26th June 2024

Bragg Building (GR.25)

16.00 – 18:00

Guest Speakers

Richard Mann & Danny Wood

From Animal Insights to Industry Heights: Active Learning & Engineering ML at Scale

Today's schedule

Time	Topic
16:00	Dr R Mann Active & Retro-Active Learning in Animal Behaviour
16:50	Pizza
17:10	Dr D Wood From Academia to Industry: Software Engineering for Machine Learning at Scale
18:00	Pub?

AI²

Dr Richard Mann

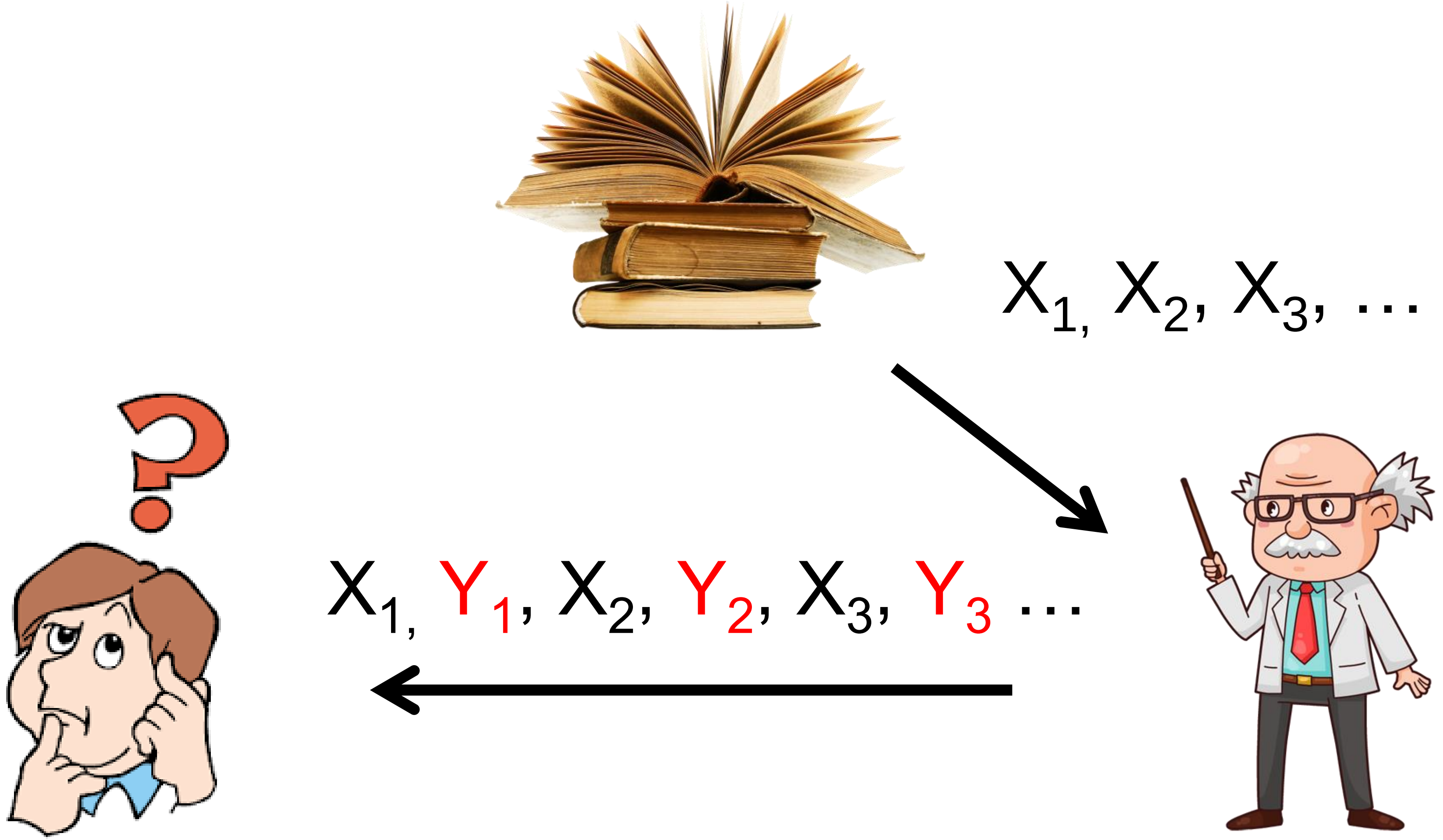
Associate Professor
School of Mathematics
University of Leeds

Active & Retroactive

Learning for animal behaviour

Richard Mann, School of Mathematics

Supervised learning


$X_{1,}\ X_2,\ X_3,\ \ldots$
$X_{1,}\ Y_1,\ X_2,\ Y_2,\ X_3,\ Y_3\ \ldots$

What is active learning?

~ Bayesian optimisation

What is active learning?

- Where to look next…
- What to do next…

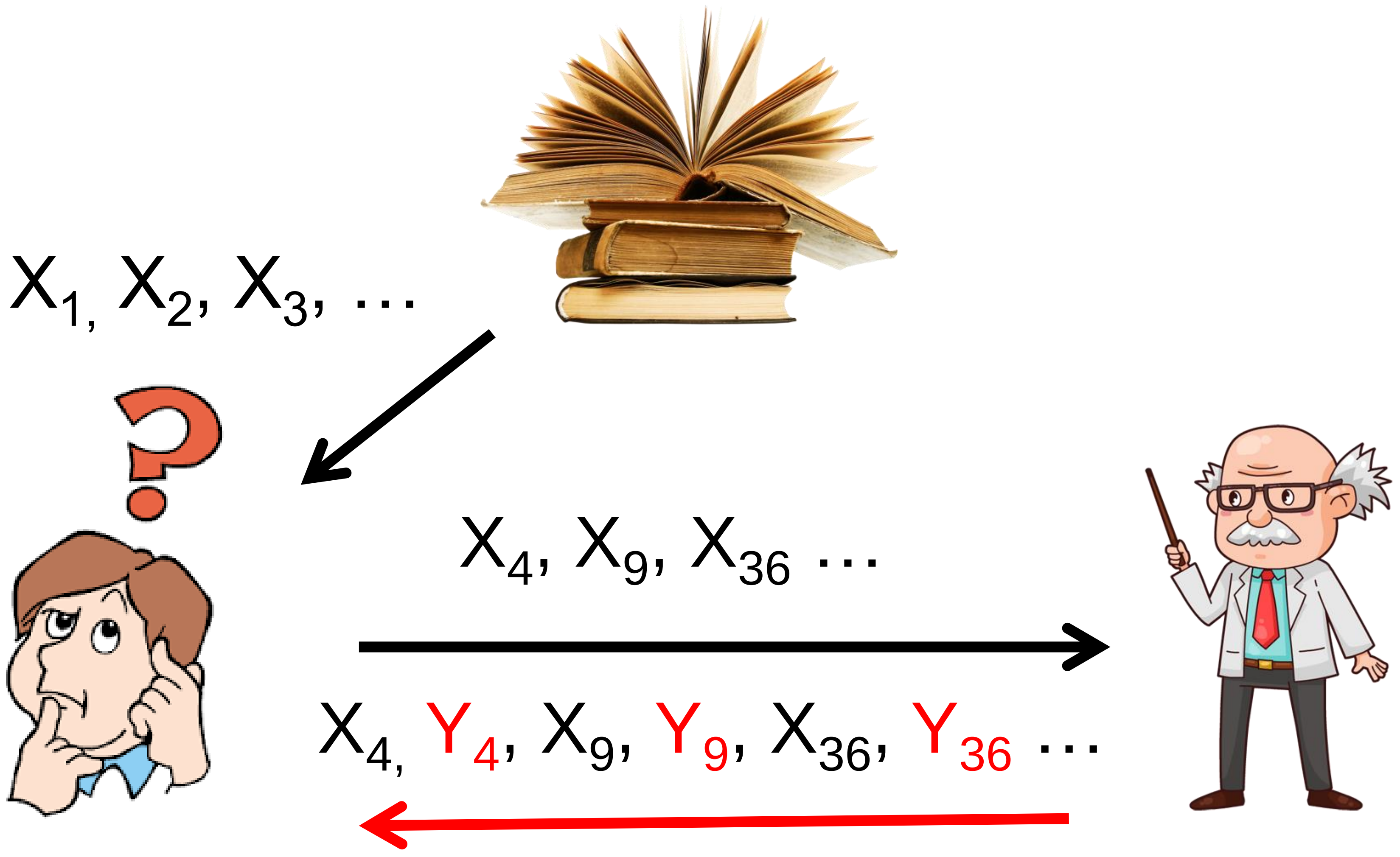
What is active learning?

Each shot:

- Potential hit
- Learn

Winners don't fire randomly!

What is active learning?


$X_1, X_2, X_3, \ldots$
$X_4, X_9, X_{36} \ldots$
$X_4, Y_4, X_9, Y_9, X_{36}, Y_{36} \ldots$

Why active learning?

- Labels are expensive
- Mimics reality
- Too much data!

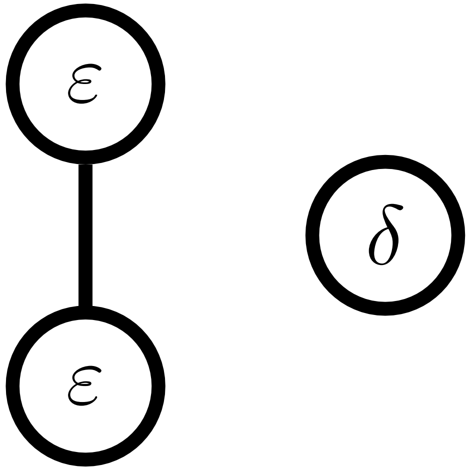
Active learning with utility

Utility:

- Win/lose
- # of hits

Lookahead

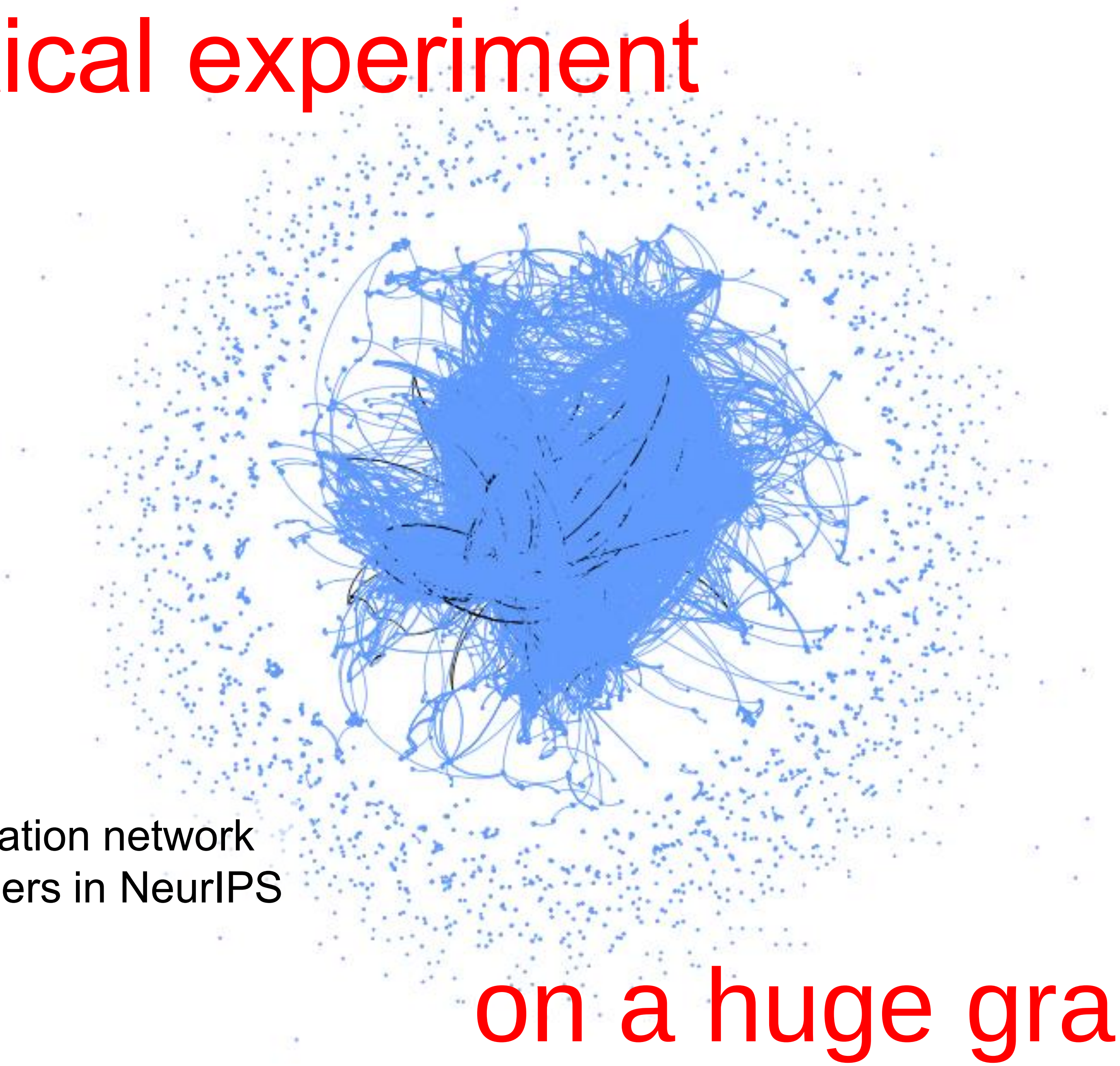
Unlike the simple greedy one-step lookahead policy, two- and more-step lookahead leads to nontrivial choices. Let δ ≥ ε, and consider two evaluations. Which point should we choose first?



- one-step: $\varepsilon + \delta$
- two-step: $2\varepsilon + (1 - \varepsilon)\delta$
- difference: $\varepsilon(1 - \delta) > 0$

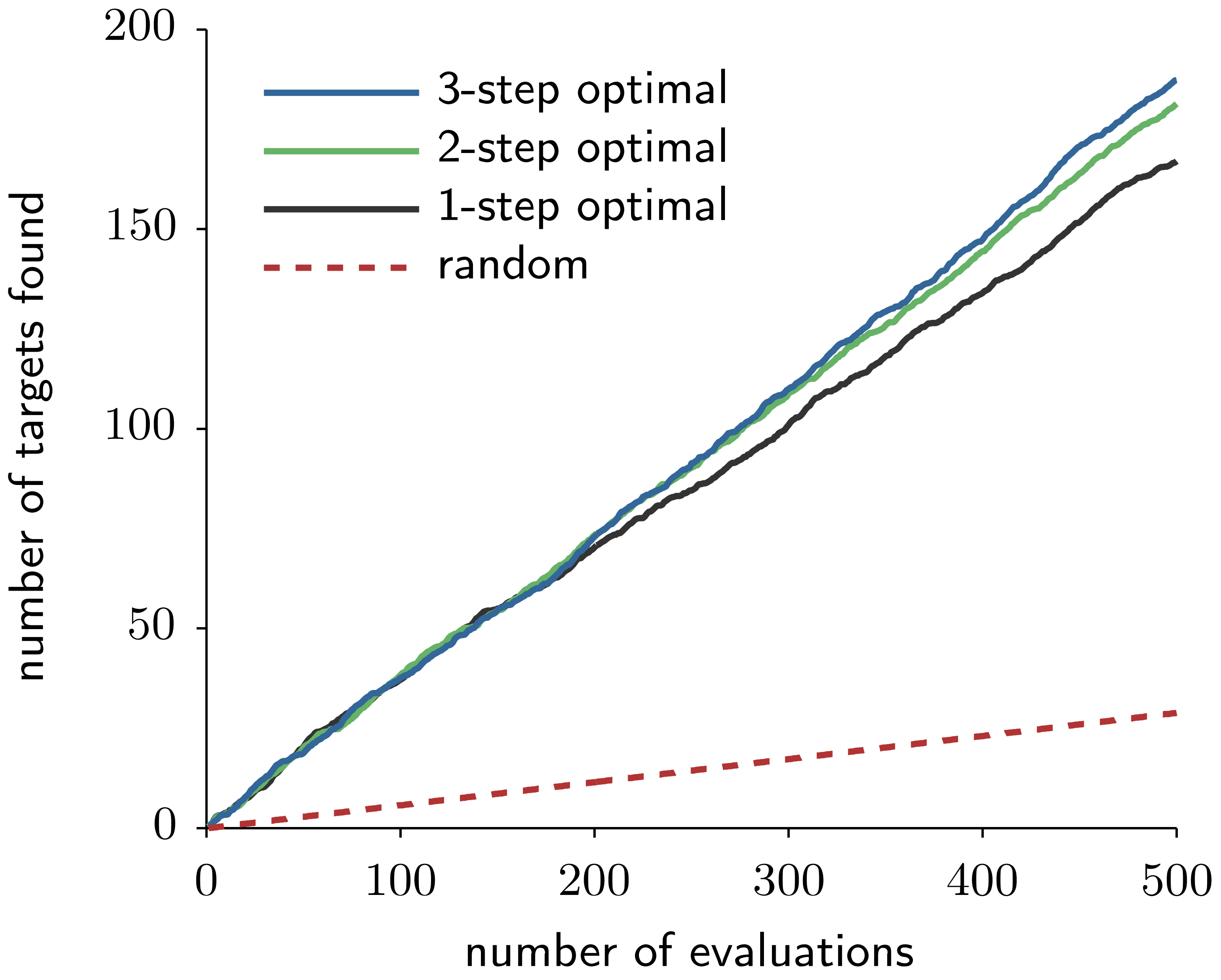
Choosing the low-probability node is always better!

