GOTO

AARHUS 2022

#GOTOaar

Bridging the chasm between Research and Software Development

A story from a fertility app

goto aarhus, 15 June 2022
Linda Stougaard Nielsen, Director Data Science, Ava AG

Linda Stougaard Nielsen, Ph.D.
Director Data Science
Ava AG

Agenda

- Presenting ava – vision and products
- The problem – two examples
- Possible solutions
- Remaining challenges
- Q&A

Ava AG

Swiss company

Founded 2014

Awards in startup, medtech, femtech

Wearable device collecting signals:

- Heart rate, heart rate variability, temperature, breathing rate, perfusion, sleep state every 10 s
- 200’000 users each collecting 1 mio data points every night

Applications:

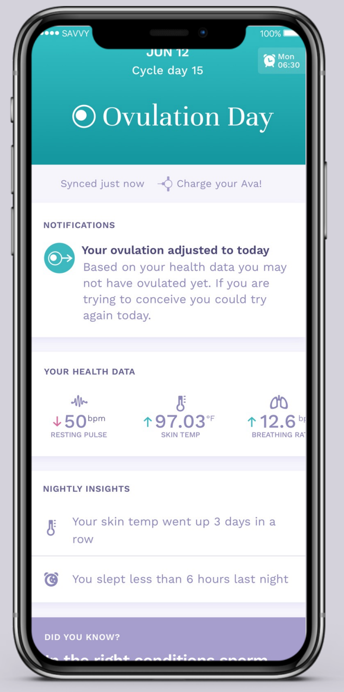
- Fertility app launched in 2016 (CE & FDA approved)
- Contraception app launching 2023 in Europe
- Side track: Covid early detection (research project)

Ava’s vision is to assist women throughout their life with their reproductive health

Track Cycle

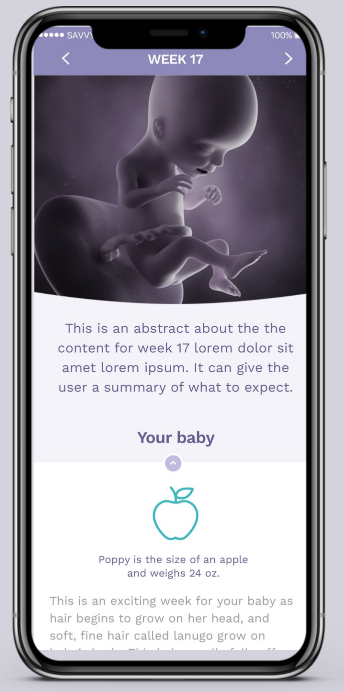
Helps young adults make sense of their bodies and cycle

Conception



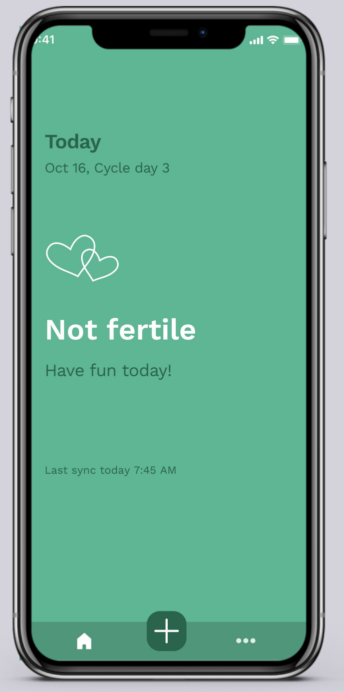
Doubles chances of pregnancy

Pregnancy



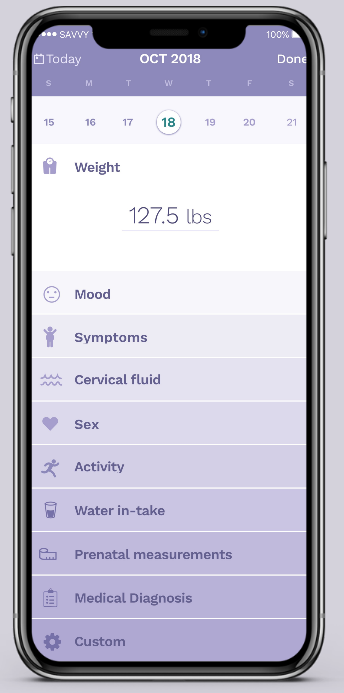
Monitors for healthy pregnancy

Contraception



Fully certified digital birth control (in progress)

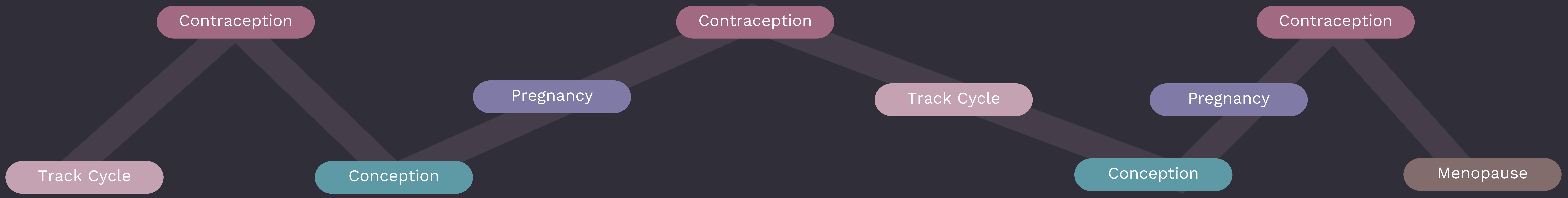
Menopause



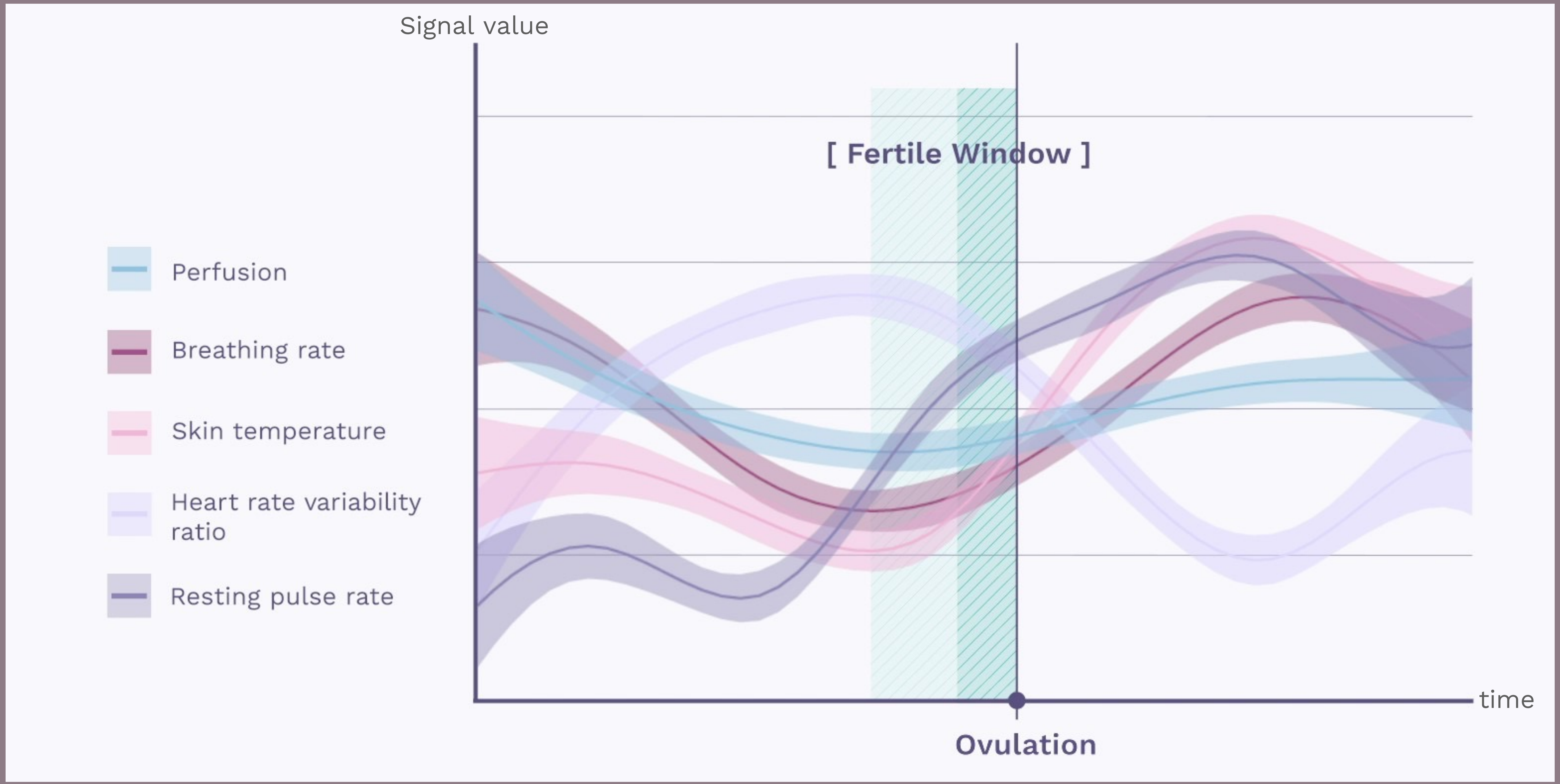
Monitors for healthy peri-menopausal life (not started)

ava fertility app

ava prevent app



How does it work



Wearable Sensors Reveal Menses-driven Changes in Physiology and Enable Prediction of the Fertile Window: an Observational Study, Goodale, B. M., Shilaih, M., Falco, L., Dammeier, F., Hamvas, G., & Leeners, B. (2019)

