.11.9, pytest-8.3.4, pluggy-1.5.(
e-AI-London-2025-Feb-18-Cursor/bin/python
cachedir: .pytest_cache
rootdir: /Users/greg/Dropbox/dev/gdconsult_work/Mindstone-AI
plugins: anyio-4.8.0
collected 1 item

tests/test_vector_search.py::test_basic_search PASSED

========================================== 1 passed in 2.4
gdconsult_work__Mindstone-AI-London-2025-Feb-18-Cursorgdcons
 git:(main) ✗
```

Great! We've successfully:

You can drive faster with guardrails

git commit *often*

Now that we have our basic implementation and test in place, let's install the requirements and run the test. First, we should create a `.env` file for our OpenAI API key:

it overwrote my .env file!

fortunately, that's easy to undo

Continue - keep the planning doc up-to-date, start fresh conversations occasionally, layer in requirements

```
Pick up from
@250216_initial_planning.md

Let's switch from command-
line to a web application.

Ask questions, make
proposals, don't make
changes yet.
```

use the planning document to re-establish context

new features:

back into information-gathering/ alignment mode

Resources

Conversation modes

Information-gathering/alignment	[Describe new feature.] Ask me questions, help me think through options & trade-offs, make a proposal, don't make changes.
Update the planning doc	Use this structure: goals/context, operating principles, prioritised hierarchical actions/stages (marked as TODO vs DONE)