Practical experiment



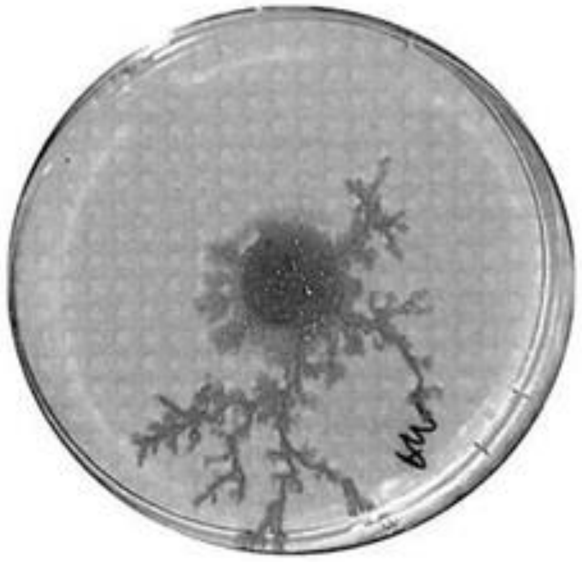
CiteSeer^x citation network
Targets: papers in NeurIPS

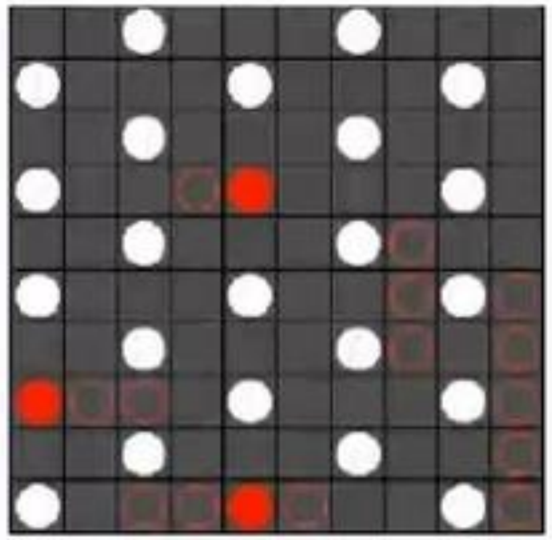
on a huge graph

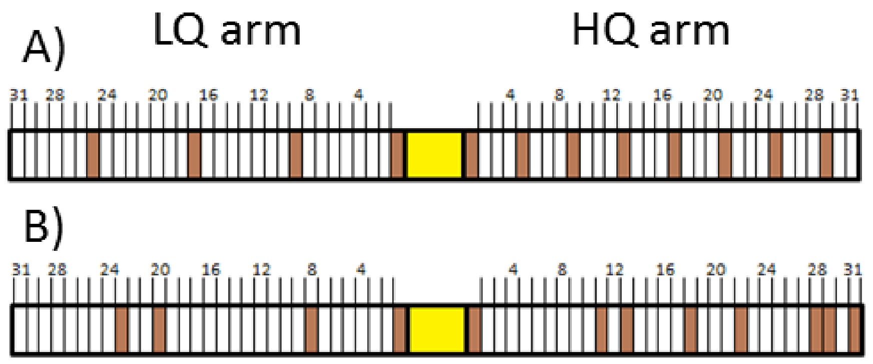


Garnett *et al.* Bayesian Optimal Active Search and Surveying, ICML 2012

What does a slime mould want?







Garnett *et al.*, *ICML* 2012

Zabzina *et al. PLoS Comp. Biol.* 2014

Reid *et al*. *J R Soc Interface* 2016

Retroactive learning

Retroactive learning

Active learning:

where *should I* look?

Retroactive learning

Active learning:

where *should I* look?

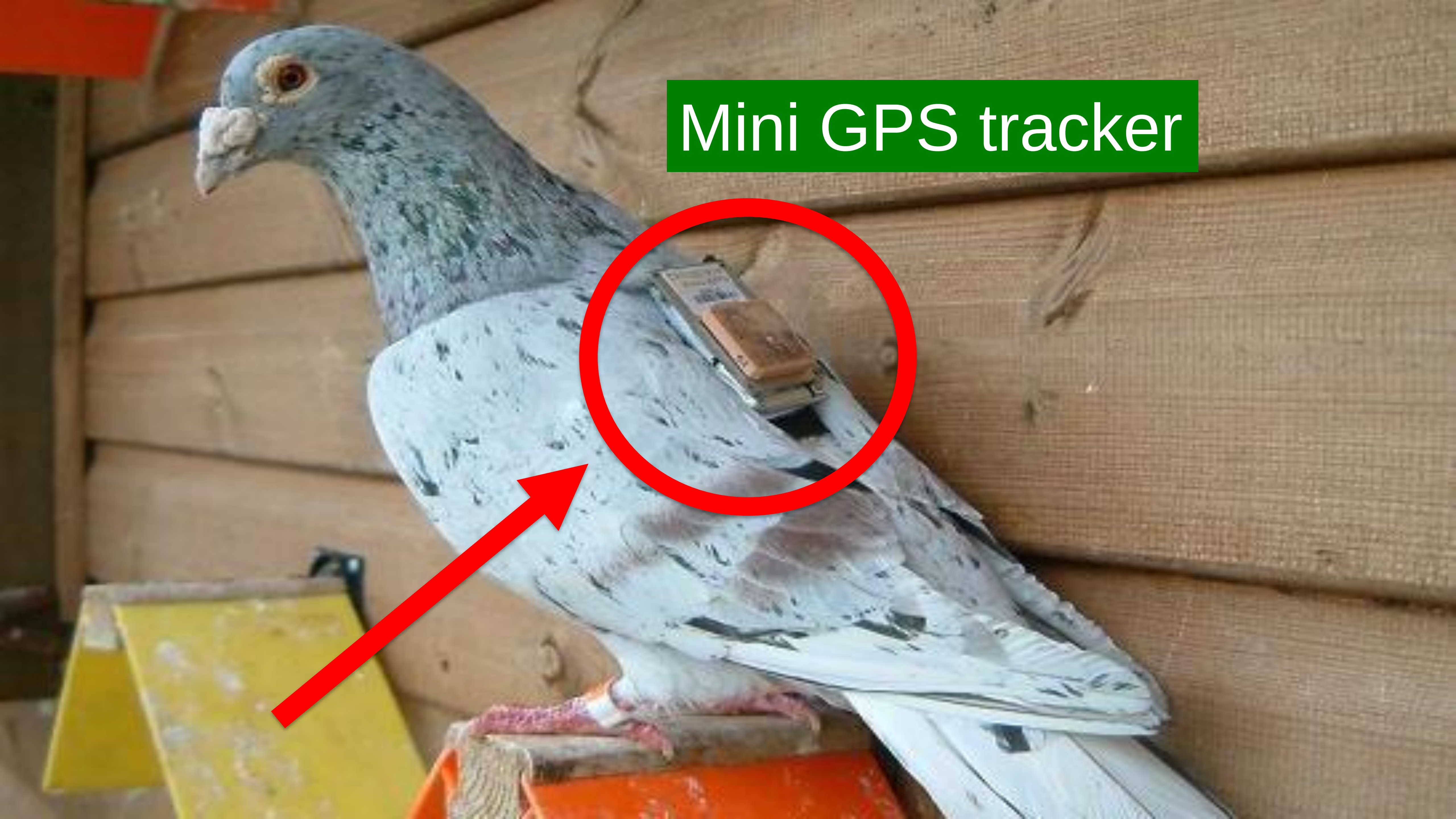
Retroactive learning:

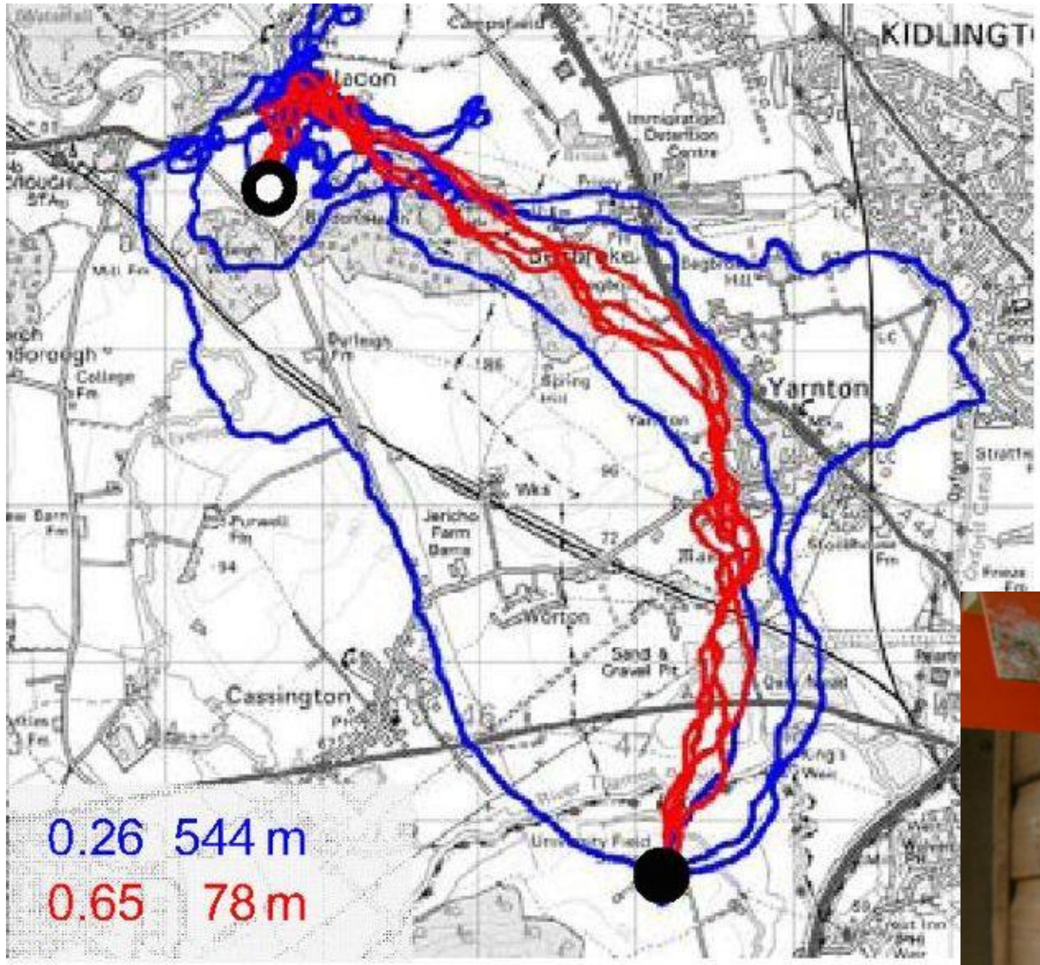
where *should I have* looked?

This is a pigeon

This is a pigeon

Mr Grizzle


Mini GPS tracker

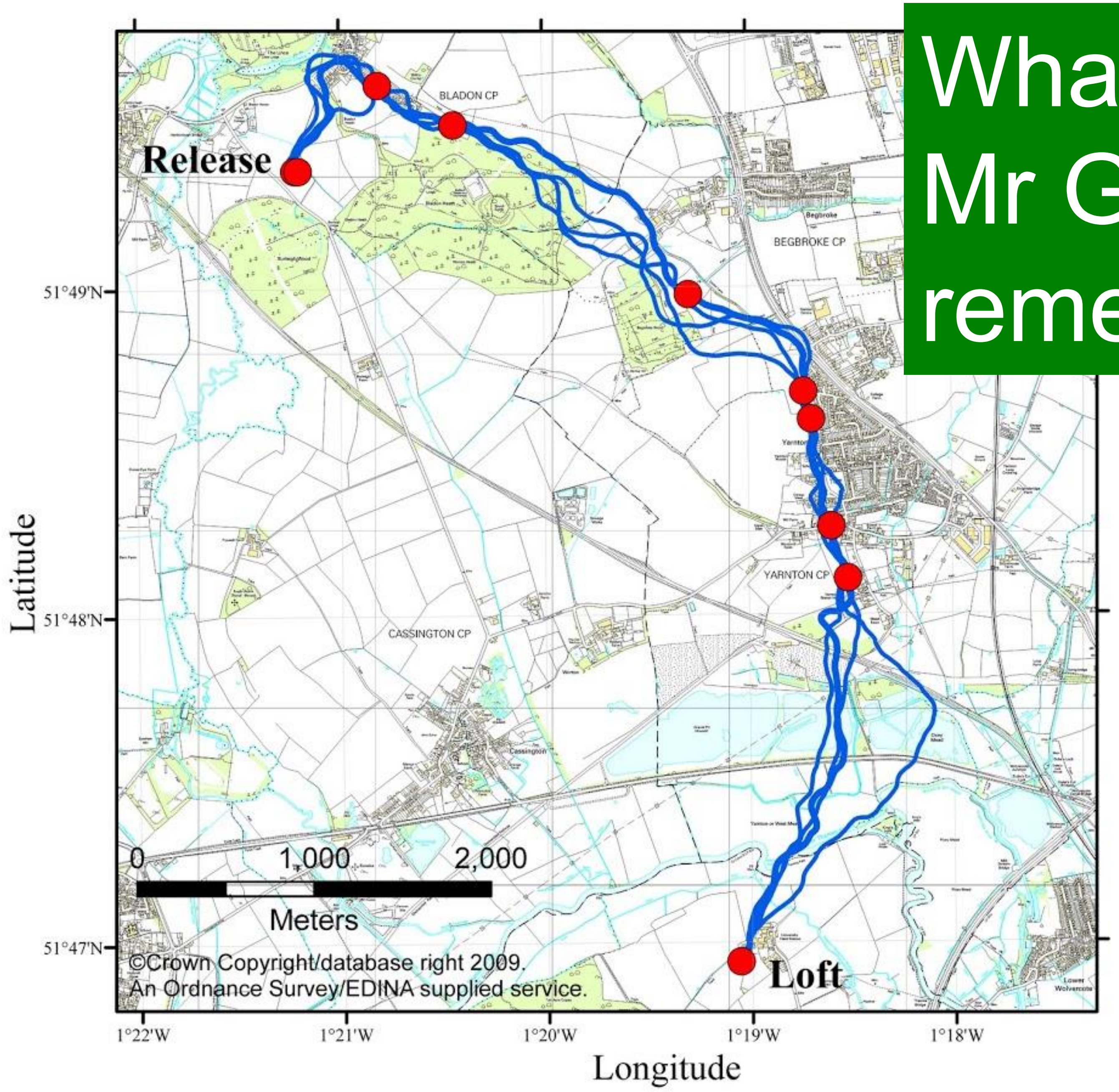


Meade *et al*: ‘Homing pigeons develop local route stereotypy’

Proc. Roy. Soc. B 2005

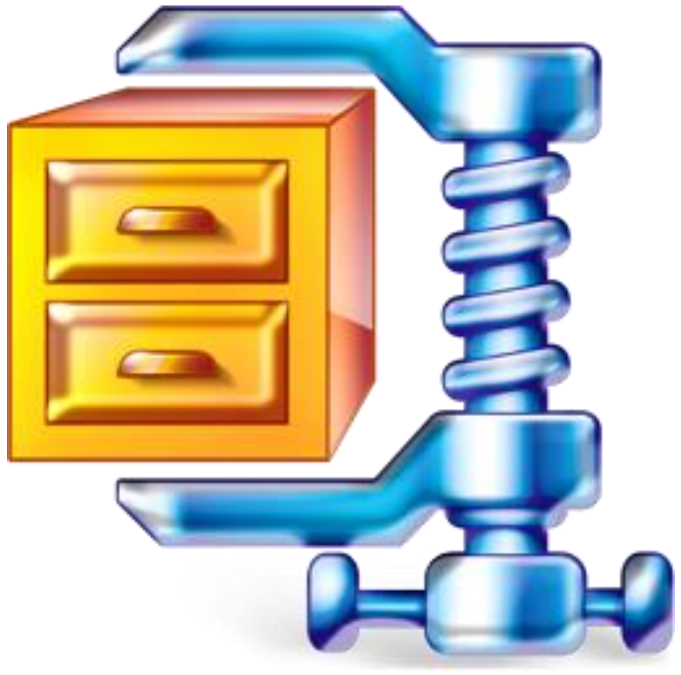
Mr Grizzle learns to fly home this way

What does Mr Grizzle remember?