Medical research

- Hormonal changes throughout cycle, pregnancy, infections, menopause
- Influence on physiological signals

Clinical trials

- Cooperation with research institutes
- Certification and postmarket surveillance
- Statistical analyses

Machine learning

- Signal processing
- Classical statistics and machine-learning
- Neural Networks
- Constraints on noisy data and risk

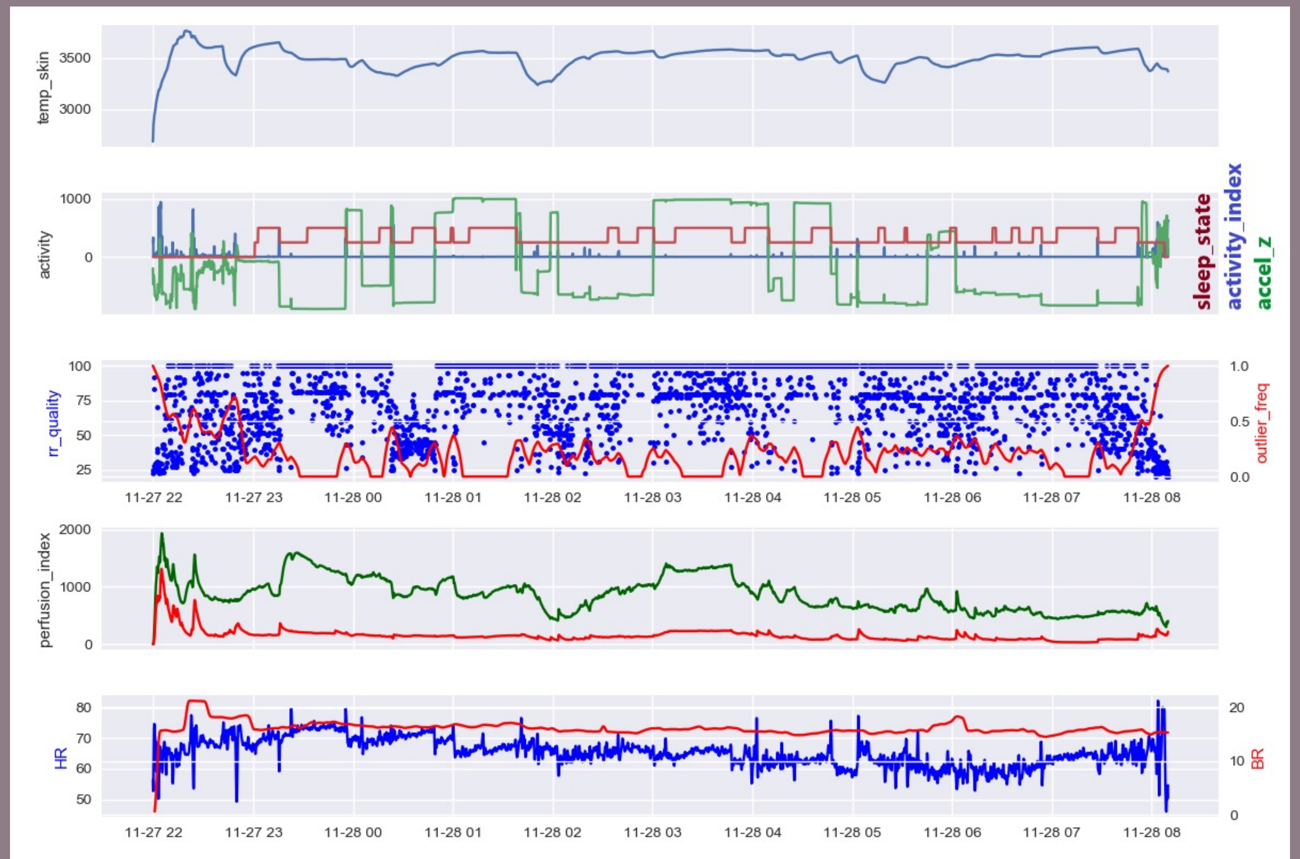
"One of the hardest parts of machine learning is effectively putting models in production."

https://neptune.ai

The problem

2 examples

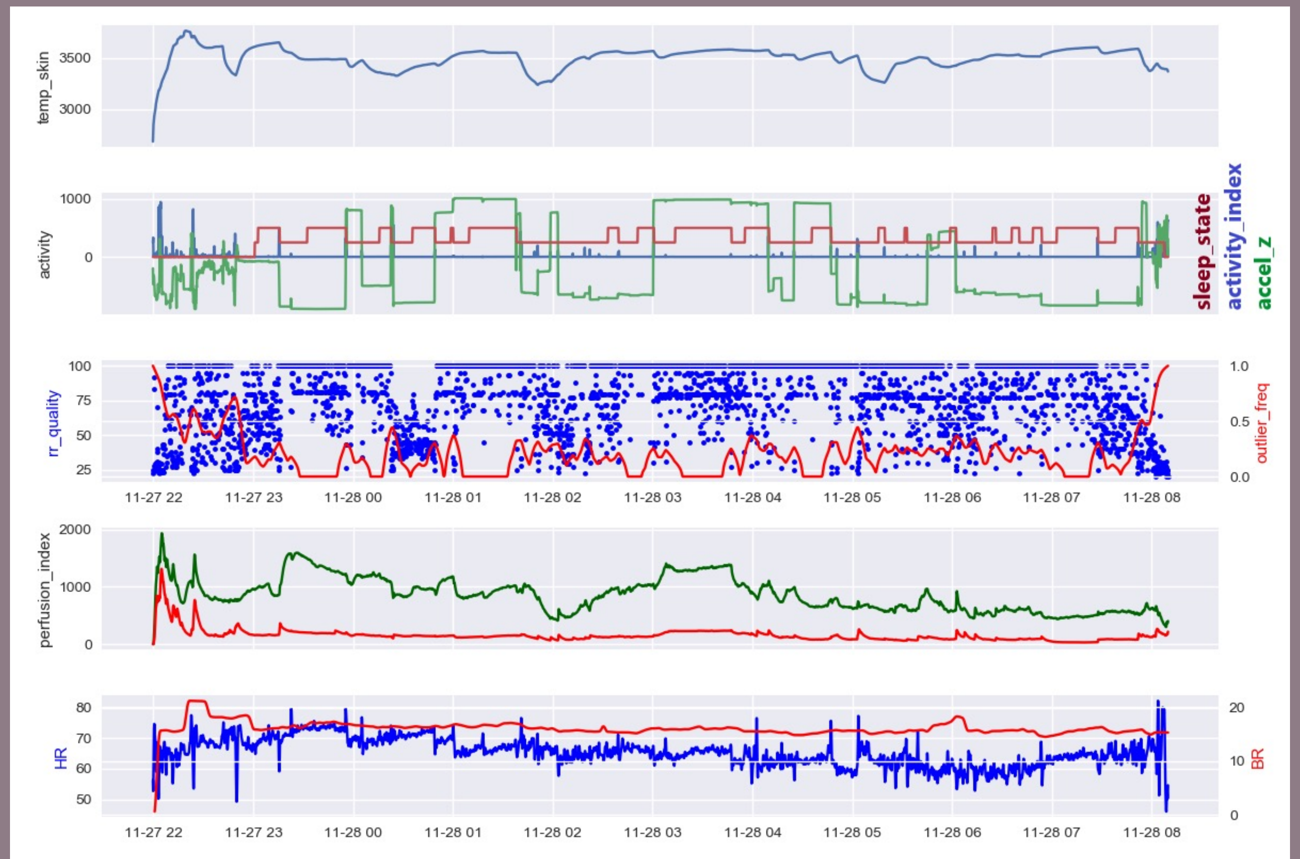
Example 1: re-process all raw data files



Parse, split, filter
Extract nightly features
Assess quality

	EXISTING
Production code	Python module for data processing
Scaling solution	Spark on AWS
Challenges	- Spark own language - Re-implement solution - Maintain 2 parallel implementations

Example 1: re-process all raw data files



Parse, split, filter
Extract nightly features
Assess quality

	OLD	NEW
Production code	Python module for data processing	
Scaling solution	Spark on AWS	Parallel on AWS
Challenges	- Spark own language - Re-implement solution - Maintain 2 parallel implementations	- Additional implementation of infrastructure to run parallelisation

Example 2: Re-train a tensorflow model

Tensorflow model – not the issue (ok packaged)

Data handling:

- Input data into tensor
- Normalisation
- Filtering of data (outlier detection)
- Model output into app input

Scattered all over the code

- Impossible to follow and to maintain / extend

Different code than used in training

- Impossible to prove consistency between eg. outlier handling in training code vs. production code

Well-known issue a.k.a "glue code"

- supporting code for getting data in and out of generic ML packages
- ~95% of code
- anti-pattern