Write the tests first	Propose the tests you plan to write, but don't write them yet. Let's think through the edge-cases we want to cover.
Write the code, run the tests, fix the code	Keep changes minimal, and relevant to the task at hand.
Help me debug this	Gather information, read through the code, look at this output, check the database, run the tests, try it yourself, etc. Think through possible causes or places to investigate. What would help us get to the bottom of this?

https://www.makingdatamistakes.com/making-tea-while-ai-codes-a-practical-guide-to-2024s-development-revolution/

https://github.com/gregdetre/Mindstone-AI-London-2025-Feb-18-Cursor

https://www.gregdetre.com/consulting/

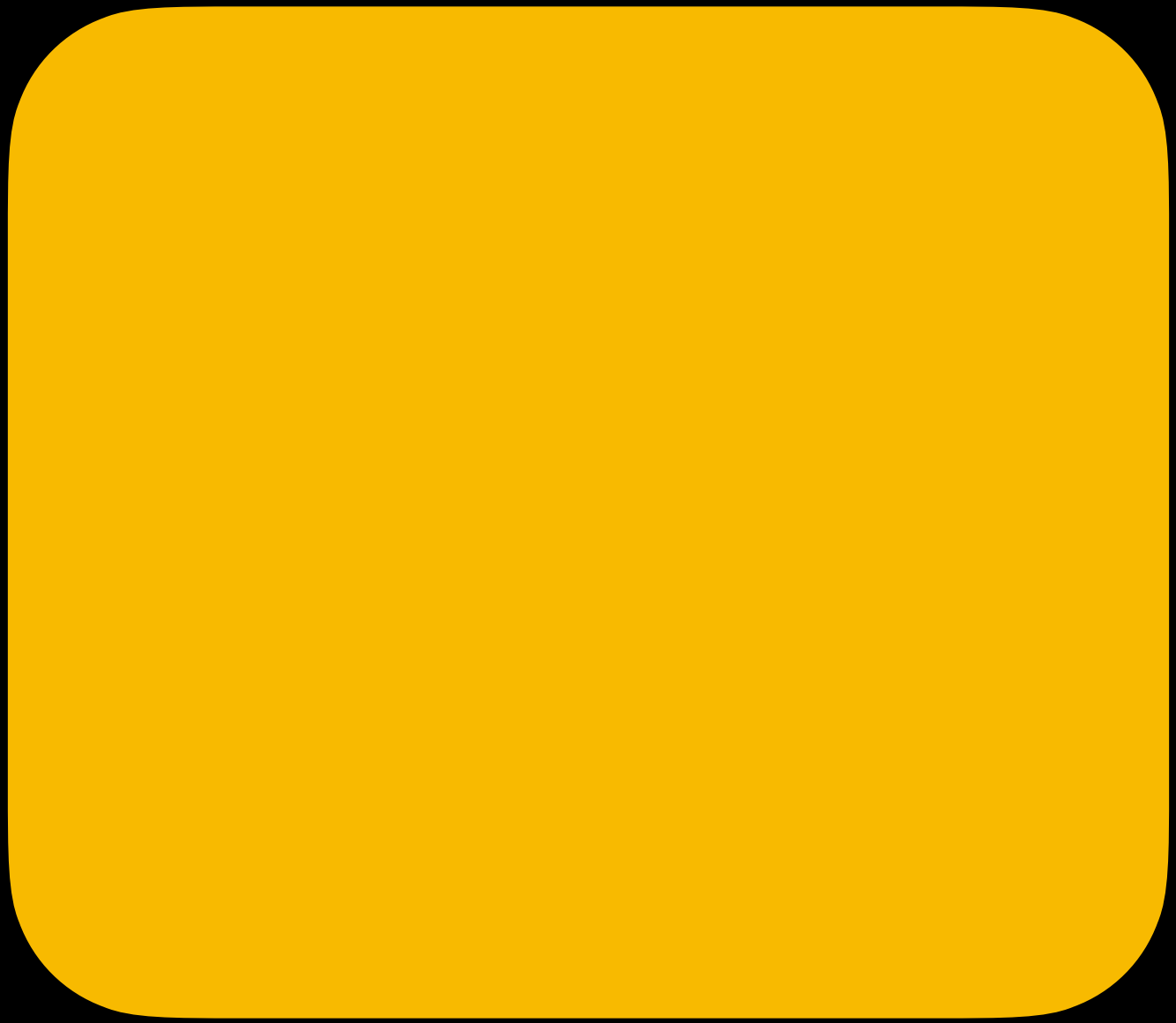
https://www.makingdatamistakes.com/#/portal/signup

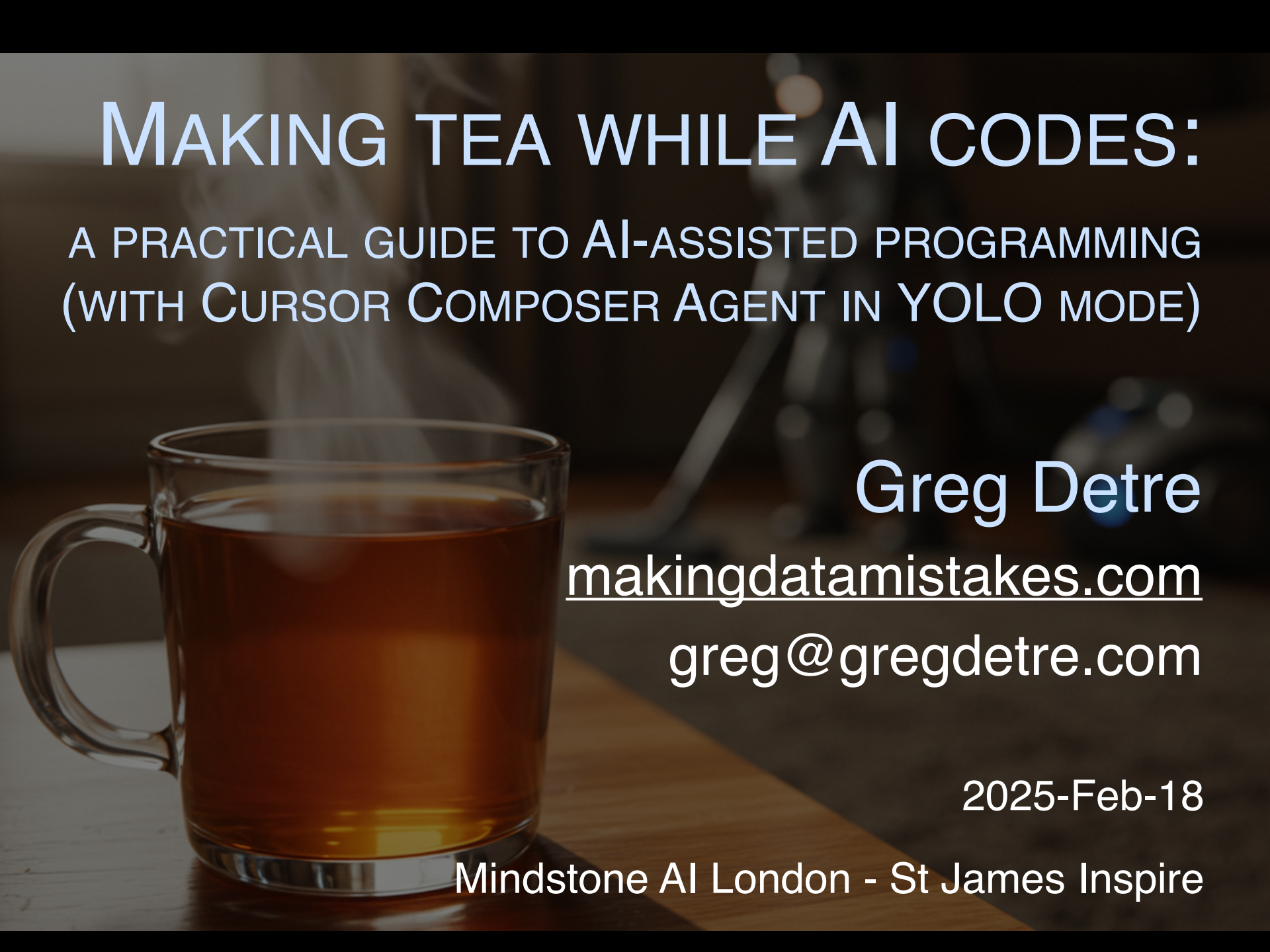
https://www.linkedin.com/in/gregdetre/

The End

XXX

XXX





Making tea while AI codes:

A practical guide to AI-assisted programming (with Cursor Composer Agent in YOLO mode)

Greg Detre

makingdatamistakes.com

greg@gregdetre.com

2025-Feb-18

Mindstone AI London - St James Inspire

Ask me about…

[goals]

[conversation modes]

[resources]

How do I get buy-in from my team?

This is fine for a side-project, but what about production code?

What's going to happen to software engineering jobs?

How can i handle bigger projects & legacy code?

What about when things go wrong?

Where *doesn't* this work well?

What can we expect from 2025?

… or anything else!