Mann *et al.* Objectively identifying landmark use and predicting flight trajectories of the homing pigeon using Gaussian processes
J R Soc Interface 2011

It’s a compression algorithm



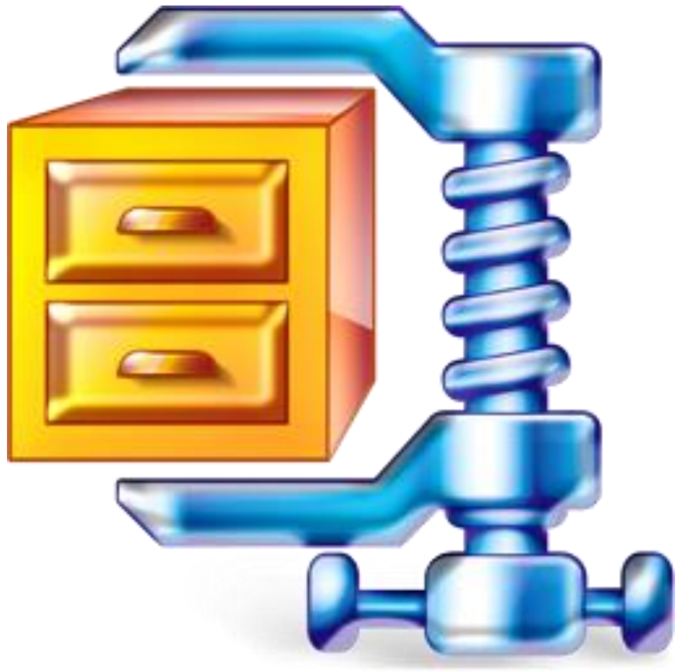
Where should I look?

- how best to learn

Where should I have looked

- how best to store

It’s a compression algorithm



Where should I look?

- how best to learn

Where should I have looked

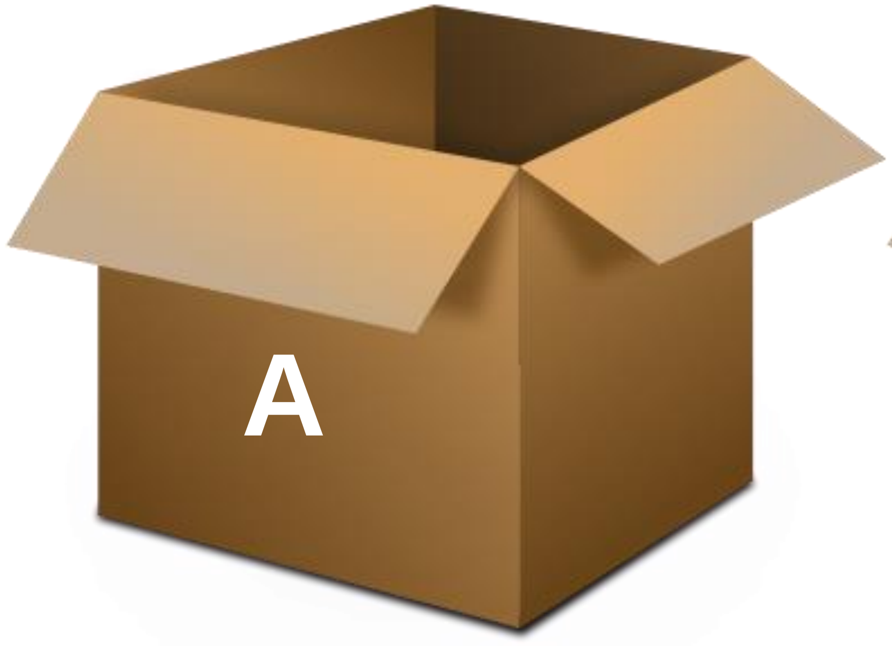
- how best to store

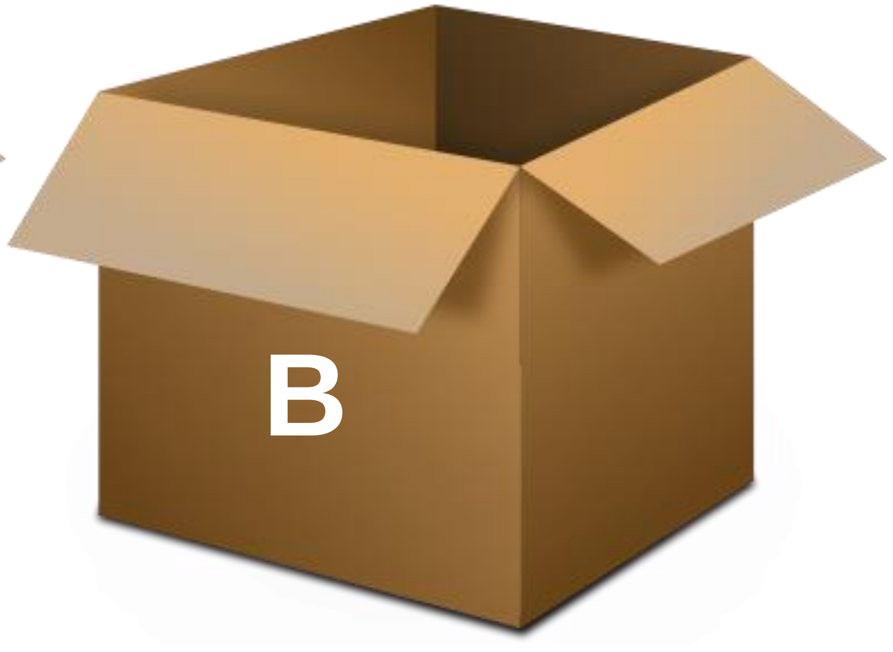
Brains are costly!

Parallel active learning

£20

£15

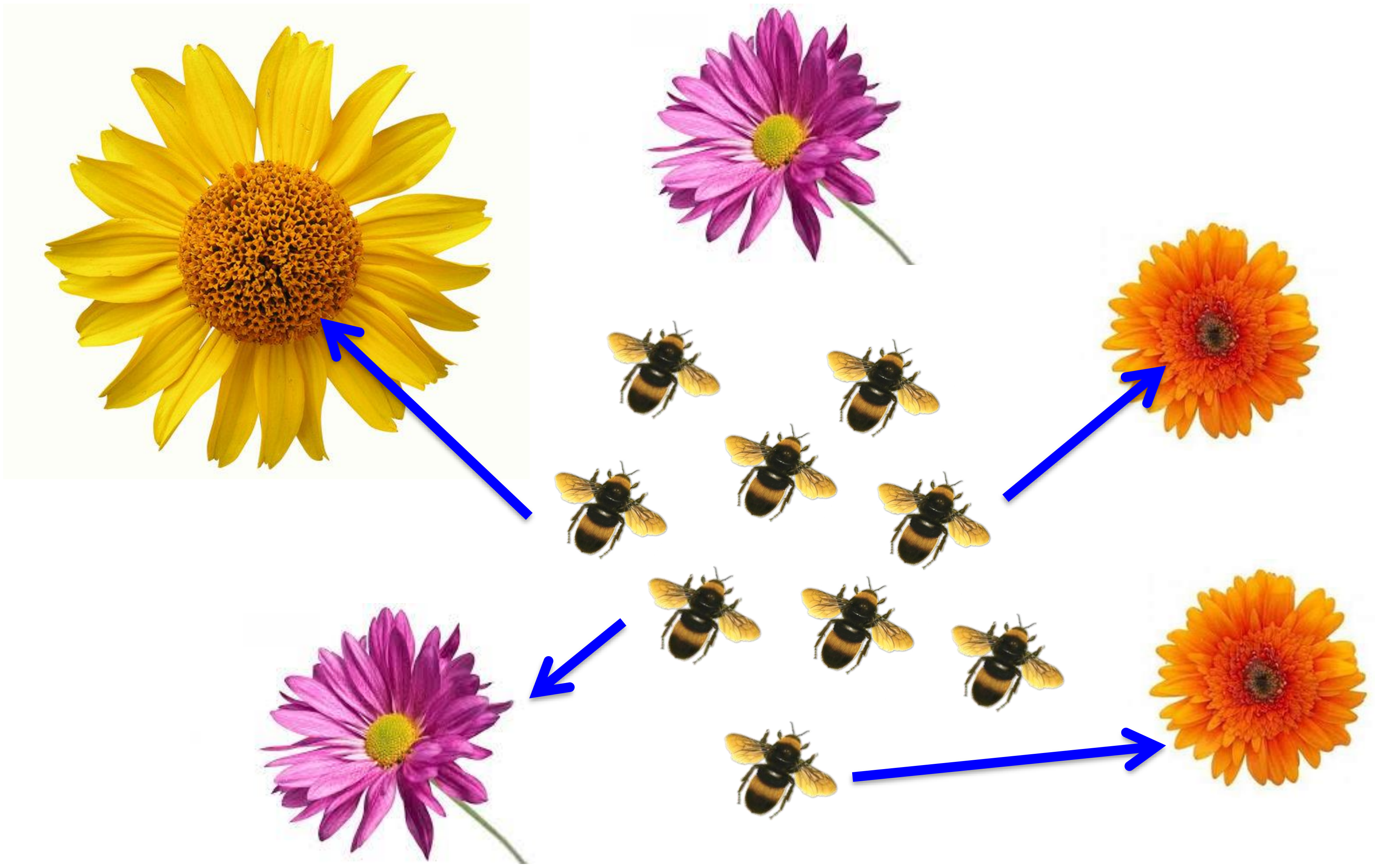


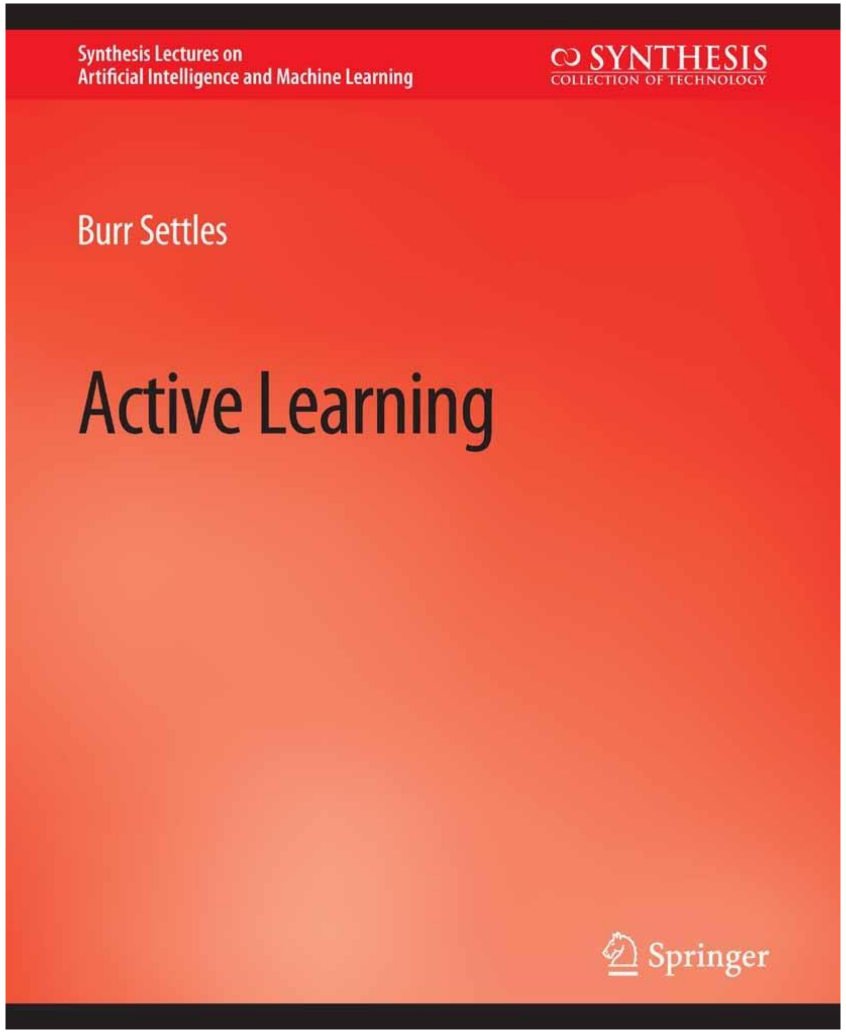


Learning together: Bees, termites, ants…

Single bee chooses best option

Many bees divide their efforts



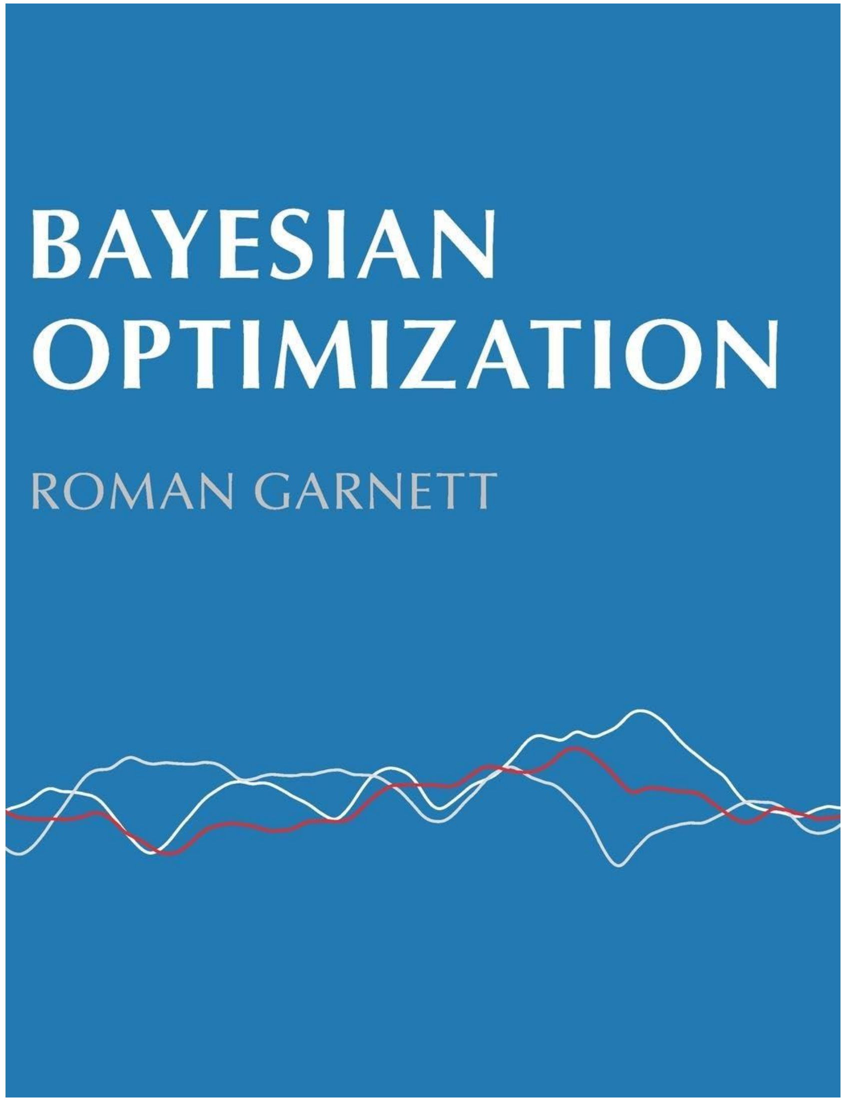

Synthesis Lectures on
Artificial Intelligence and Machine Learning

SYNTHESIS
COLLECTION OF TECHNOLOGY

Burr Settles

Active Learning

Springer


BAYESIAN OPTIMIZATION

ROMAN GARNETT

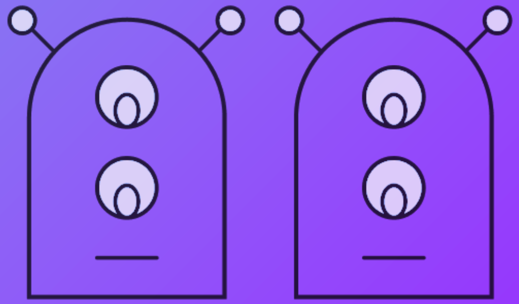
Pizza Time!