Re-implementation of code

Issues identified

- Re-implementation of code
- “Glue code”

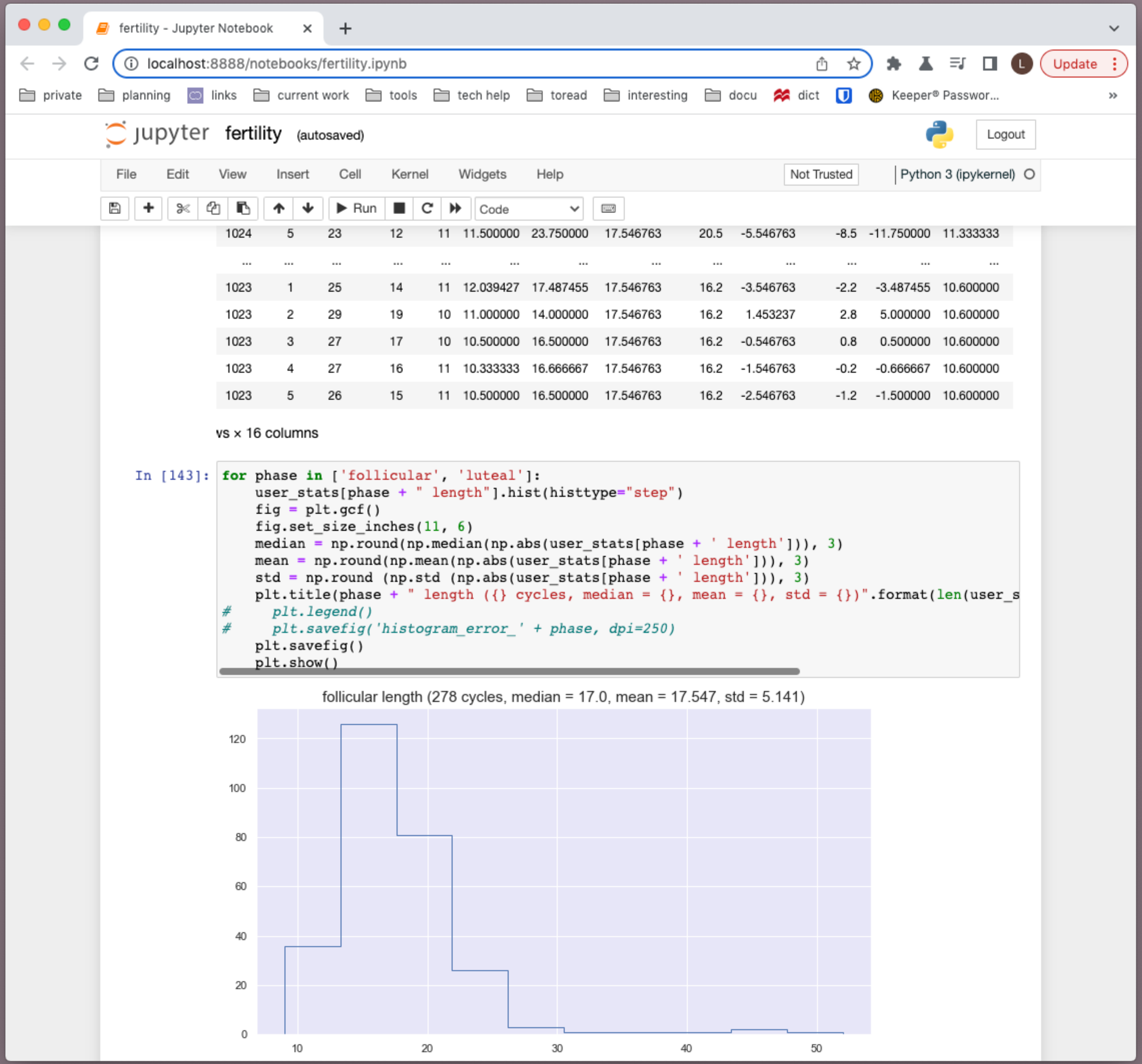
Solutions

Just a question of writing good code, right?

Re-use of code

- But some-one has to write the code that can be re-used
- And it may require additional infrastructure (eg. batch processing)
- And what about experimentation: modifying code slightly in various ways

Experimentation / research



Research code

Scientific tools are different

- Ex Jupyter Notebook
- Line-by-line execution

Iterative approach

- Experimentation requires trial & error
- Lot of alternative code for different approaches
- Often incorporates with external tools (ex tensorboard for training)

Scientists are generalists

- “One solution to fits all”
  -> End up with long scripts that can do EVERYTHING
- Not specific to single problem (many trials)

Transitioning from research code to production code

Research code	Production code	Consequences / differences
Flexibility (can do everything)	Specialised (one purpose / fixed final version)	Full of redundant code
Manual usage	Automated repeated usage	No focus on stability, exception handling, and unit testing
Used by 1 person	Maintained by many people	Not following best practices
Row by row execution	Script execution	Not object oriented / patterns
Stand-alone	Interacting with system	No clear interfaces
Data from large data sets	One data point from stream	Data handling is different
Need parallelising within	Need parallelising around	Need restructuring and optimising

Machine learning code into production (bridging the chasm)

Experimentation

- Some data handling
- Signal analysis
- Signal processing
- Data mining
- Data manipulation
- Algorithm training
- Feature engineering
- Verification
- Evaluation

Re-implement

Production

- Test and debug
- Monitoring
- Ressource management
- Automation
- ML code
- UX
- Configuration
- Security
- Logging
- Data handling

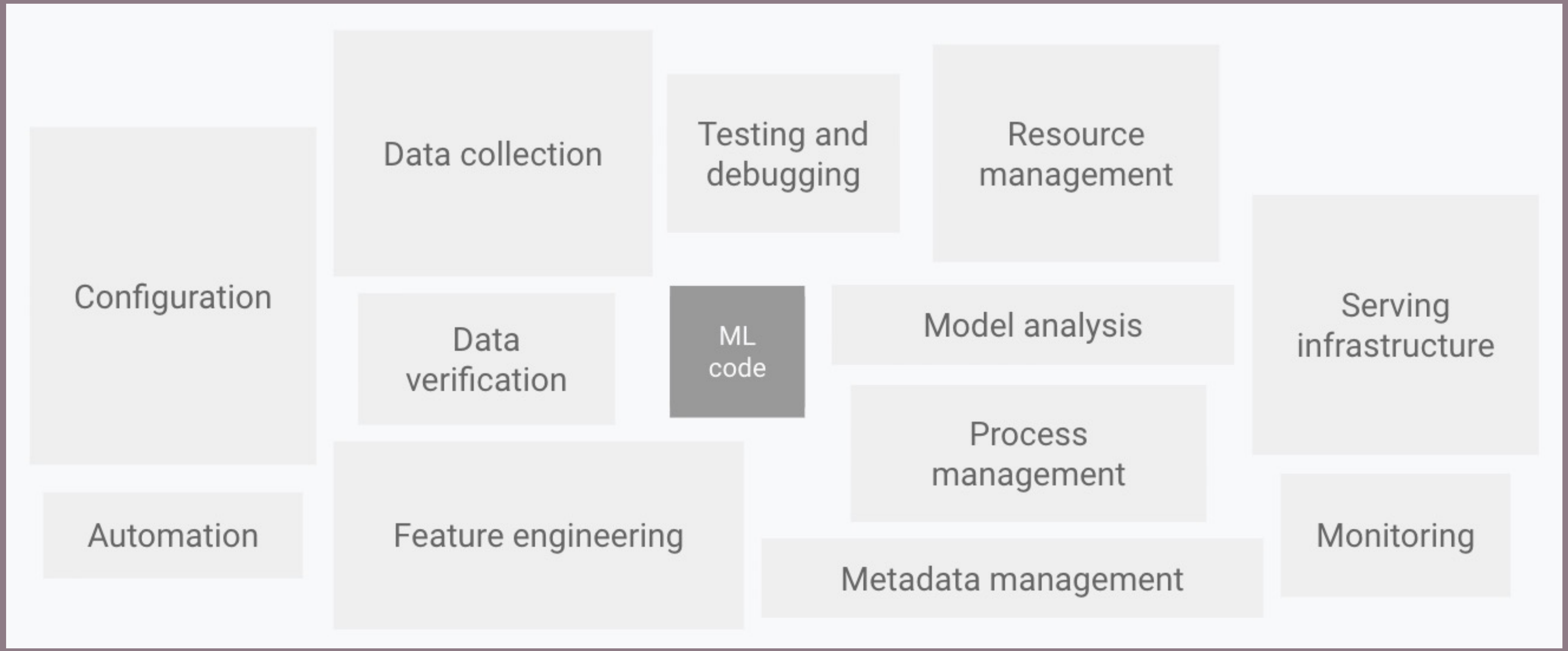
Required infrastructure surrounding lifecycle of ML code

Glue code (ML):

- Getting data in and out of ML packages

Glue code (SW):

- Connecting incompatible components
- No contribution towards requirements



Source: “Hidden Technical Debt in Machine Learning Systems”, NIPS 28 (2015) by Google

Who should implement all this glue code

Educate researchers to be good SW devs

NO!