Dr Danny Wood

ML Ops Engineer
Fuzzy Labs

fuzzy labs

From Academia to Industry:
Software Engineering for Machine Learning at Scale



Introduction

A PhD gives you lots of skills that are very valuable in industry

But for jobs involving programming, there are lots of things that it doesn't teach you

In academia and industry, code is written in very different ways, for very different reasons

- Academia has an emphasis on experimentation and being able to change things quickly
- Industry has an emphasis on building robust systems, and more close collaboration on codebases

📺 Introduction

In this talk:

- I’ll talk about the transition from academia to industry
- The ways that it’s made me grow as a programmer
- The tools and systems that I’ve learnt
- Which ones I wish I’d learnt about sooner:
    - Which I think are useful in academia
    - Which are worth getting experience with before moving to industry

📺 Who am I?

- PhD in machine learning
  - Looking at memory in recurrent neural networks
  - Mostly theory, some experiments
- Postdocs in ensemble learning and explainability
  - A mix of applied and theoretical research
- Now an MLOps engineer at Fuzzy Labs
  - Manchester based MLOps start-up/consultancy
  - Deploying production-ready machine learning systems
  - Some *very* applied research, mostly engineering

🧑‍💼 Moving to Industry

- How easy/hard it is to find a job is based on a lot of factors outside of your control
- But there are also a factors that you can control
  - Look for opportunities to add to your CV
  - Give yourself plenty of time
  - Use your contacts
- The first job out of academia will be the hardest to get
- Be prepared to learn a lot of stuff very quickly

MANCHESTER 1824
The University of Manchester

fuzzy labs

🖥 Coding at Fuzzy Labs

- Projects tend to have 2-5 people working on them simultaneously
- A lot of projects involving building cloud-based machine learning systems (a lot of LLMs!)
- Fuzzy Labs really care about code quality
- Like any tech company, knows long term success is dependent on consistent quality
- We do a lot of stuff that is best practice, but my experience won’t be universal

🖥 Tools and Systems

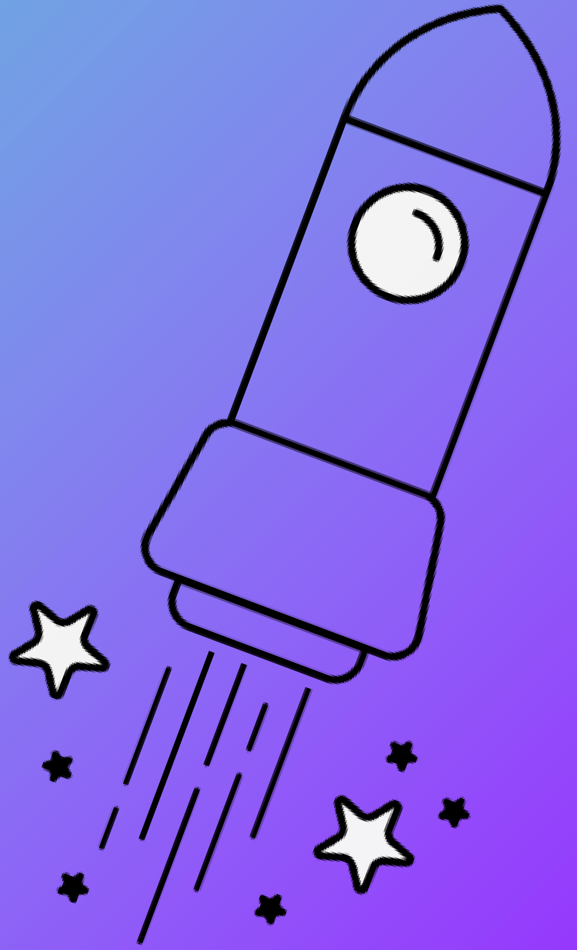
- Coding as a team activity
  - Version control and code review
  - Virtual Environments
- Good code by default
  - Pre-commit hooks
  - Typing
  - Testing
- Finding problems quickly
  - Debugging

pytest

Pre-Commit

fuzzy labs

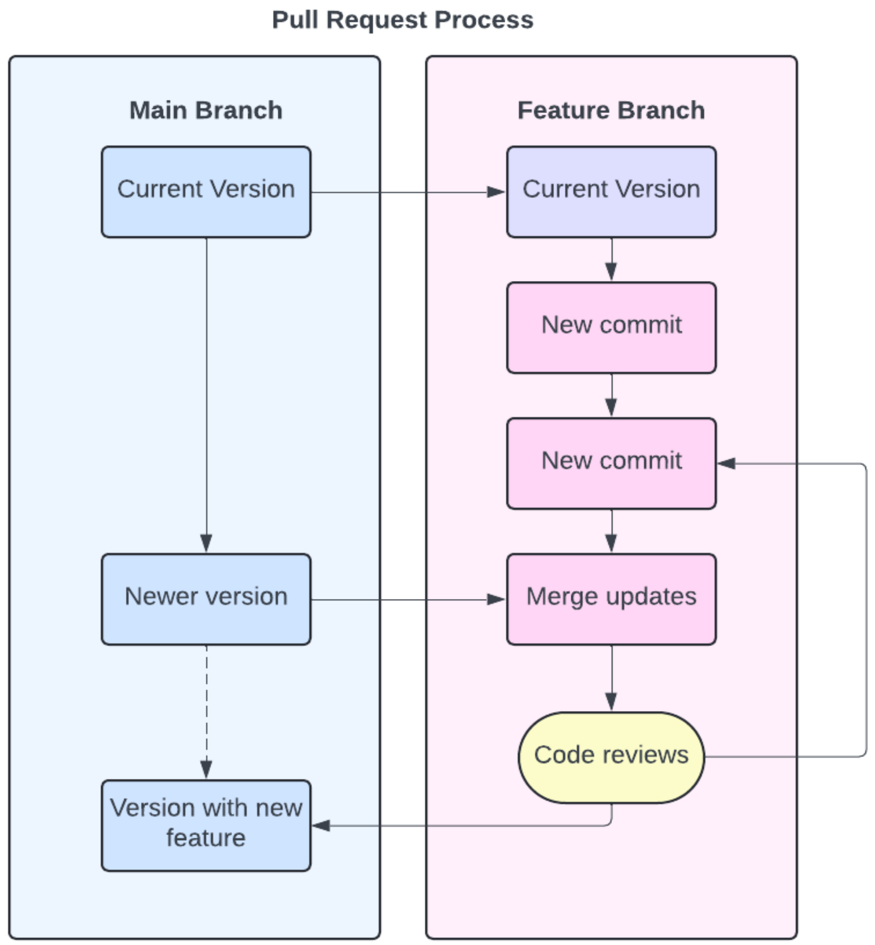
Coding as a team activity



fuzzy labs

Version Control

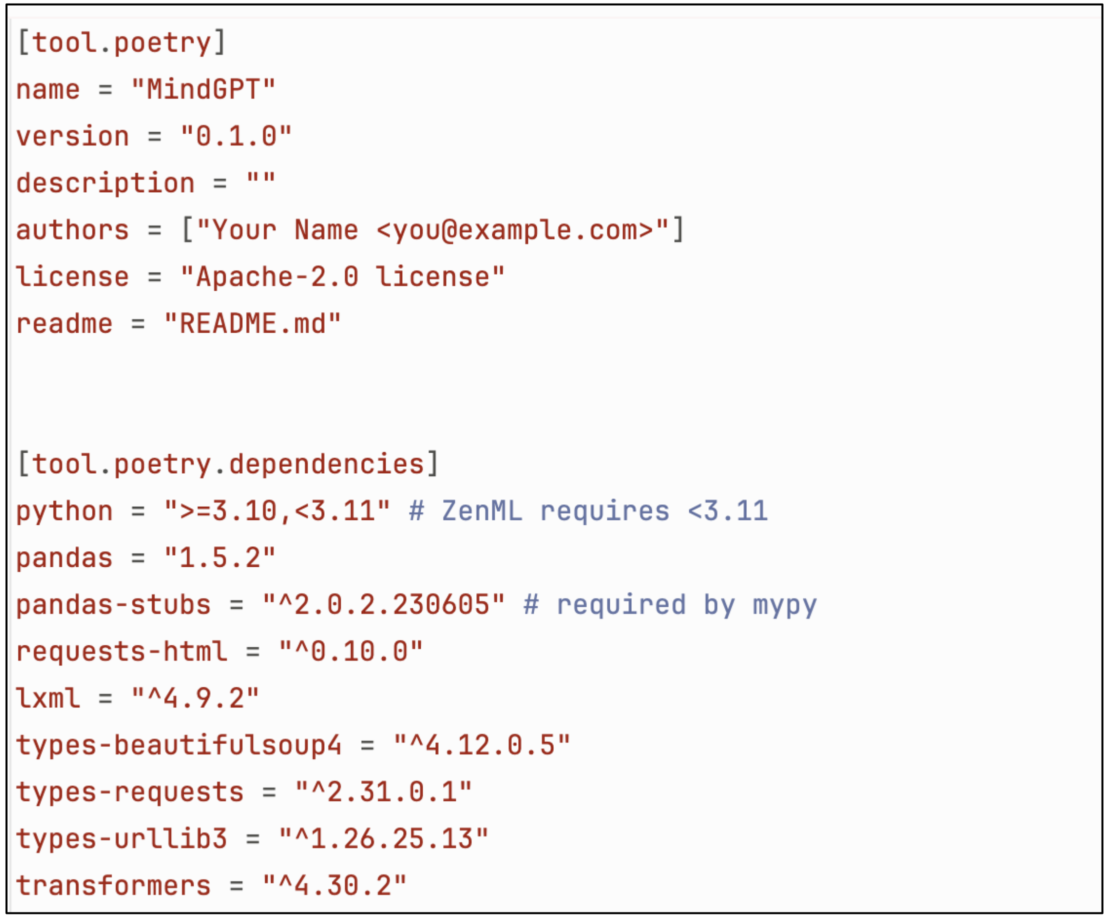
- All changes are tracked with git/Github
- Changes are done in separate feature branches
- Adds friction to small changes, but allows for systematically building larger systems



fuzzy labs

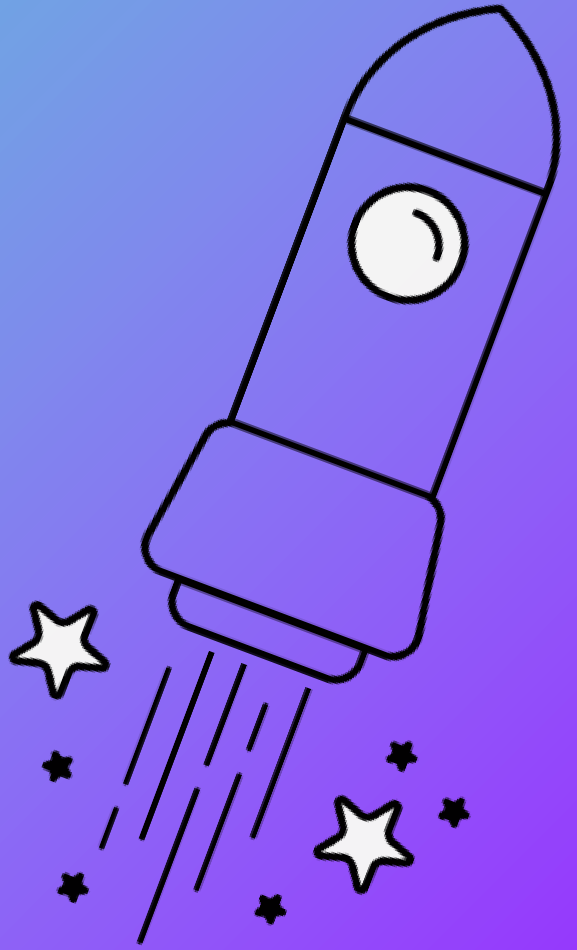
Virtual Environments

- We want all engineers to be working with the same versions of each library
- We have a shared configuration file specifying what tools/libraries/settings we want in the environment
- An automatically generated poetry.lock file that the environment is built from
- This defines not just the versions of explicit dependencies, but also 2nd order ones, 3rd order ones, etc



```toml
[tool.poetry]
name = "MindGPT"
version = "0.1.0"
description = ""
authors = ["Your Name <you@example.com>"]
license = "Apache-2.0 license"
readme = "README.md"


[tool.poetry.dependencies]
python = ">=3.10,<3.11" # ZenML requires <3.11
pandas = "1.5.2"
pandas-stubs = "^2.0.2.230605" # required by mypy
requests-html = "^0.10.0"
lxml = "^4.9.2"
types-beautifulsoup4 = "^4.12.0.5"
types-requests = "^2.31.0.1"
types-urllib3 = "^1.26.25.13"
transformers = "^4.30.2"
```

Good Code by Default

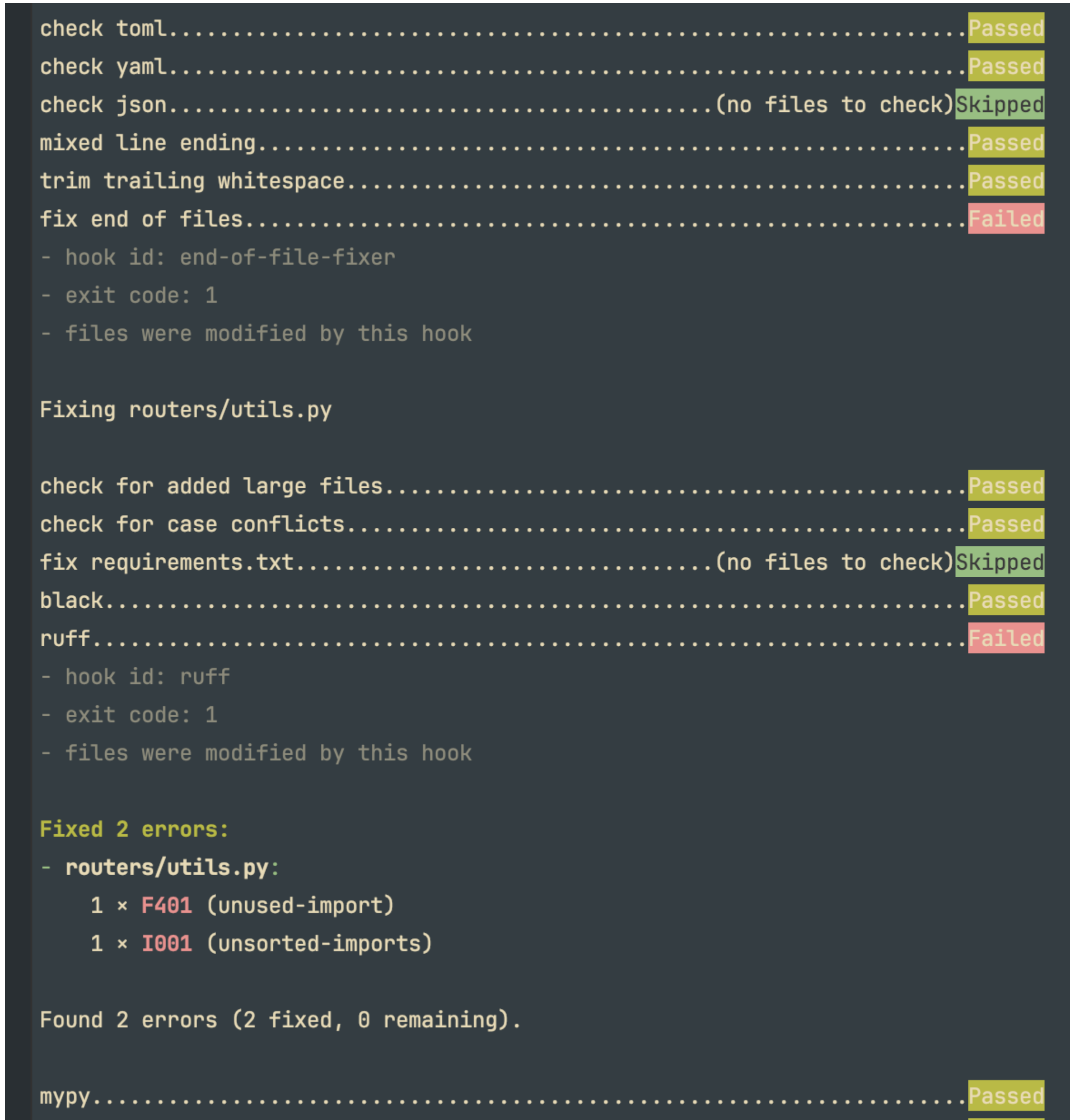


fuzzy labs

Pre-Commit Hooks

- Pre-commit hooks do automatic code quality checks before letting you commit your code
- They can check for
  - Code formatting (whitespace, line length etc)
  - Comments and docstrings
  - Semantic errors
  - Accidentally committing keys/passwords or large files
  - Type errors
  - Typos
- Setting up pre-commit hooks will instantly improve your code quality!