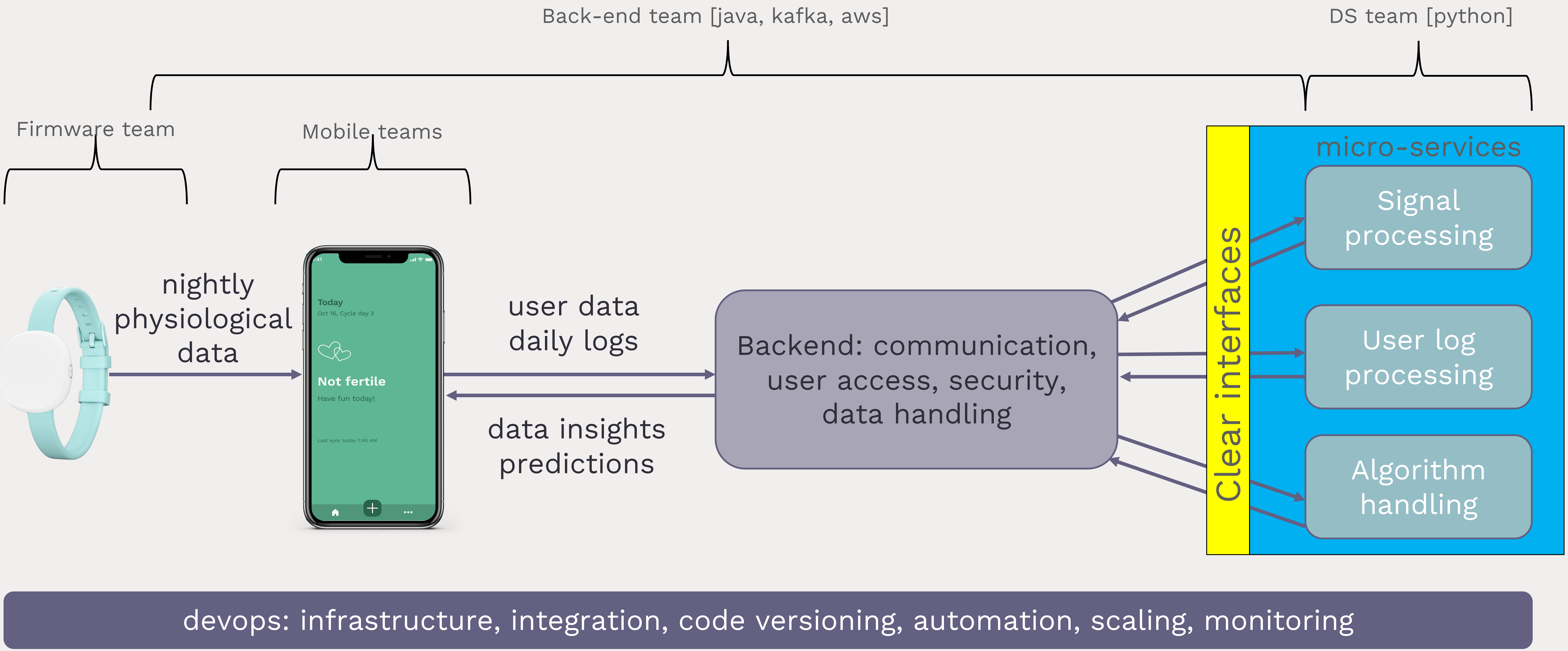
- No interest / no skills in this area
- Focus on research / modelling / data processing

Let SW devs build the code

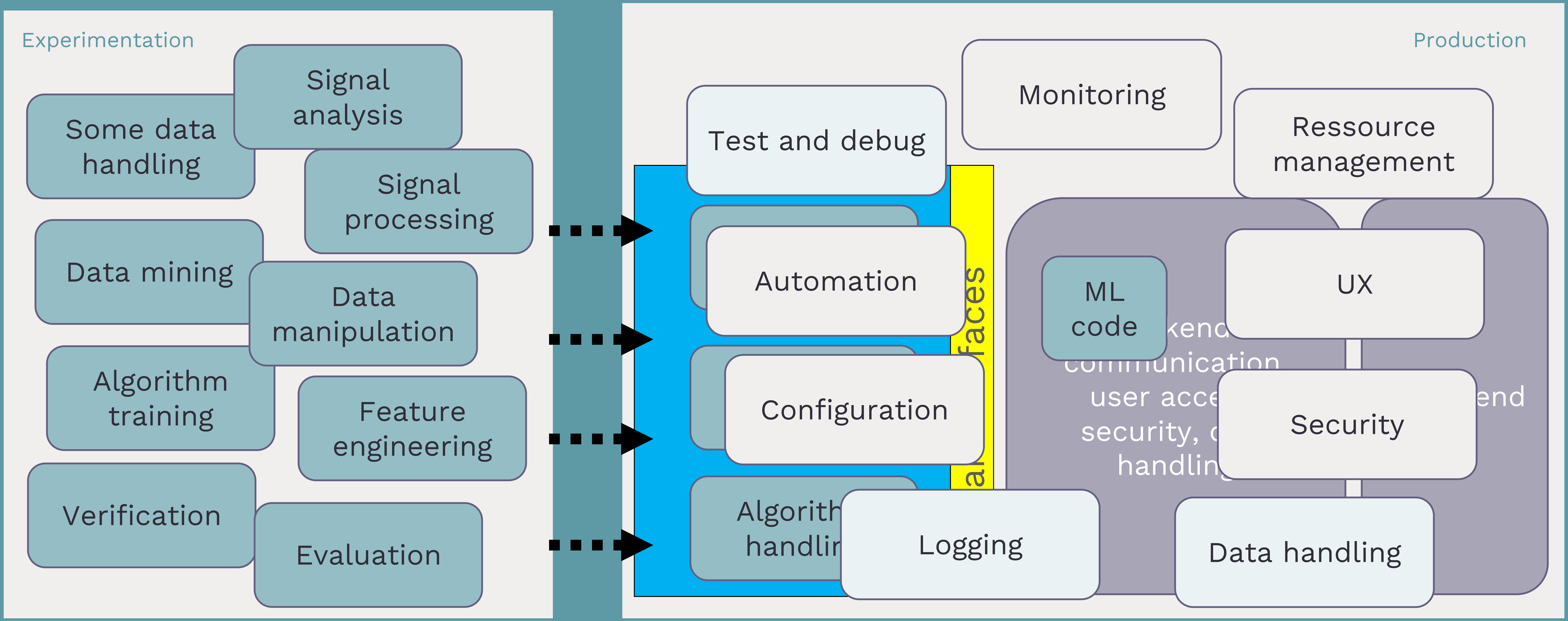
Depends...

- Parallel implementations
- Inconsistency problems
- Time to market

High level architecture



ML modules as microservices



devops: infrastructure, integration, code versioning, automation, scaling, monitoring