Pre-Commit Hooks

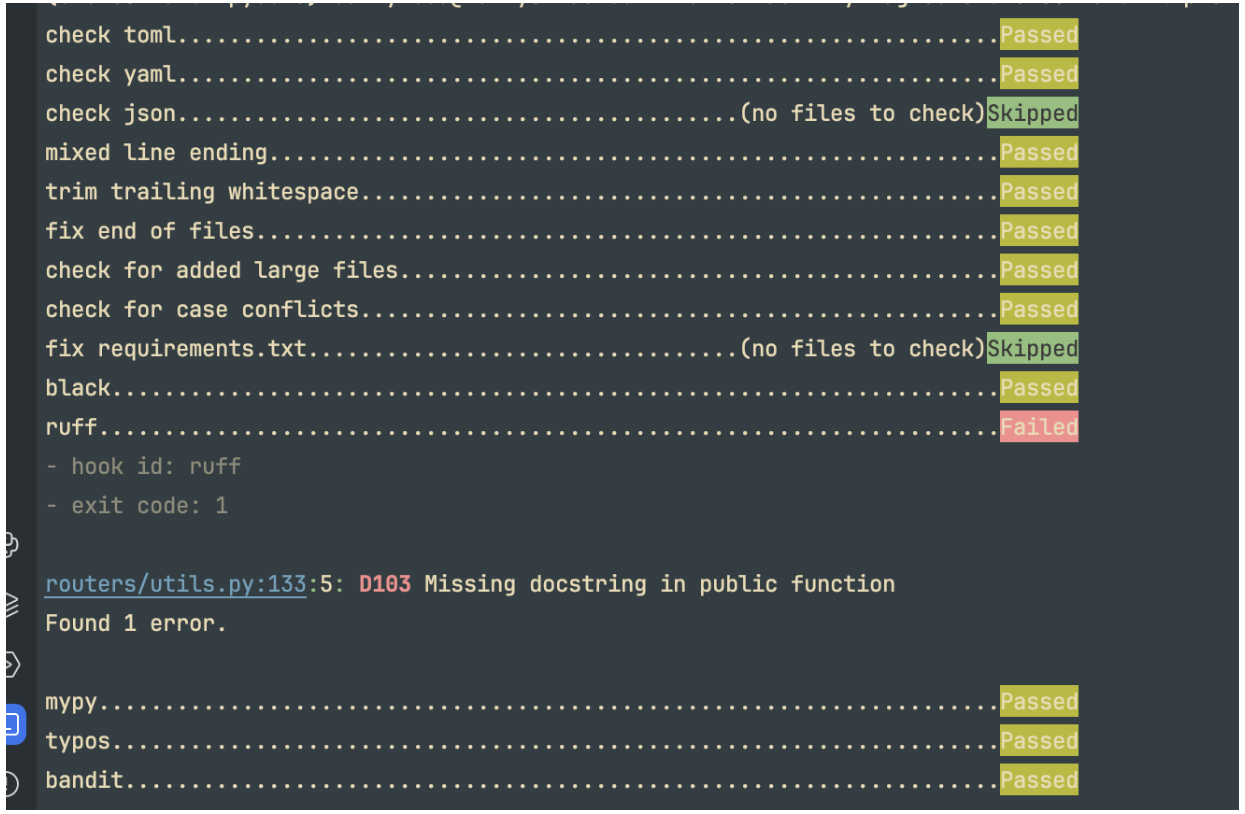


```
check toml...............................................................Passed
check yaml...............................................................Passed
check json..........................................(no files to check)Skipped
mixed line ending........................................................Passed
trim trailing whitespace.................................................Passed
fix end of files.........................................................Failed
- hook id: end-of-file-fixer
- exit code: 1
- files were modified by this hook

Fixing routers/utils.py

check for added large files..............................................Passed
check for case conflicts.................................................Passed
fix requirements.txt................................(no files to check)Skipped
black....................................................................Passed
ruff.....................................................................Failed
- hook id: ruff
- exit code: 1
- files were modified by this hook

Fixed 2 errors:
- routers/utils.py:
    1 × F401 (unused-import)
    1 × I001 (unsorted-imports)

Found 2 errors (2 fixed, 0 remaining).

mypy.....................................................................Passed
```

For minor and cosmetic issues, they will fix your code for you automatically

fuzzy labs

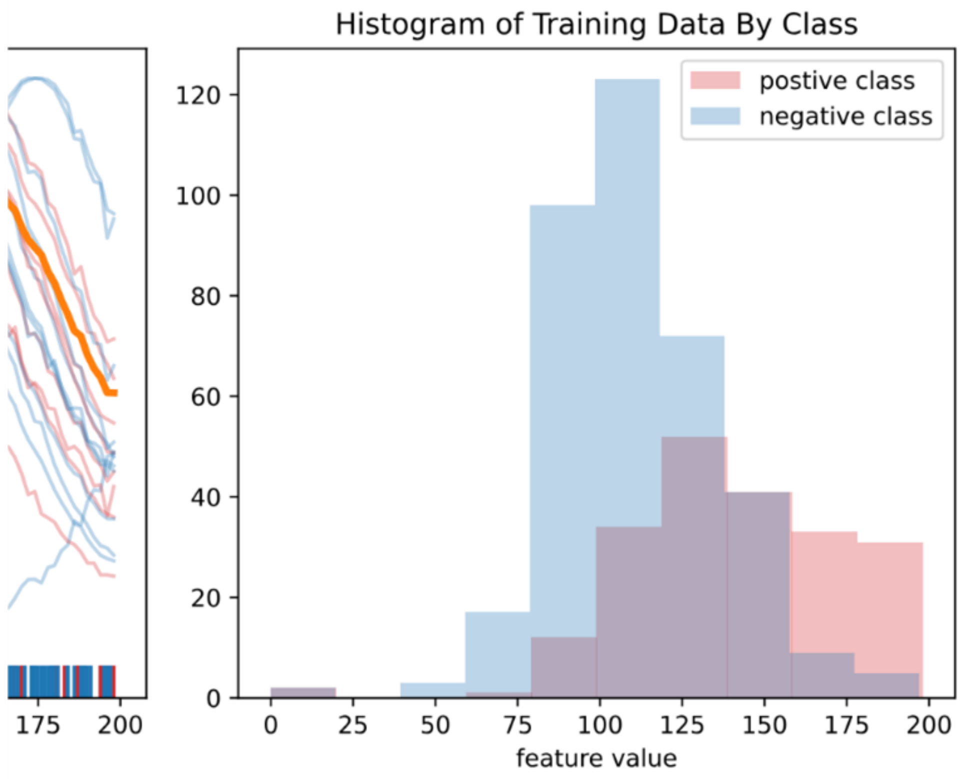
Pre-Commit Hooks



```
check toml...............................................................Passed
check yaml...............................................................Passed
check json..........................................(no files to check)Skipped
mixed line ending........................................................Passed
trim trailing whitespace.................................................Passed
fix end of files.........................................................Passed
check for added large files..............................................Passed
check for case conflicts.................................................Passed
fix requirements.txt................................(no files to check)Skipped
black....................................................................Passed
ruff.....................................................................Failed
- hook id: ruff
- exit code: 1

routers/utils.py:133:5: D103 Missing docstring in public function
Found 1 error.

mypy.....................................................................Passed
typos....................................................................Passed
bandit...................................................................Passed
```

For issues which it can't fix automatically, it will fail and tell you how to fix it

Pre-Commit Hooks

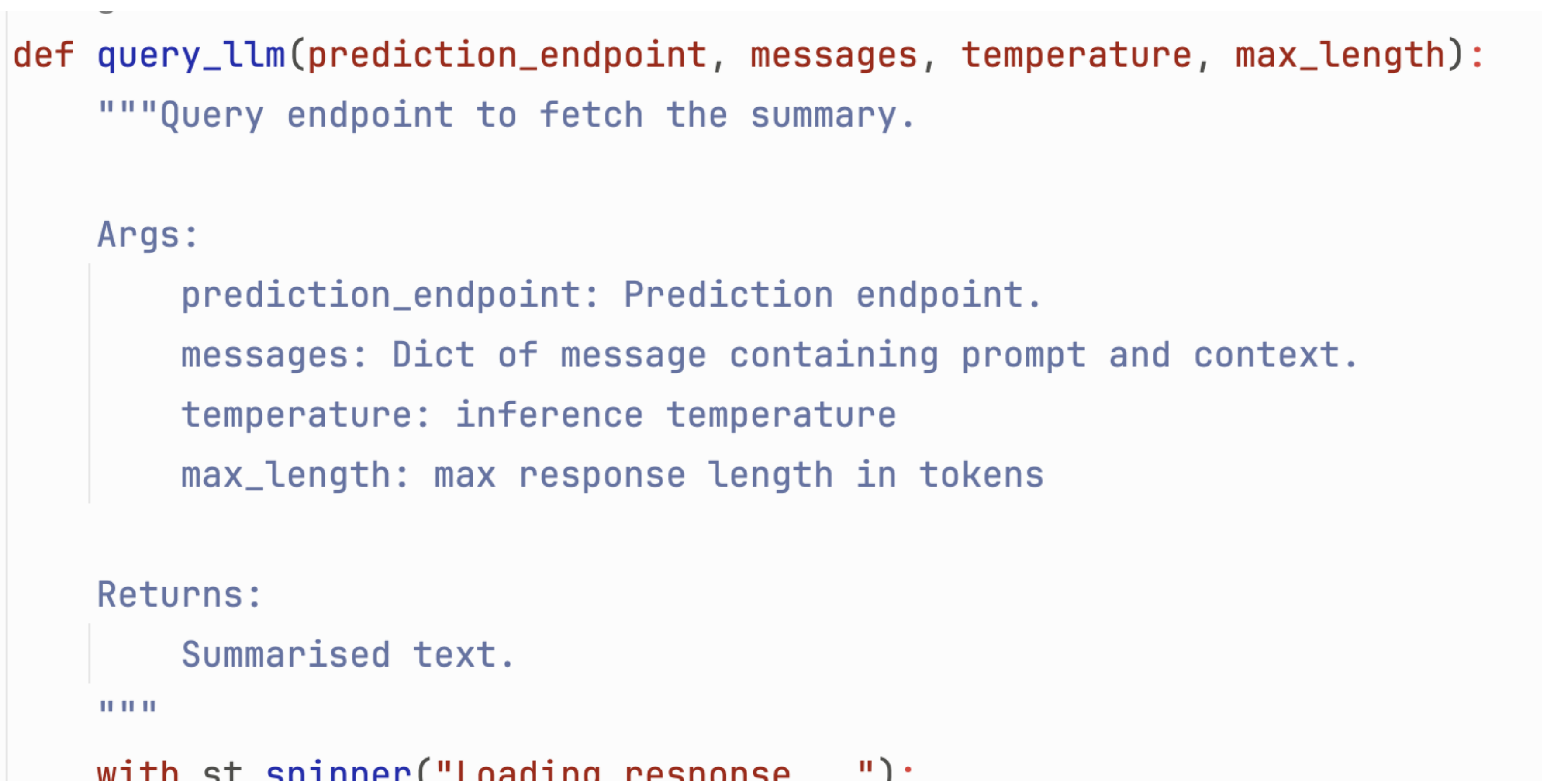


The typo in this plot was missed by me, 3 co-authors and 4 reviewers

Adding pre-commit hooks to the repository flagged it immediately

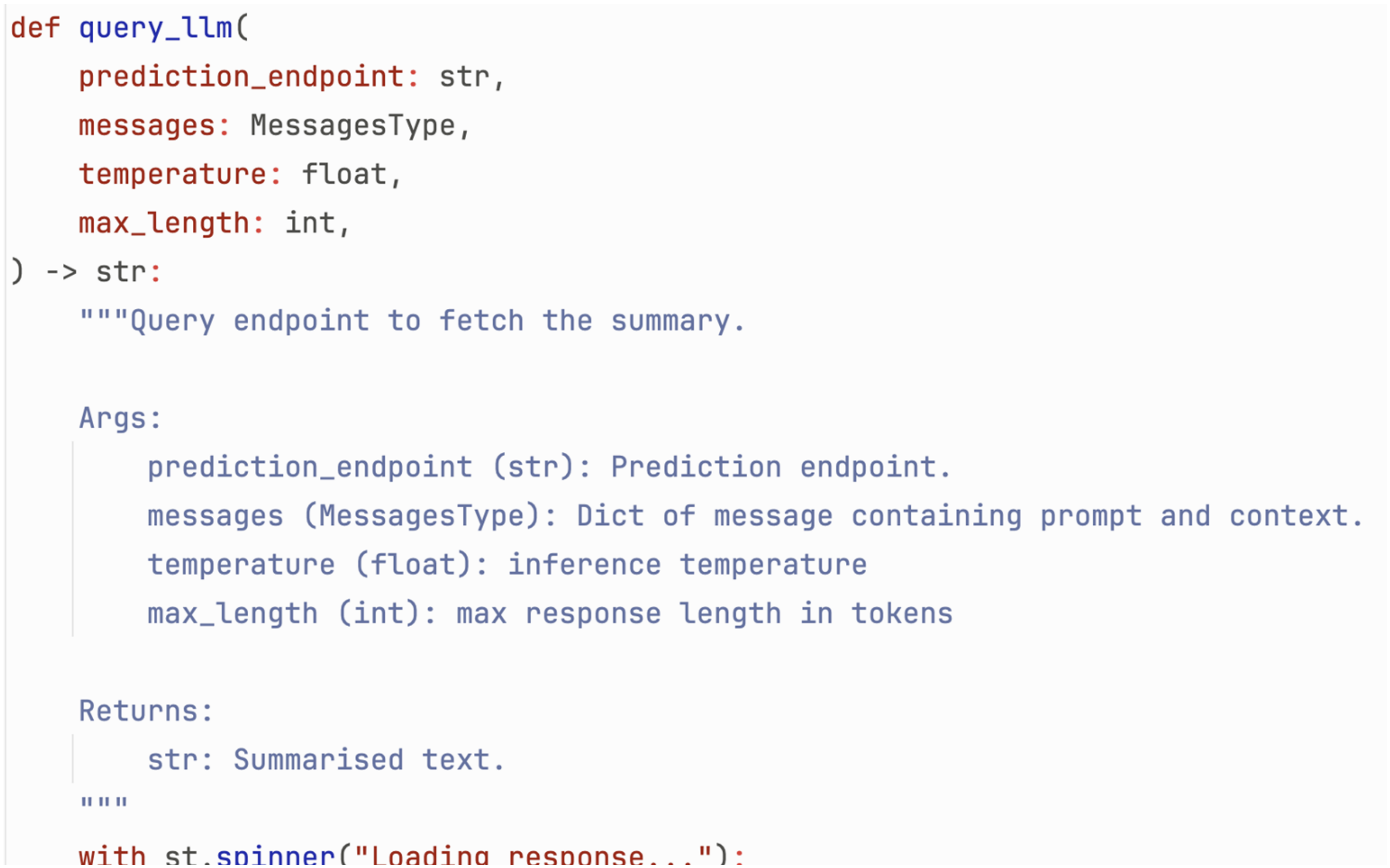
fuzzy labs

⌨️ Typing



```python
def query_llm(prediction_endpoint, messages, temperature, max_length):
    """Query endpoint to fetch the summary.

    Args:
        prediction_endpoint: Prediction endpoint.
        messages: Dict of message containing prompt and context.
        temperature: inference temperature
        max_length: max response length in tokens

    Returns:
        Summarised text.
    """

    with st.spinner("Loading response..."):
```