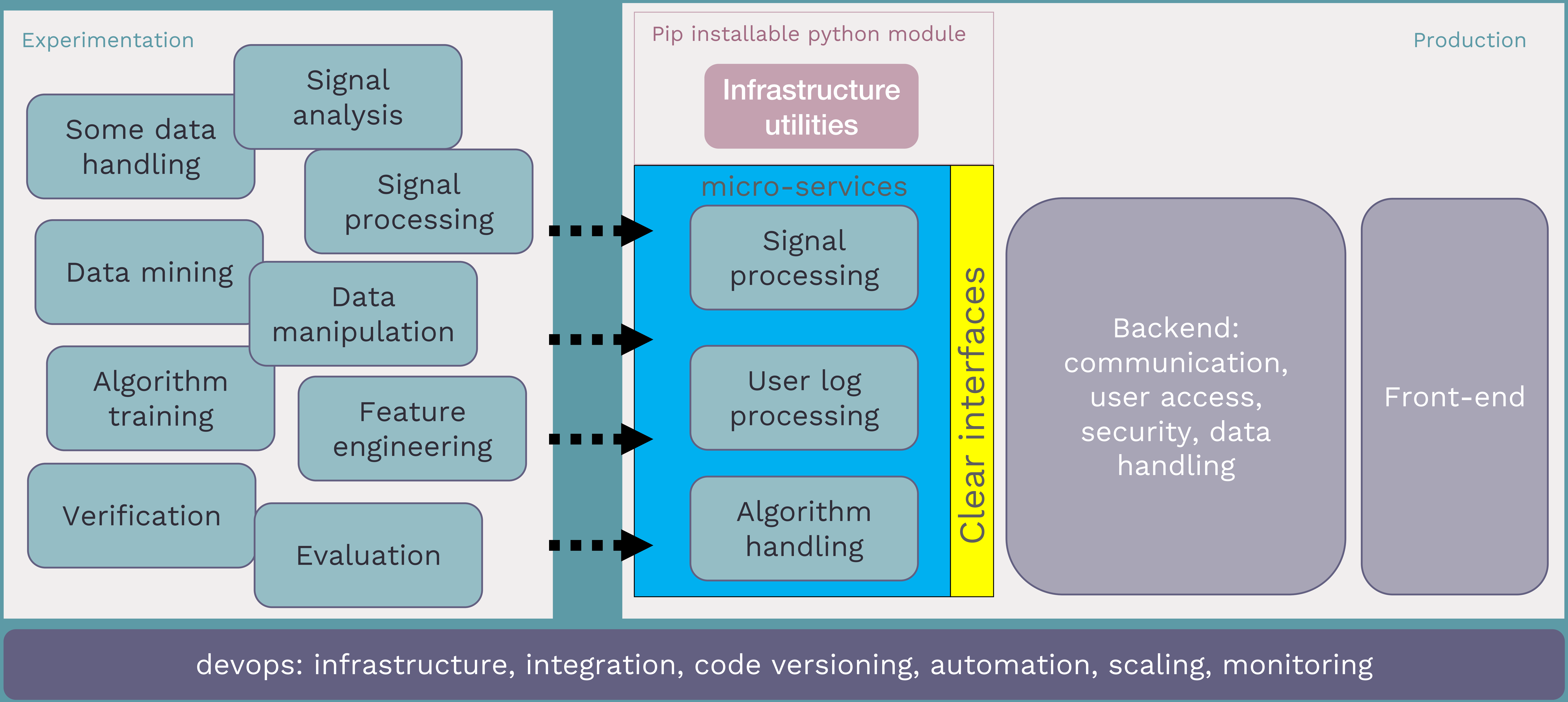
Modularise the DS modules

Algorithm module

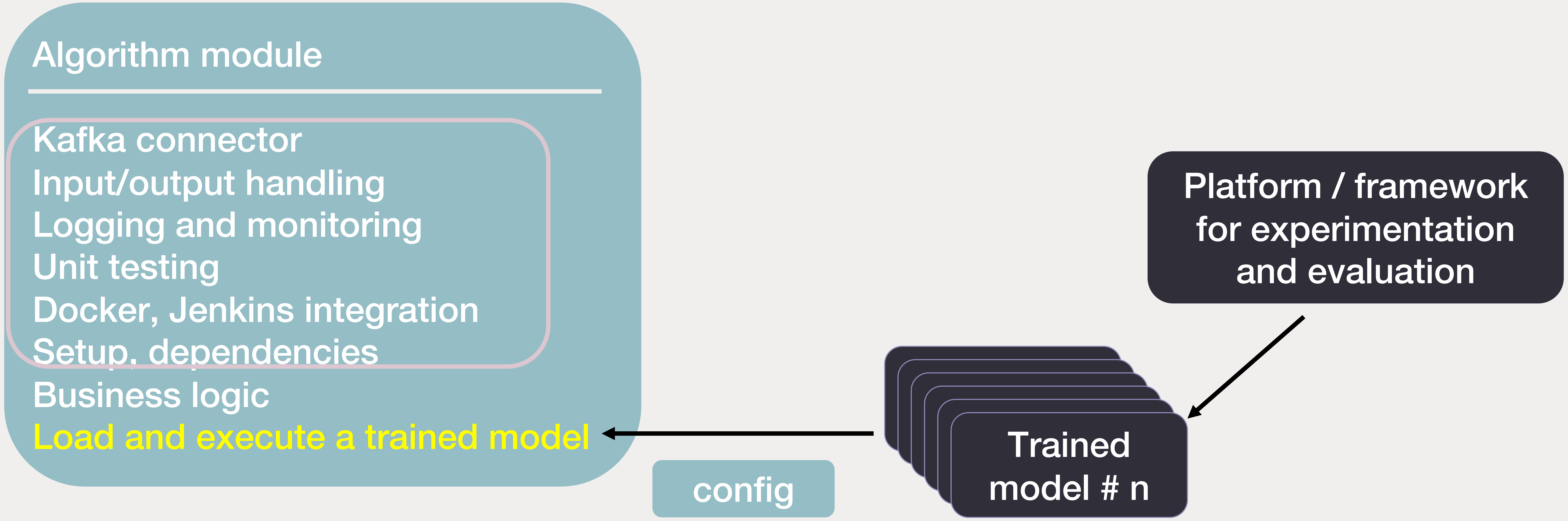
Kafka connector
Input/output handling
Logging and monitoring
Unit testing
Docker, Jenkins integration
Setup, dependencies
Business logic
Load and execute a trained model

Infrastructure utilities

Microservices and modularising the common code

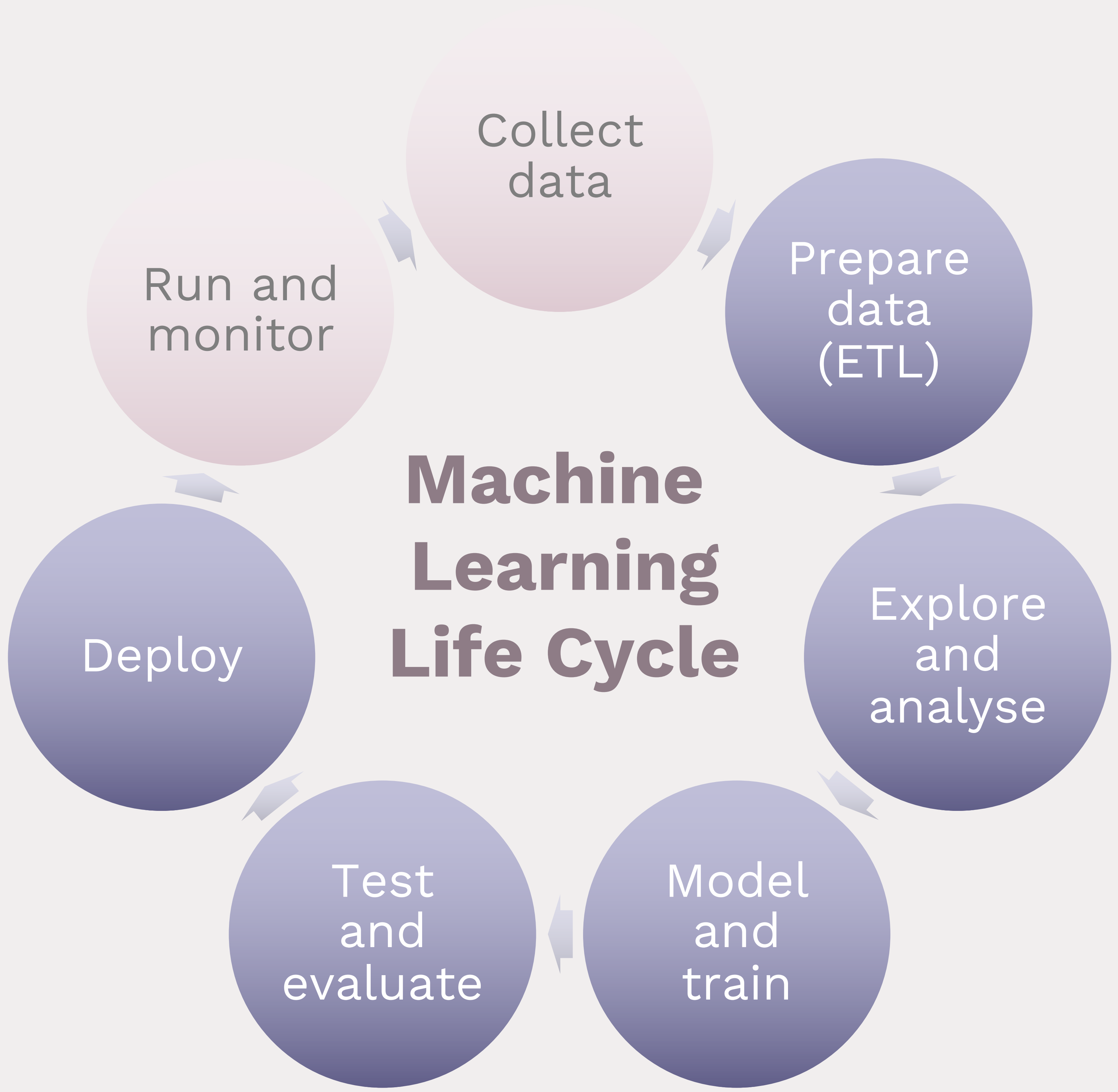


Inject trained models

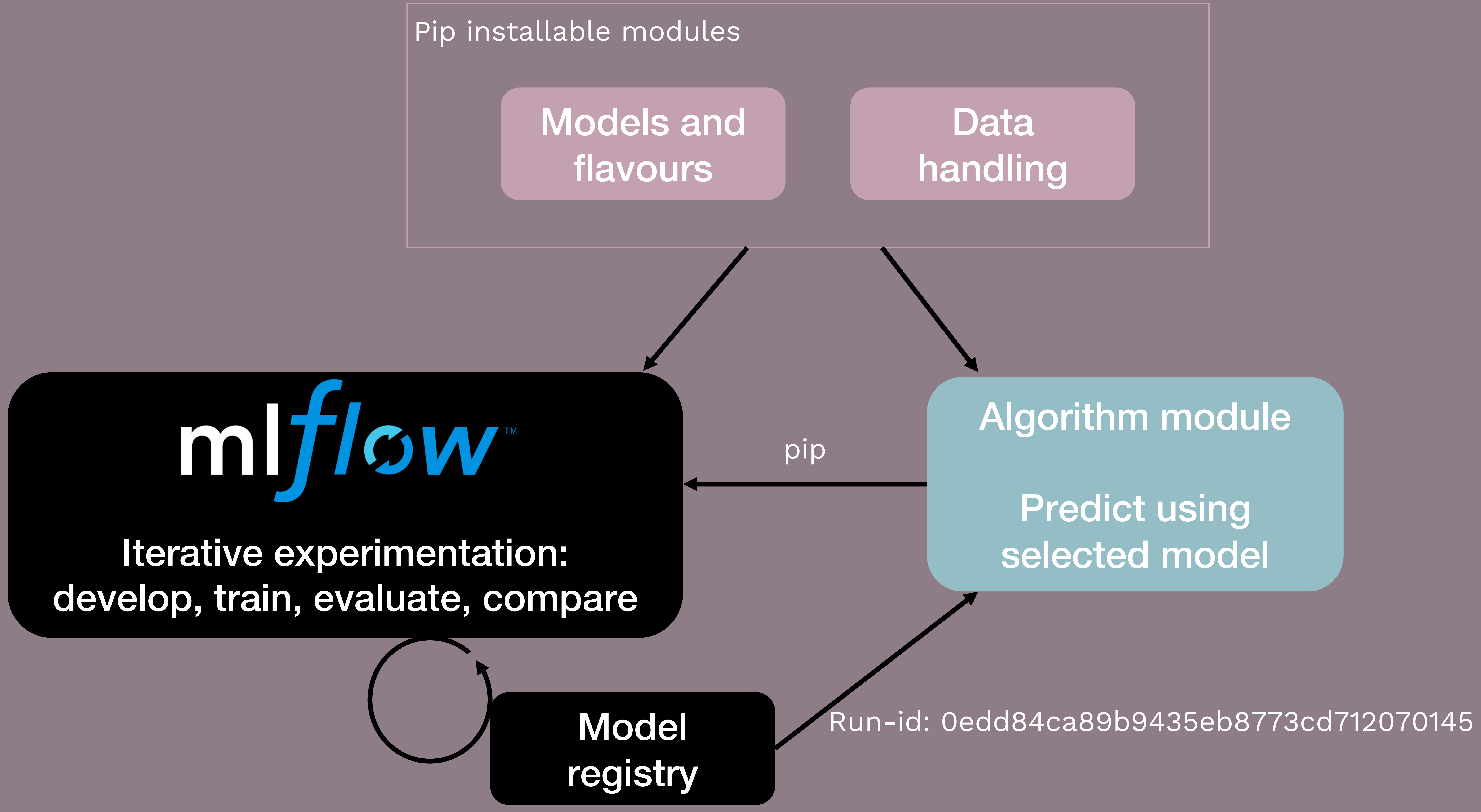


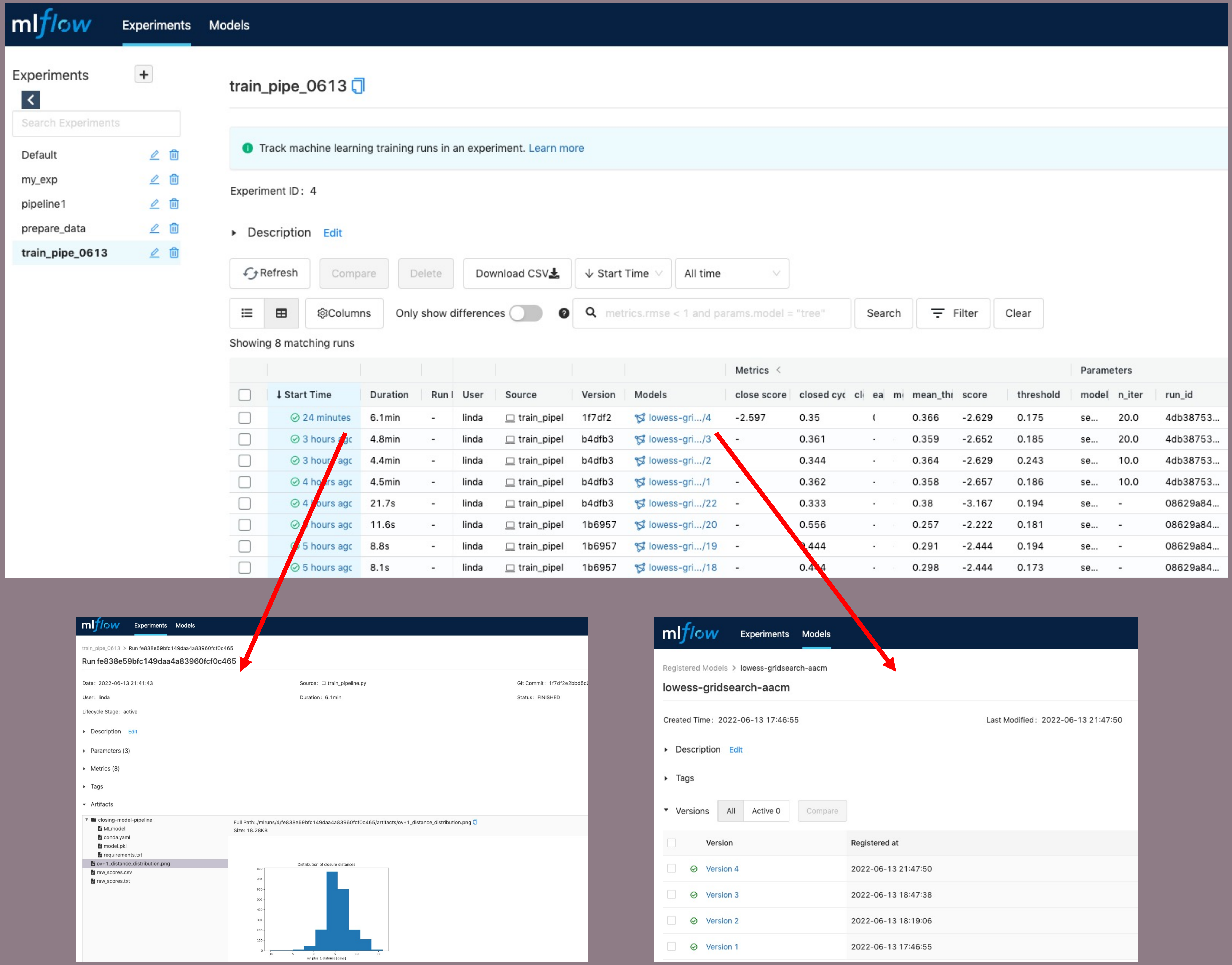
Most popular mlops tools for managing ML life cycle

Kubeflow	mlflow
Google	Databricks
Tensorflow (TFX)	Python
Pipeline based	Experimentation based
Deploy and manage complex ML systems	Tracking of experiments
Kubernetes infrastructure (more complex)	Local or cloud, eg develop locally and track remote



ML lifecycle Architecture





Experimentation

Try out different models and data sets

Track: parameters, metrics, and artifacts

Comparison between experiments

Model registry

Store trained models for later use

Load selected trained model by ID

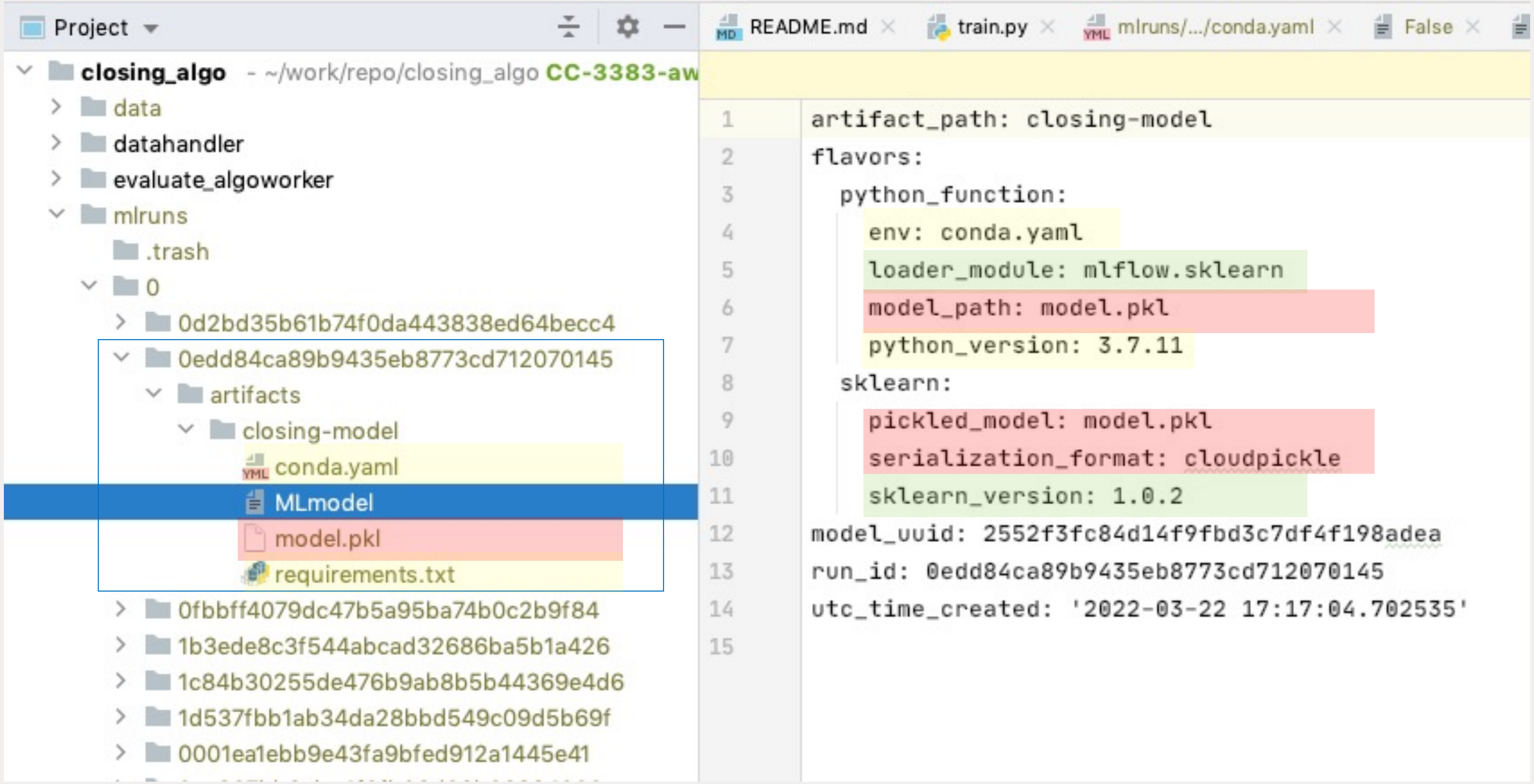
Advantages

Tracking of experiments

Reproducibility (models, params, data sets)