⌨️ Typing



```python
def query_llm(
    prediction_endpoint: str,
    messages: MessagesType,
    temperature: float,
    max_length: int,
) -> str:
    """Query endpoint to fetch the summary.

    Args:
        prediction_endpoint (str): Prediction endpoint.
        messages (MessagesType): Dict of message containing prompt and context.
        temperature (float): inference temperature
        max_length (int): max response length in tokens

    Returns:
        str: Summarised text.
    """
    with st.spinner("Loading response..."):
```

⌨️ Advantages of Typing

- Makes explicit how different parts of your code are expected to interact
- Allows you to read and understand code faster
- Lets your IDE give you better autocomplete options
- Spots lots of errors before runtime

But type-checking in Python is not perfect… it can be really annoying

🧪 Testing

Two kinds of tests:

[illegible]: Test behaviours of individual functions and classes

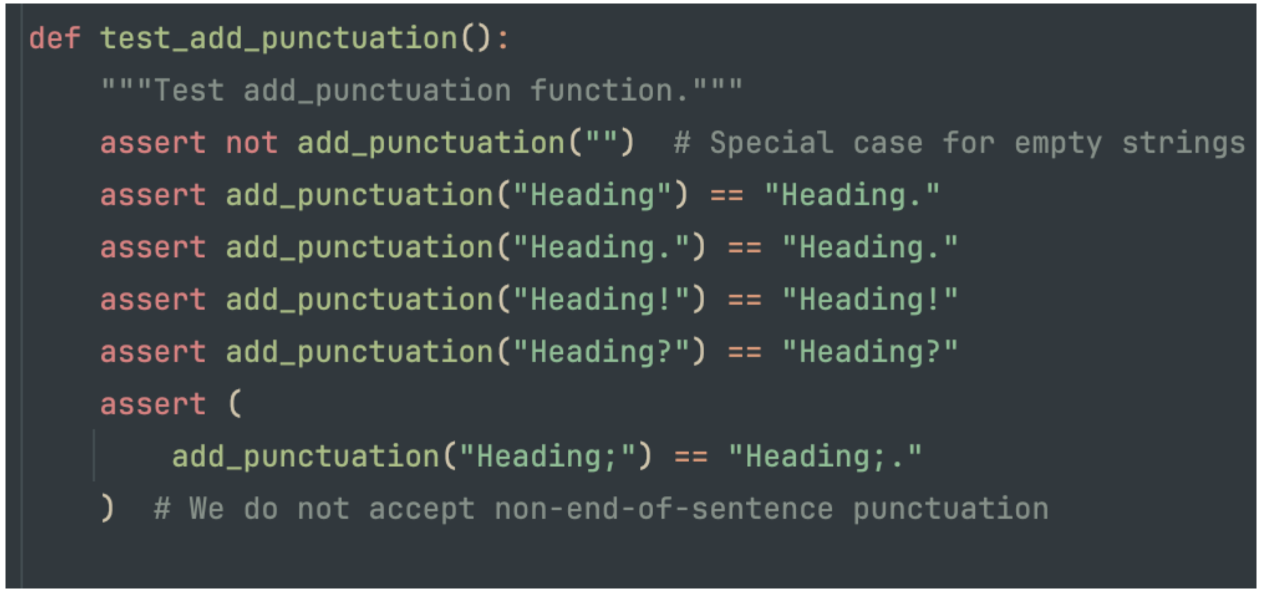
[illegible]: Test the behaviour of the system as a whole

🧪 Testing

Unit tests can check a lot of things about your code:

- Do given inputs give you the correct output
- Is your program in the correct state after a function is called?
- Are the correct intermediate functions called, with the correct arguments?
- Are functions called the correct number of times?
- Do functions attempt to access the correct external resources (filesystems, URLs, databases, etc)

Testing

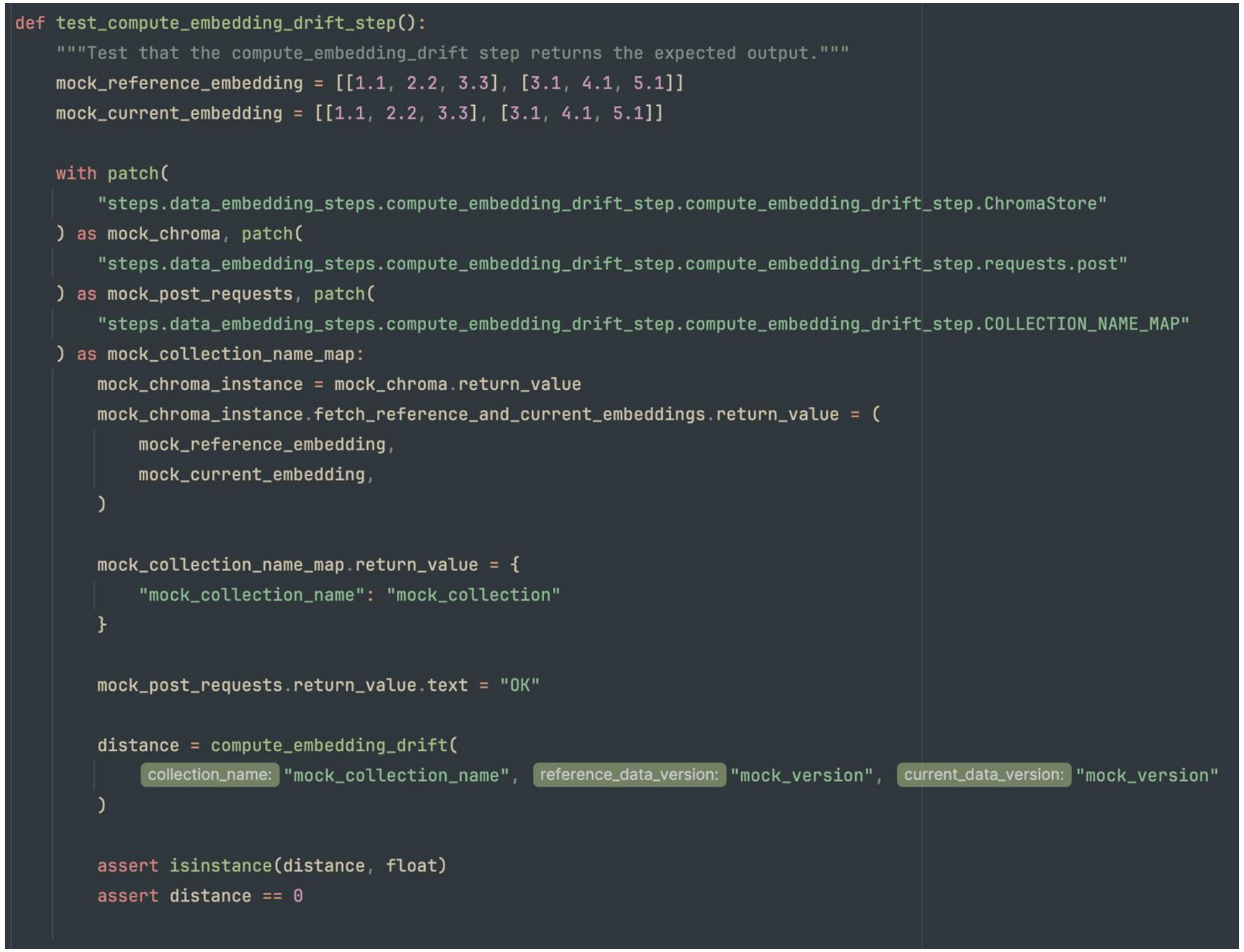


```python
def test_add_punctuation():
    """Test add_punctuation function."""
    assert not add_punctuation("")  # Special case for empty strings
    assert add_punctuation("Heading") == "Heading."
    assert add_punctuation("Heading.") == "Heading."
    assert add_punctuation("Heading!") == "Heading!"
    assert add_punctuation("Heading?") == "Heading?"
    assert (
        add_punctuation("Heading;") == "Heading;."
    )  # We do not accept non-end-of-sentence punctuation
```

Tests can be as simple as just testing that the function output is what's expected for a list of inputs

fuzzy labs

Testing

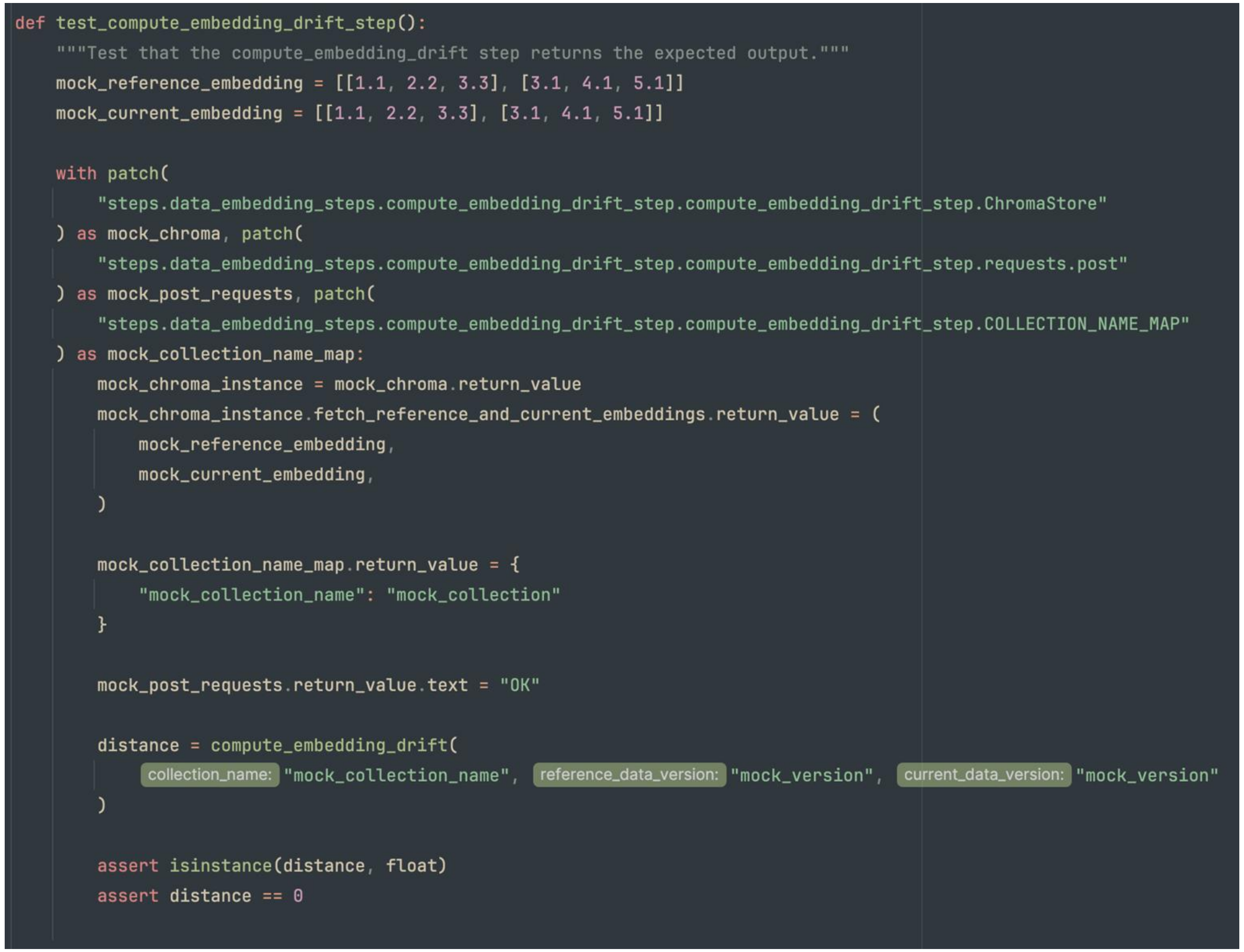


```
def test_compute_embedding_drift_step():
    """Test that the compute_embedding_drift step returns the expected output."""
    mock_reference_embedding = [[1.1, 2.2, 3.3], [3.1, 4.1, 5.1]]
    mock_current_embedding = [[1.1, 2.2, 3.3], [3.1, 4.1, 5.1]]

    with patch(
        "steps.data_embedding_steps.compute_embedding_drift_step.compute_embedding_drift_step.ChromaStore"
    ) as mock_chroma, patch(
        "steps.data_embedding_steps.compute_embedding_drift_step.compute_embedding_drift_step.requests.post"
    ) as mock_post_requests, patch(
        "steps.data_embedding_steps.compute_embedding_drift_step.compute_embedding_drift_step.COLLECTION_NAME_MAP"
    ) as mock_collection_name_map:
        mock_chroma_instance = mock_chroma.return_value
        mock_chroma_instance.fetch_reference_and_current_embeddings.return_value = (
            mock_reference_embedding,
            mock_current_embedding,
        )

        mock_collection_name_map.return_value = {
            "mock_collection_name": "mock_collection"
        }

        mock_post_requests.return_value.text = "OK"

        distance = compute_embedding_drift(
            collection_name: "mock_collection_name", reference_data_version: "mock_version", current_data_version: "mock_version"
        )

        assert isinstance(distance, float)
        assert distance == 0
```

Or they can become more complex, especially if your function wants to call other systems/libraries

fuzzy labs

Testing

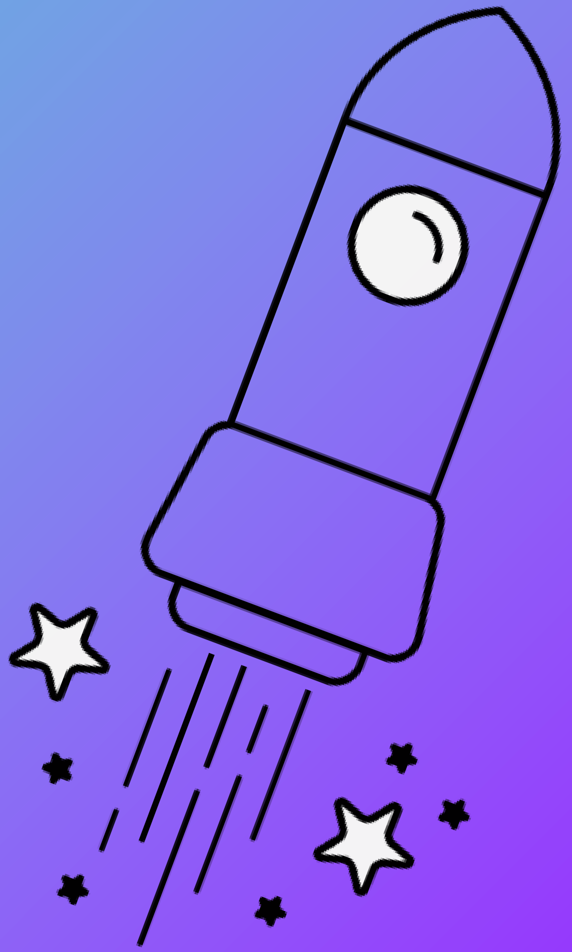


```python
def test_compute_embedding_drift_step():
    """Test that the compute_embedding_drift step returns the expected output."""
    mock_reference_embedding = [[1.1, 2.2, 3.3], [3.1, 4.1, 5.1]]
    mock_current_embedding = [[1.1, 2.2, 3.3], [3.1, 4.1, 5.1]]

    with patch(
        "steps.data_embedding_steps.compute_embedding_drift_step.compute_embedding_drift_step.ChromaStore"
    ) as mock_chroma, patch(
        "steps.data_embedding_steps.compute_embedding_drift_step.compute_embedding_drift_step.requests.post"
    ) as mock_post_requests, patch(
        "steps.data_embedding_steps.compute_embedding_drift_step.compute_embedding_drift_step.COLLECTION_NAME_MAP"
    ) as mock_collection_name_map:
        mock_chroma_instance = mock_chroma.return_value
        mock_chroma_instance.fetch_reference_and_current_embeddings.return_value = (
            mock_reference_embedding,
            mock_current_embedding,
        )

        mock_collection_name_map.return_value = {
            "mock_collection_name": "mock_collection"
        }

        mock_post_requests.return_value.text = "OK"

        distance = compute_embedding_drift(
            collection_name: "mock_collection_name", reference_data_version: "mock_version", current_data_version: "mock_version"
        )

        assert isinstance(distance, float)
        assert distance == 0
```

Or they can become more complex, especially if your function wants to call other systems/libraries

You can *patch* out calls to external libraries, and make *mock* versions of complex objects

Finding Problems Quickly



fuzzy labs

🧙 Use an IDE

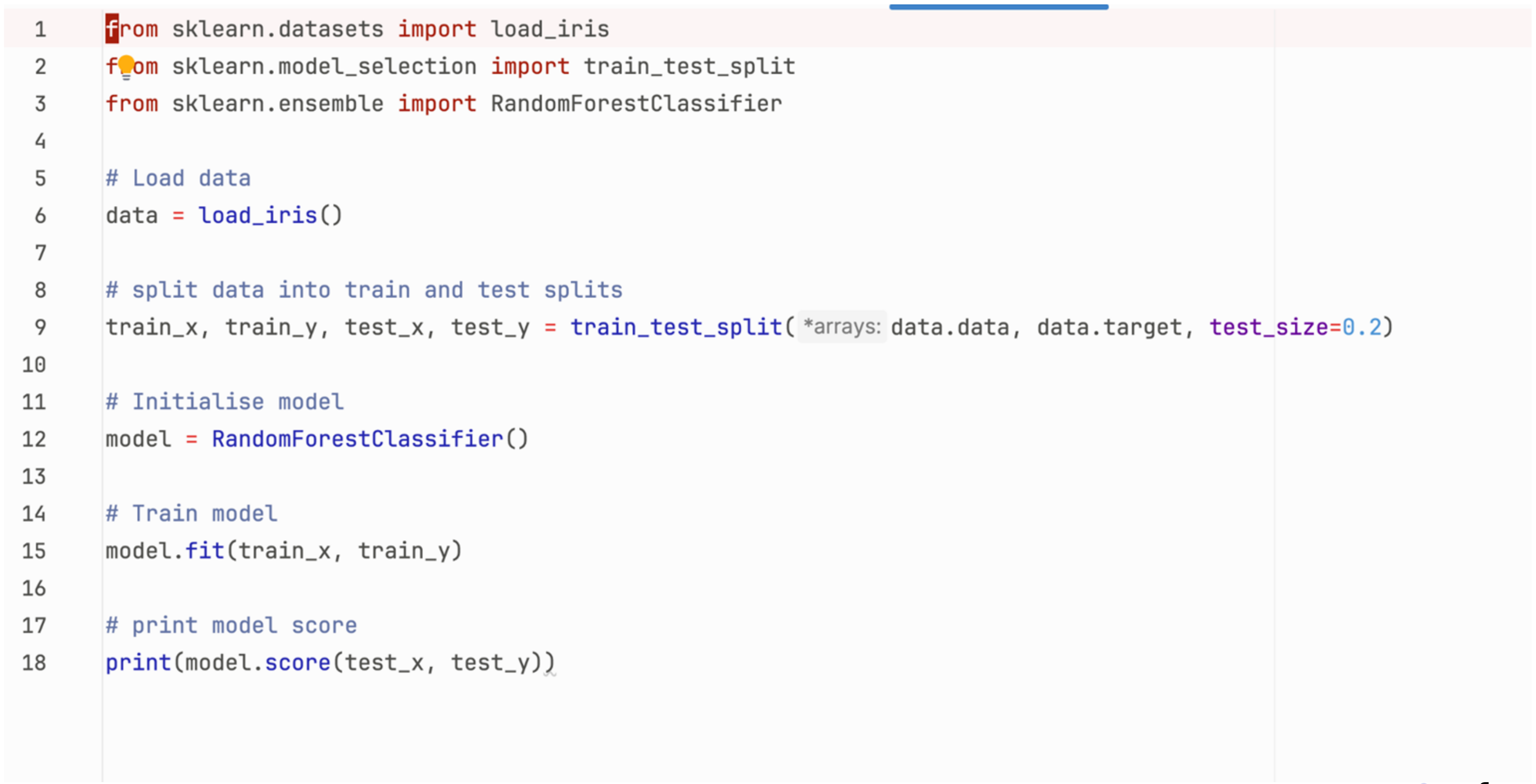
Using an IDE that you’re comfortable with let’s you work a lot more effectively

Knowing the tools in the IDE will allow you to solve problems quicker

Also, being able navigate effectively reduces friction when working on hard problems

Which IDE isn’t too important, but learning one well is

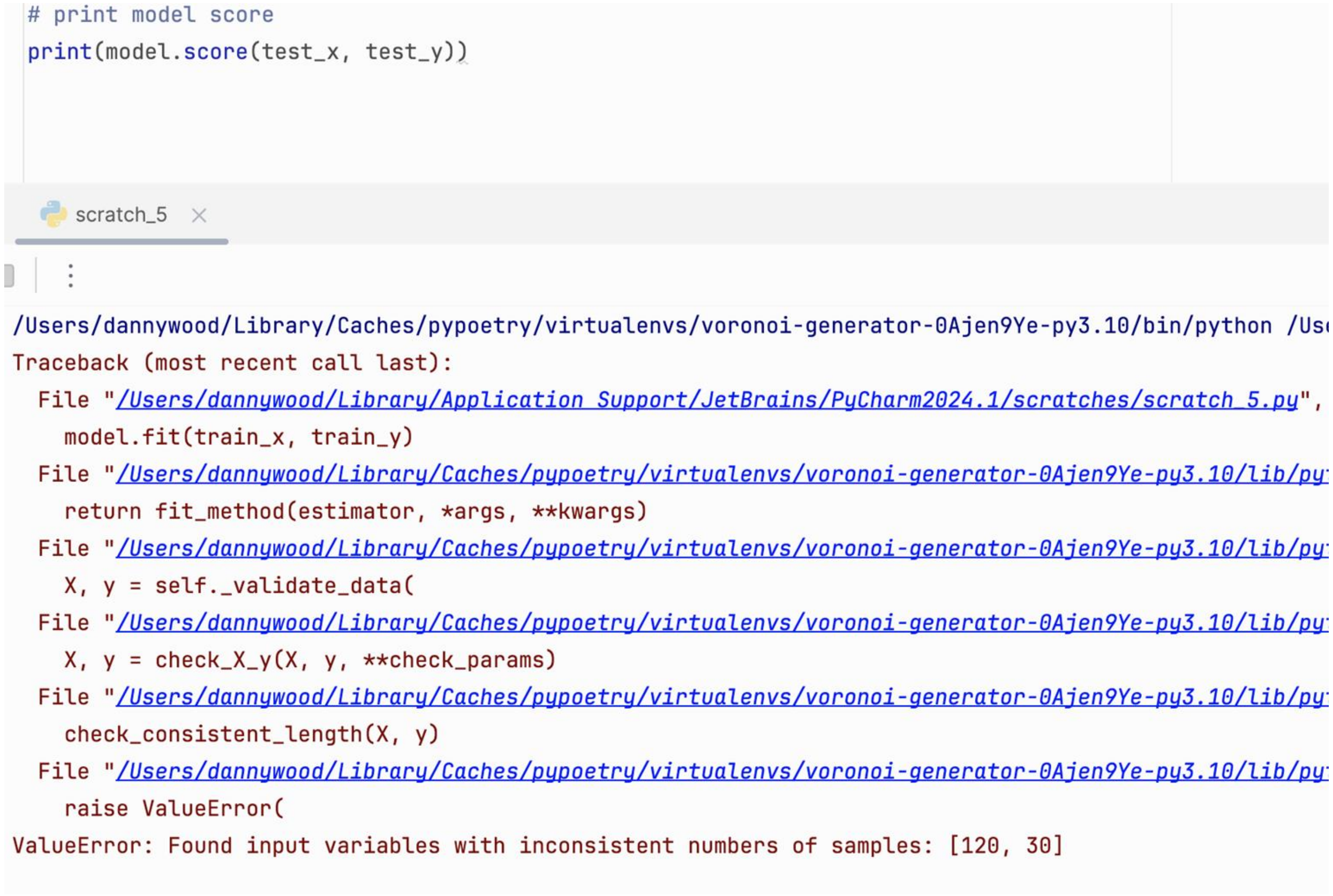
Debuggers



```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# Load data
data = load_iris()

# split data into train and test splits
train_x, train_y, test_x, test_y = train_test_split( *arrays: data.data, data.target, test_size=0.2)

# Initialise model
model = RandomForestClassifier()

# Train model
model.fit(train_x, train_y)

# print model score
print(model.score(test_x, test_y))
```

fuzzy labs

Debuggers



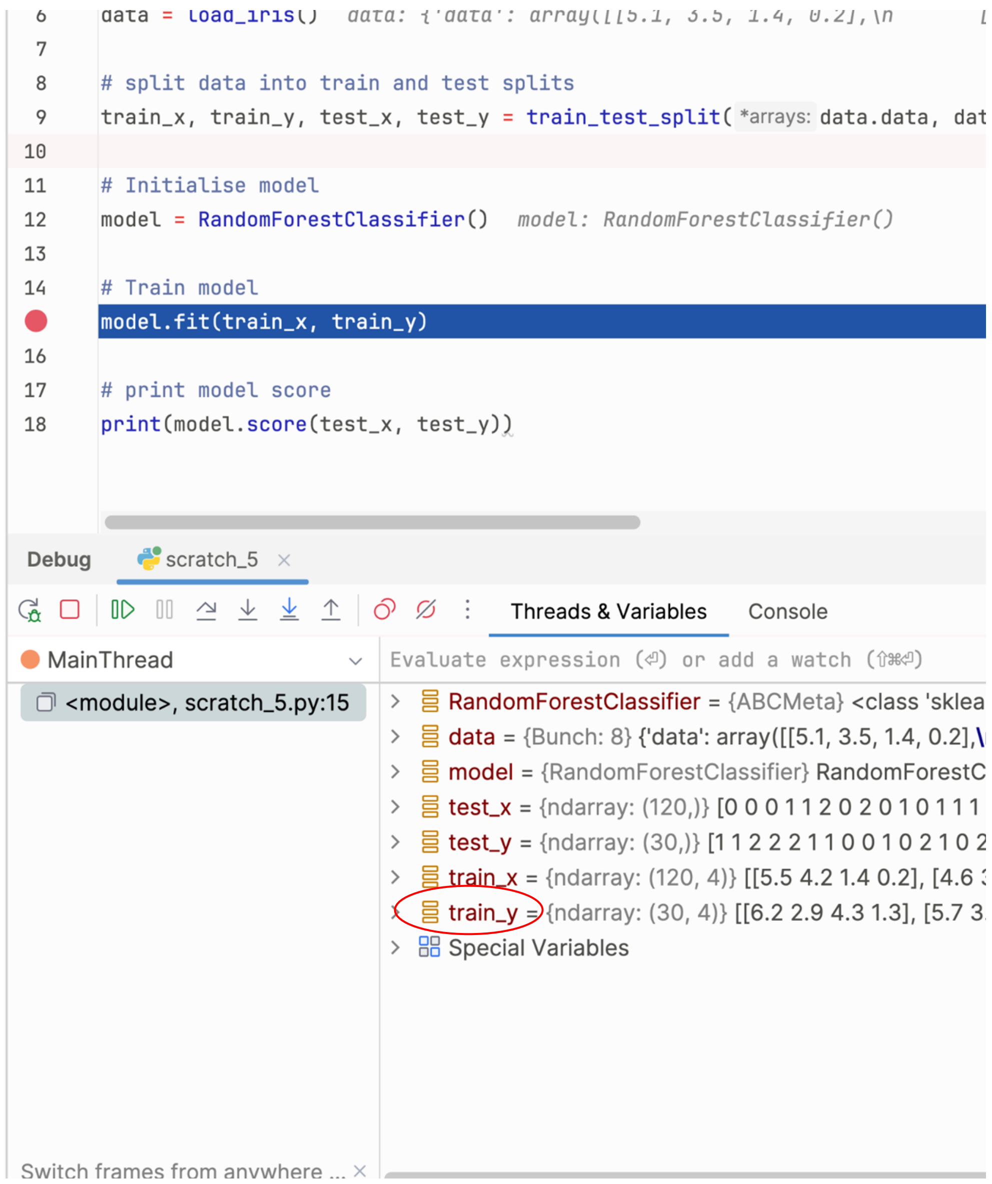
```
# print model score
print(model.score(test_x, test_y))
```

scratch_5

```
/Users/dannywood/Library/Caches/pypoetry/virtualenvs/voronoi-generator-0Ajen9Ye-py3.10/bin/python /Us
Traceback (most recent call last):
  File "/Users/dannywood/Library/Application Support/JetBrains/PyCharm2024.1/scratches/scratch_5.py",
    model.fit(train_x, train_y)
  File "/Users/dannywood/Library/Caches/pypoetry/virtualenvs/voronoi-generator-0Ajen9Ye-py3.10/lib/py
    return fit_method(estimator, *args, **kwargs)
  File "/Users/dannywood/Library/Caches/pypoetry/virtualenvs/voronoi-generator-0Ajen9Ye-py3.10/lib/py
    X, y = self._validate_data(
  File "/Users/dannywood/Library/Caches/pypoetry/virtualenvs/voronoi-generator-0Ajen9Ye-py3.10/lib/py
    X, y = check_X_y(X, y, **check_params)
  File "/Users/dannywood/Library/Caches/pypoetry/virtualenvs/voronoi-generator-0Ajen9Ye-py3.10/lib/py
    check_consistent_length(X, y)
  File "/Users/dannywood/Library/Caches/pypoetry/virtualenvs/voronoi-generator-0Ajen9Ye-py3.10/lib/py
    raise ValueError(
ValueError: Found input variables with inconsistent numbers of samples: [120, 30]
```

fuzzy labs

Debuggers



```
data = load_iris()   data: {'data': array([[5.1, 3.5, 1.4, 0.2],\n

# split data into train and test splits
train_x, train_y, test_x, test_y = train_test_split( *arrays: data.data, dat

# Initialise model
model = RandomForestClassifier()   model: RandomForestClassifier()

# Train model
model.fit(train_x, train_y)

# print model score
print(model.score(test_x, test_y))
```

Debug   scratch_5 ×

Threads & Variables   Console

MainThread

<module>, scratch_5.py:15

Evaluate expression (⏎) or add a watch (⇧⌘⏎)

```
> RandomForestClassifier = {ABCMeta} <class 'sklea
> data = {Bunch: 8} {'data': array([[5.1, 3.5, 1.4, 0.2],\
> model = {RandomForestClassifier} RandomForestC
> test_x = {ndarray: (120,)} [0 0 0 1 1 2 0 2 0 1 0 1 1 1
> test_y = {ndarray: (30,)} [1 1 2 2 2 1 1 0 0 1 0 2 1 0 2
> train_x = {ndarray: (120, 4)} [[5.5 4.2 1.4 0.2], [4.6 
> train_y = {ndarray: (30, 4)} [[6.2 2.9 4.3 1.3], [5.7 3
> Special Variables
```

Switch frames from anywhere ... ×

Setting breakpoints lets you stop the code just before the place it crashes

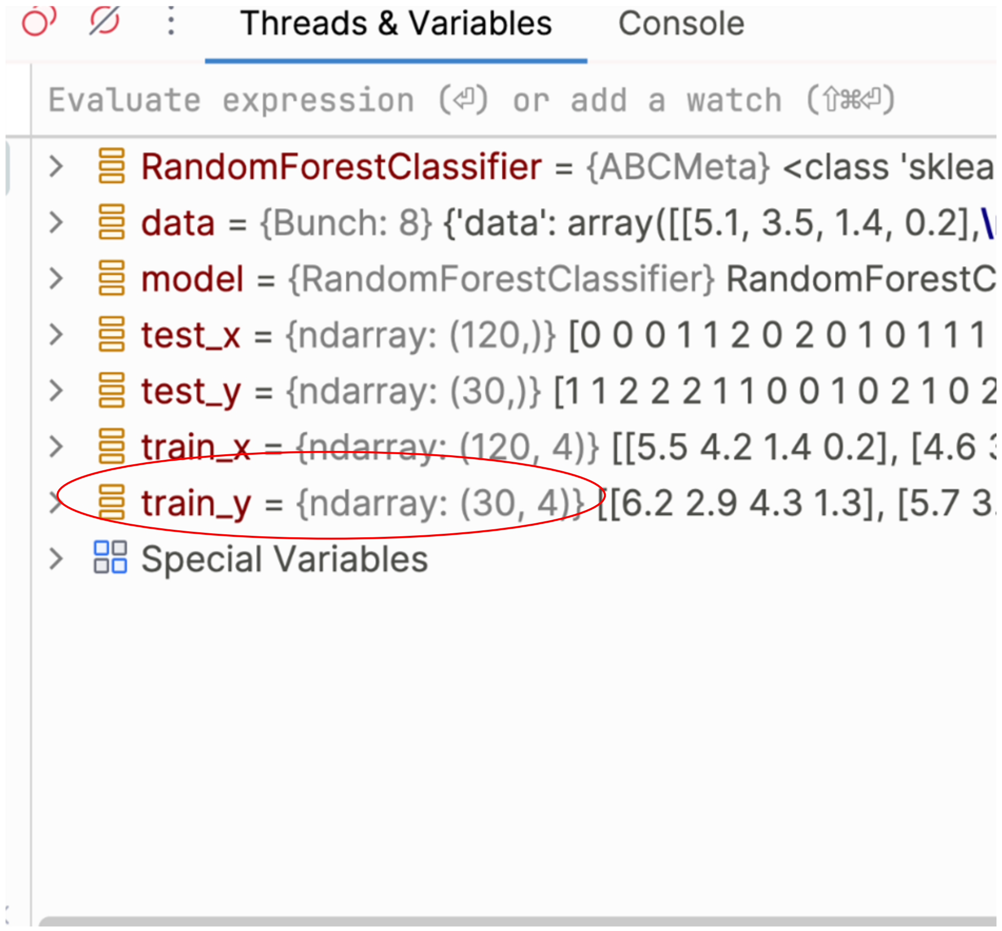
You can then inspect all variables in memory