Easy deployment of trained models

Everything needed for serving model is automatically saved



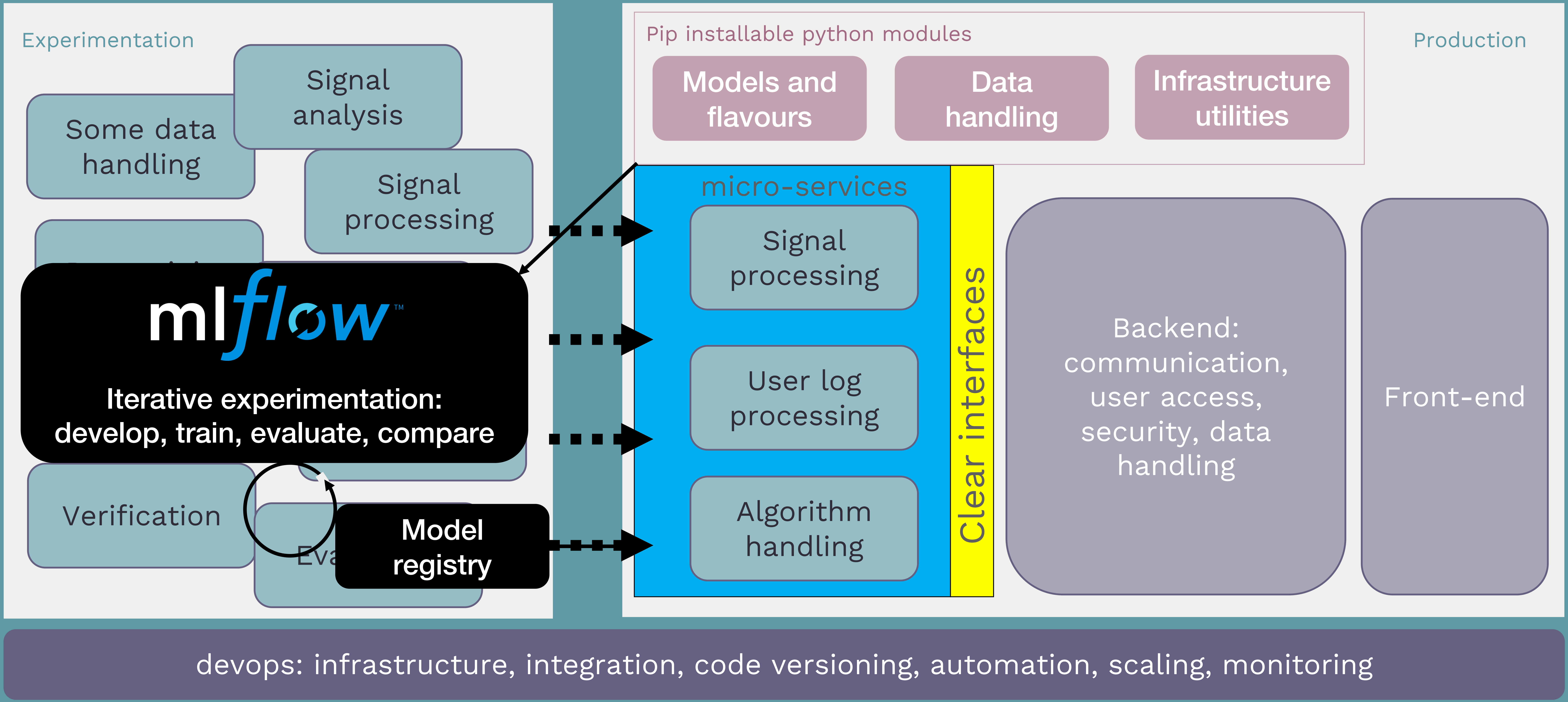
Follow pattern from *scikit-learn (sklearn)*

Standard classes eg:

- RandomForestClassifier
- BaggingClassifier
- GridSearchCV
- StandardScaler
- Pipeline

```
class FertilityModel(BaseEstimator, ClassifierMixin, metaclass=abc.ABCMeta):
    MODEL = "algo_model"
    MODEL_PARAMETERS = "algo_model_parameters"

    def __init__(self):...

    @abc.abstractmethod
    def fit(self, X=None, y=None, sample_weight=None, check_input=True):...

    @abc.abstractmethod
    def predict(self, cycle: CycleData, *args, **kwargs) -> (np.ndarray, np.ndarray):...

    @classmethod
    def get_model_name(cls):...

    def serialise(self) -> dict:...

    def score(
        self, cycle: CycleData, true_fertility_indications, sample_weight=None
    ) -> float:...
```

Mlflow for experimentation and model serving



Summary

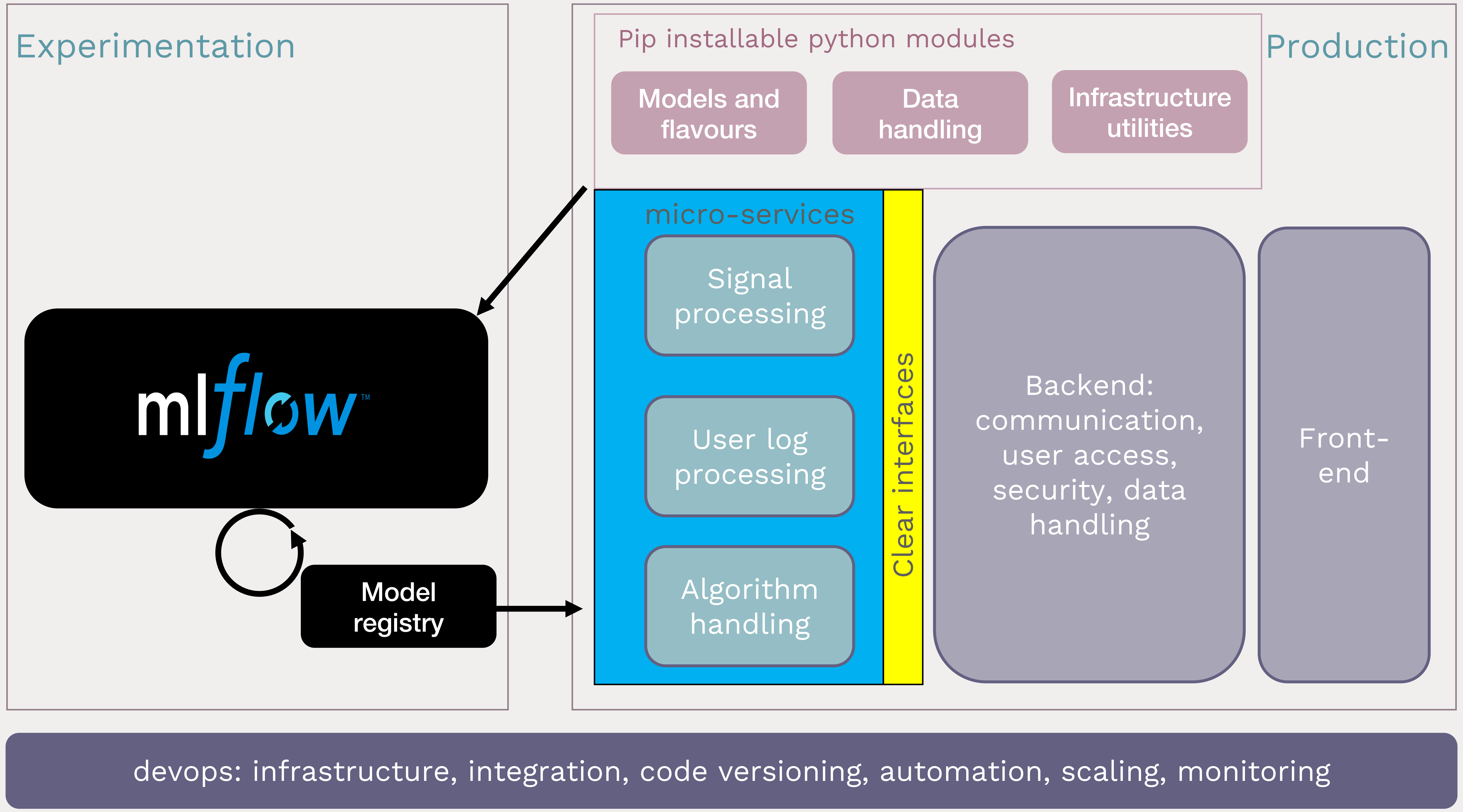
And challenges

Solution at ava

Separate data science modules as micro-services with clean interfaces to the rest of the system (most of the glue)