You can see the stack trace

You can even write and execute code before continuing the rest of the program

fuzzy labs

Debuggers



Threads & Variables   Console

```
Evaluate expression (⏎) or add a watch (⇧⌘⏎)
```

```
> RandomForestClassifier = {ABCMeta} <class 'sklea
> data = {Bunch: 8} {'data': array([[5.1, 3.5, 1.4, 0.2],\
> model = {RandomForestClassifier} RandomForestC
> test_x = {ndarray: (120,)} [0 0 0 1 1 2 0 2 0 1 0 1 1 1
> test_y = {ndarray: (30,)} [1 1 2 2 2 1 1 0 0 1 0 2 1 0 2
> train_x = {ndarray: (120, 4)} [[5.5 4.2 1.4 0.2], [4.6
> train_y = {ndarray: (30, 4)} [[6.2 2.9 4.3 1.3], [5.7 3.
> Special Variables
```

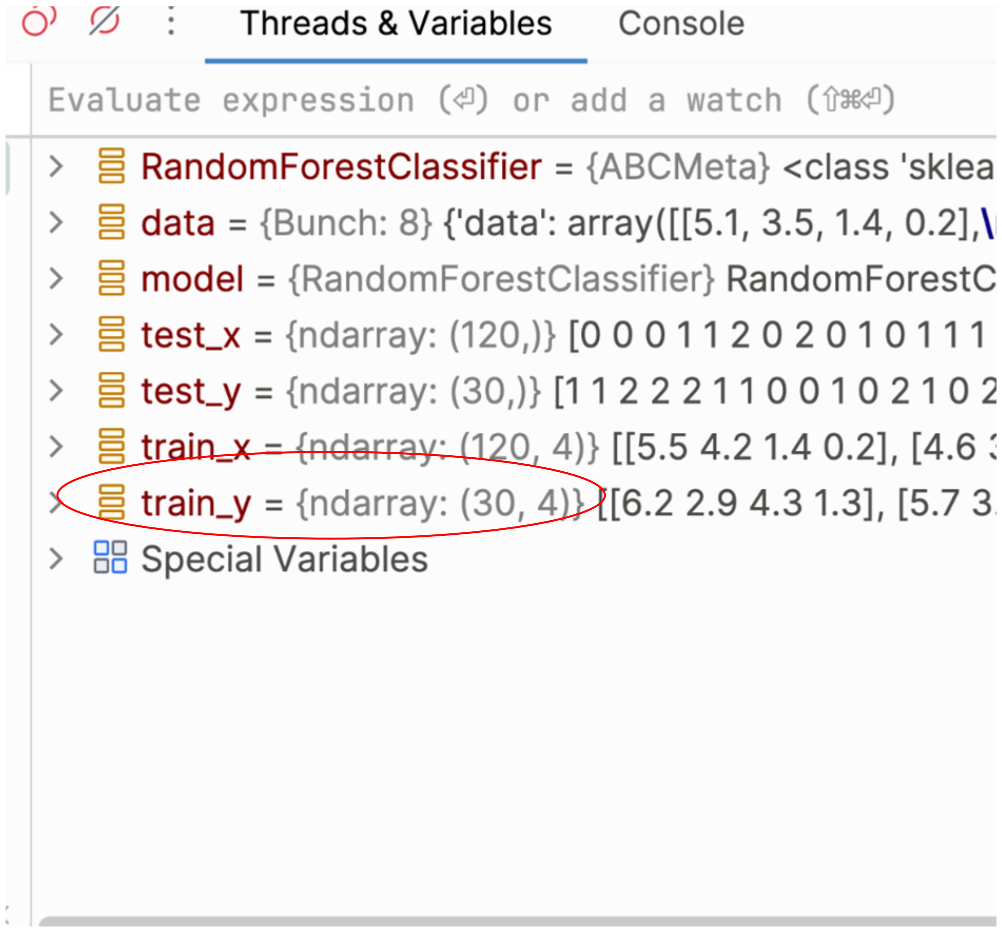
Setting breakpoints lets you stop the code just before the place it crashes

You can then inspect all variables in memory

You can see the stack trace

You can even write and execute code before continuing the rest of the program

Debuggers



Threads & Variables   Console

```
Evaluate expression (⏎) or add a watch (⇧⌘⏎)
> RandomForestClassifier = {ABCMeta} <class 'sklea
> data = {Bunch: 8} {'data': array([[5.1, 3.5, 1.4, 0.2],\
> model = {RandomForestClassifier} RandomForestC
> test_x = {ndarray: (120,)} [0 0 0 1 1 2 0 2 0 1 0 1 1 1
> test_y = {ndarray: (30,)} [1 1 2 2 2 1 1 0 0 1 0 2 1 0 2
> train_x = {ndarray: (120, 4)} [[5.5 4.2 1.4 0.2], [4.6
> train_y = {ndarray: (30, 4)} [[6.2 2.9 4.3 1.3], [5.7 3
> Special Variables
```

Even outside an IDE, you can add breakpoints to your code with a single line of code:

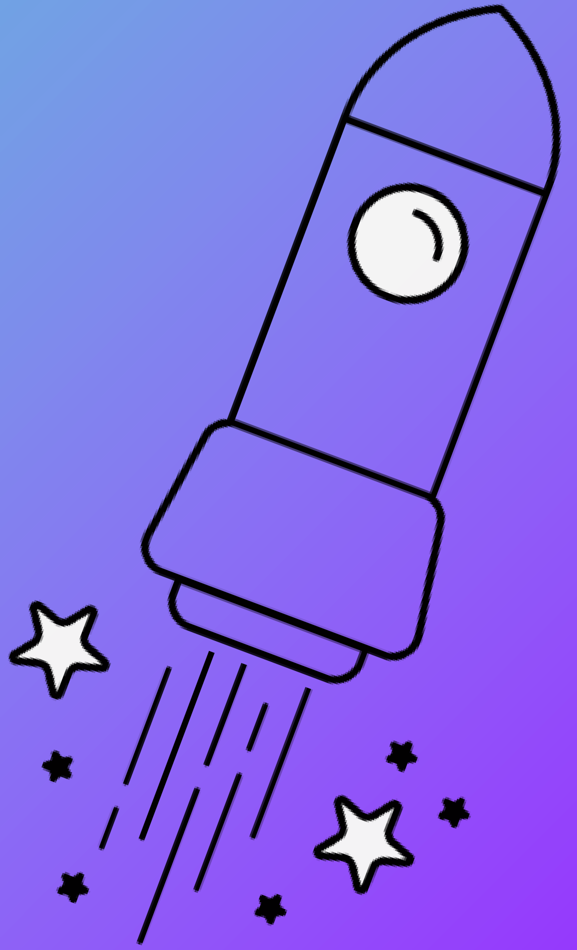
`breakpoint()`

No imports required!

There are also command line debugging tools that can be very useful

fuzzy labs

Automate the Boring Stuff



fuzzy labs

💥 Github Actions

There are lots of things you need to do regularly:

- Install updates
- Run tests
- Scan for vulnerabilities
- Push your code to run in production

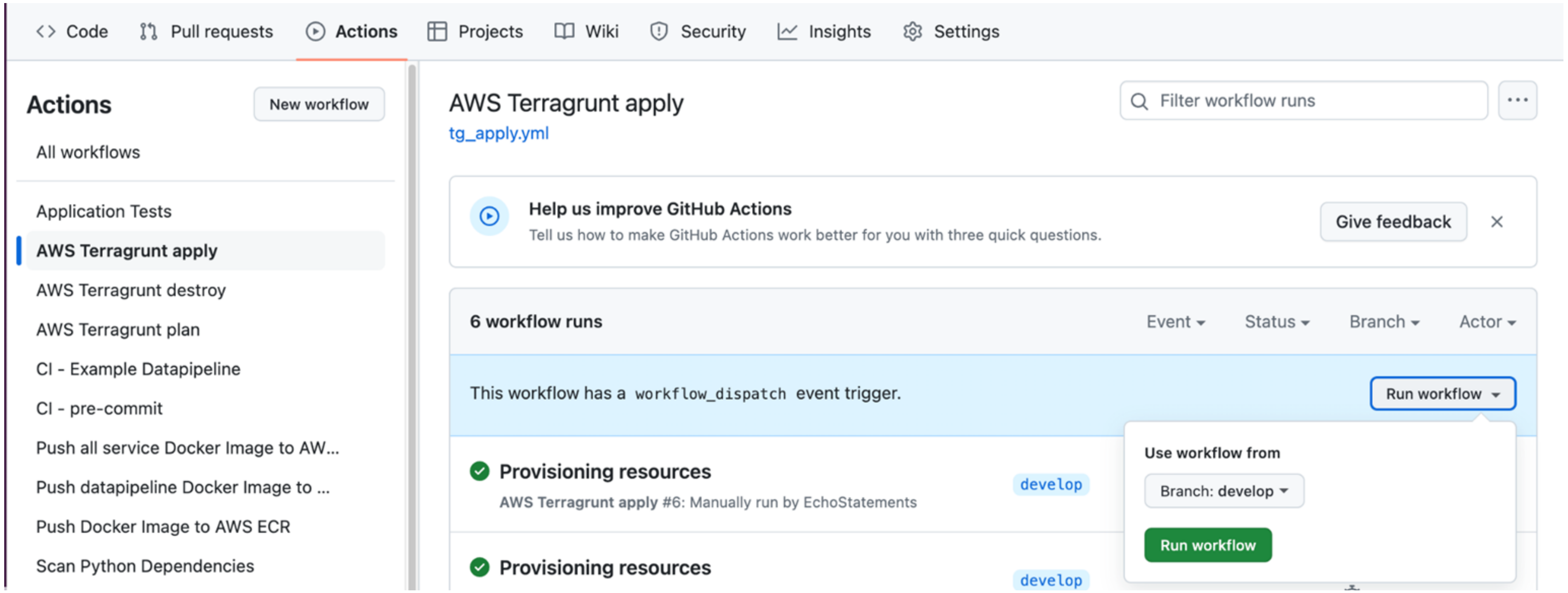
Github lets you automate all of these.

If you can put it in a bash script, Github can run it!

Actions can run on a fixed schedule, whenever there’s an update, or can be triggered manually

💥 Github Actions

Building the right actions can save you lots of time and effort!



Actions New workflow

All workflows

Application Tests
AWS Terragrunt apply
AWS Terragrunt destroy
AWS Terragrunt plan
CI - Example Datapipeline
CI - pre-commit
Push all service Docker Image to AW...
Push datapipeline Docker Image to ...
Push Docker Image to AWS ECR
Scan Python Dependencies

AWS Terragrunt apply
tg_apply.yml

Filter workflow runs

Help us improve GitHub Actions
Tell us how to make GitHub Actions work better for you with three quick questions.
Give feedback

6 workflow runs Event Status Branch Actor

This workflow has a `workflow_dispatch` event trigger. Run workflow

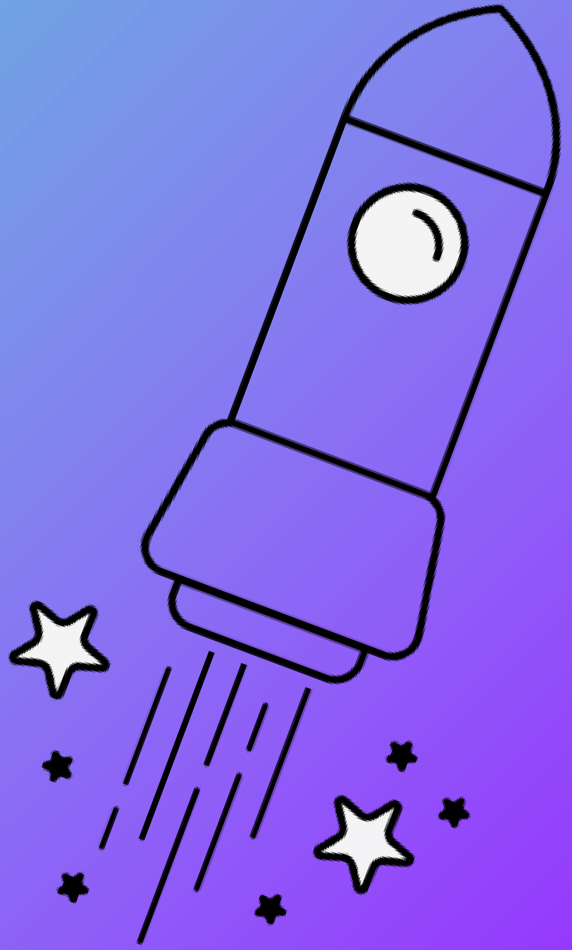
Use workflow from
Branch: develop
Run workflow

Provisioning resources `develop`
AWS Terragrunt apply #6: Manually run by EchoStatements

Provisioning resources `develop`

fuzzy labs

Wrapping Up



fuzzy labs

🤔 What I’d recommend

Start now

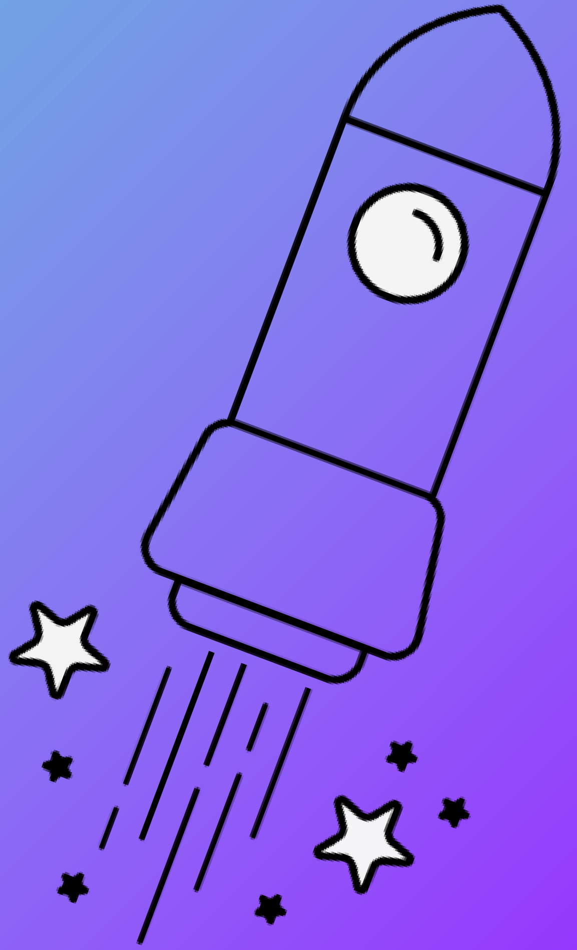
- Use version control
- Pre-commit hooks are great
- Use a unique virtual environment for each project
- Use a debugger

What you’ll need to know in industry

- Using Github collaboratively (PRs, merging branches, squashing commits)
- Unit tests

fuzzy labs

Thank You



fuzzy labs

Resources

[illegible]

https://docs.github.com/en/actions/quickstart

[illegible]

https://python-poetry.org/docs/basic-usage/

https://www.youtube.com/watch?v=Ji2XDxmXSOM

[illegible]

https://medium.com/@anton-k./how-to-set-up-pre-commit-hooks-with-python-2b512290436

Some recommendations for hooks to try: black, ruff, mypy (for typing)

[illegible]

https://www.youtube.com/watch?v=j0Wz_uBaDmo

fuzzy labs

Next month:

Wednesday 31th July

Dr Nicola Dinsdale (University of Oxford)

Domain Adaptation

Fancy more networking?

Head over to the pub!