Separate parts of each data science module that is related to standard software tasks (some of the glue)

The core part is the models, the lifecyle is managed by a mlops tool and a trained model is stored in a blob



Remaining challenges

It is still possible to build a mess

- Follow best practices
- Recognize when code can be re-used
- Communication to prevent duplication

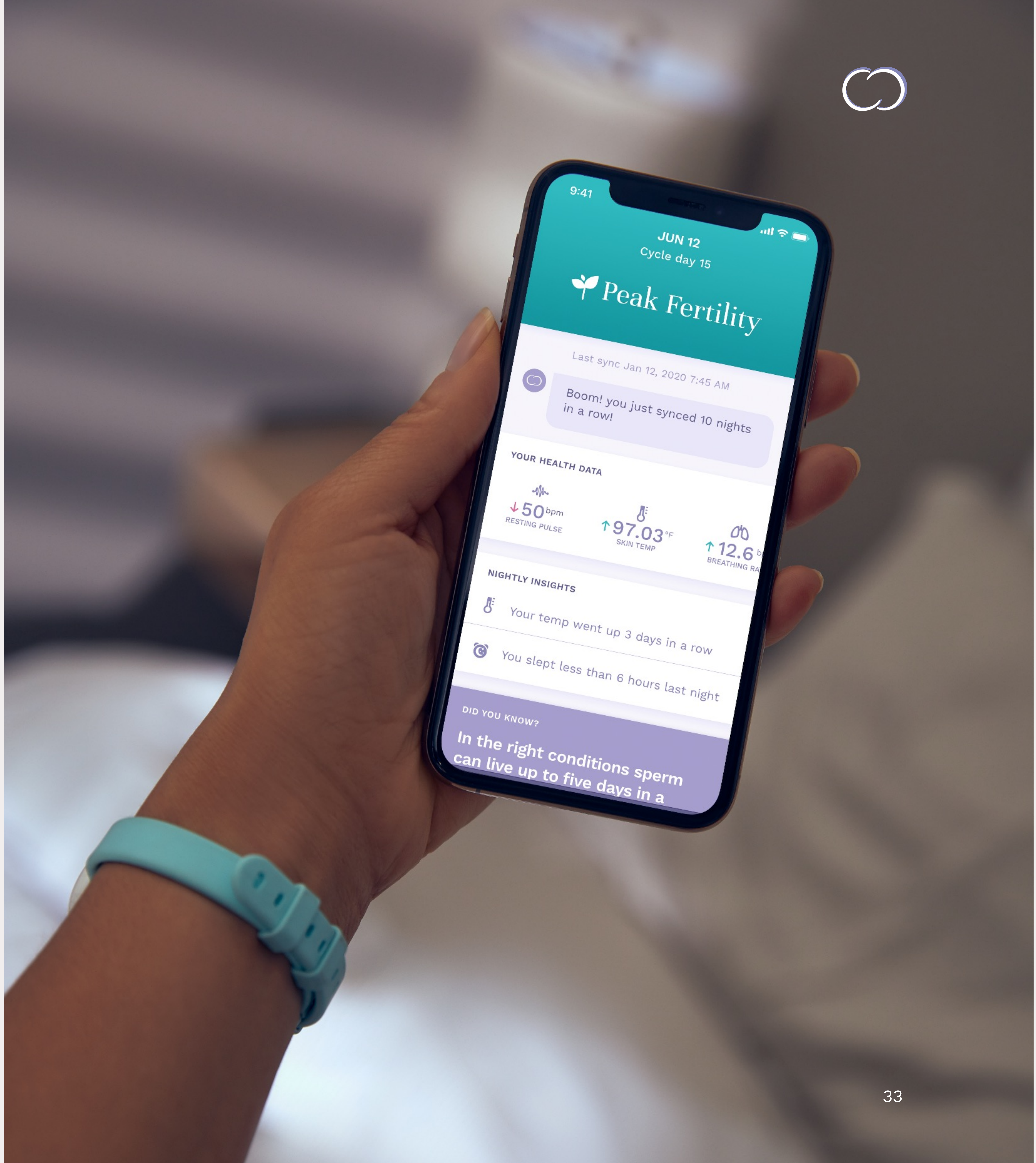
It is out of comfort zone for most researchers

- Different tools
- Outside core competences
- Need training and support
- Still some researchers will never go down this path...

ML engineer / SW developer is needed for parts

- To write the glue
- Use same libraries and language
- Coordination and communication

Not solving all problems (eg. Example 1)

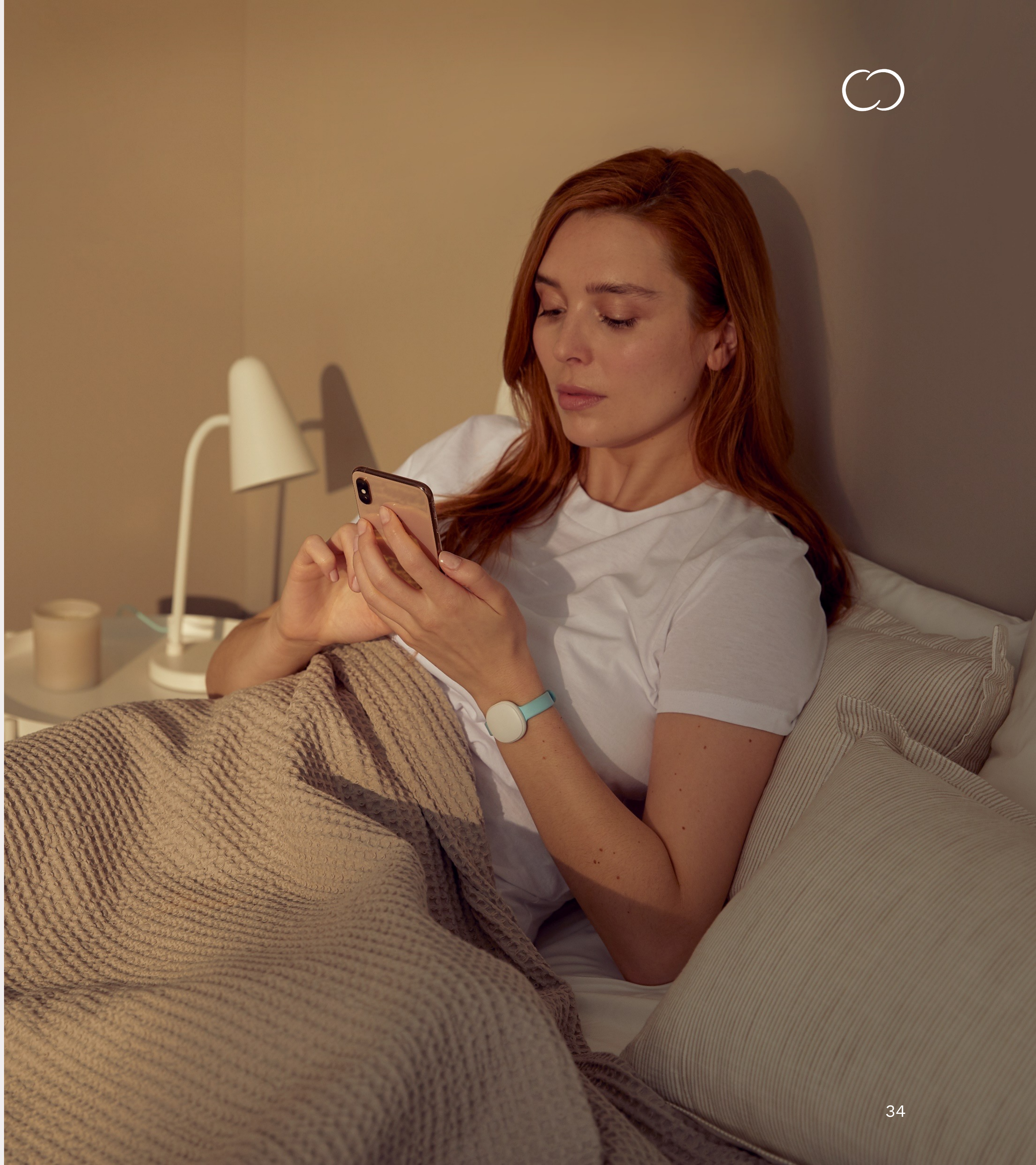


– and the solution is algile?

✓ Cross-functional teams:
  researchers and data engineers / sw devs
  T-shaped skills

✓ Pairing up, code review

✓ Work on different projects




Thank you!
Questions?

goto;

DON'T FORGET TO
RATE THE SESSIONS

#GOTOaar

Rate a minimum of **5 sessions** and
claim your **reward** at the
Registration Desk at the Trifork Hall