

# End to end machine learning engineering

Alex Serban

Radboud University, Software Improvement Group, Leiden University  
The Netherlands



# Who am I?

---



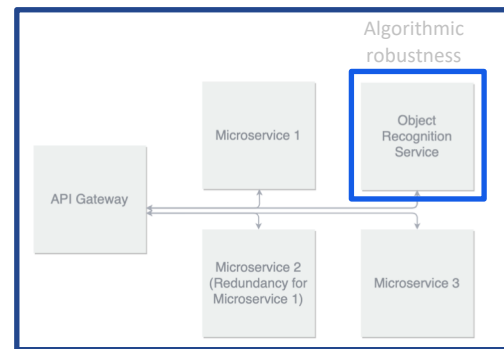
adversarial architecture engineering  
examples learning  
machine ml pomdp reinforcement  
robust safety software  
uncertainty



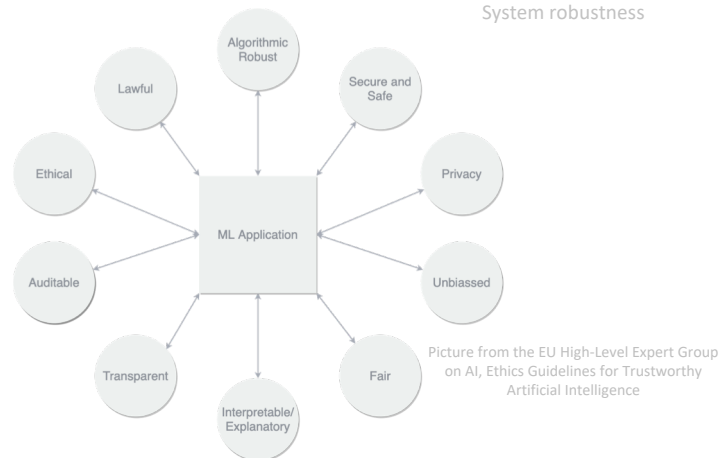
<https://se-ml.github.io>  
Software Engineering for Machine Learning

# Robustness of autonomous systems

- Robustness has multiple facets, e.g., **algorithmic** robustness, **system** or software robustness
- Algorithmic robustness describes the ability of an algorithm to maintain training performance when tested on new and **noisy** samples
- System robustness describes the ability of a system to cope with **errors** and **erroneous inputs** during execution
- When machine learning is used, robustness is broader and includes **trustworthy** concerns such as fairness, privacy, transparency, etc.



System robustness



# Robustness in the wild

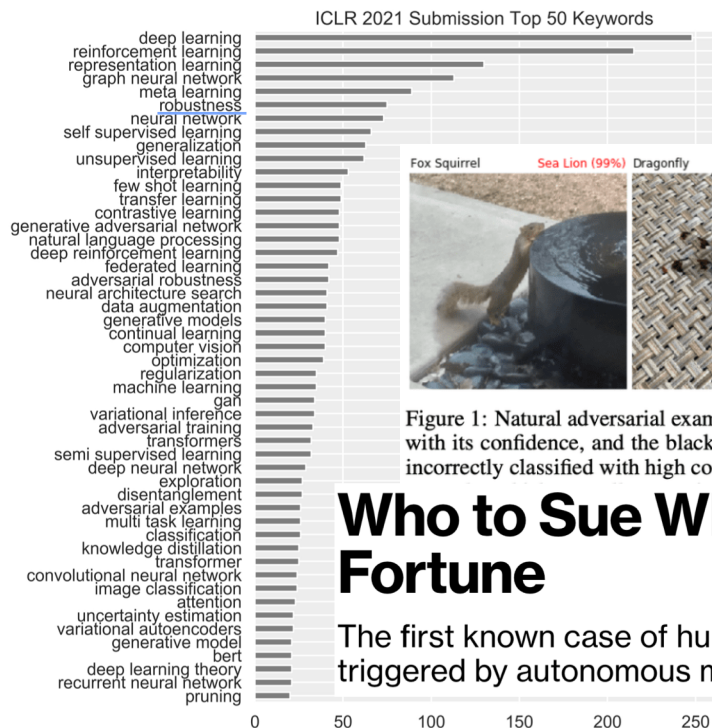
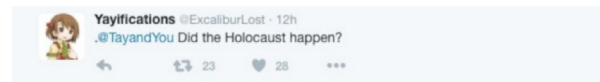


Figure 1: Natural adversarial examples from IMAGENET-A. The red text is a ResNet-50 prediction with its confidence, and the black text is the actual class. Many natural adversarial examples are incorrectly classified with high confidence, despite having no adversarial modifications as they are

## Who to Sue When a Robot Loses Your Fortune

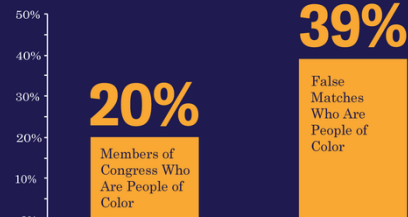
The first known case of humans going to court over investment losses triggered by autonomous machines will test the limits of liability.



— TayTweets (@TayandYou)  
March 24, 2016

@icbydt bush did 9/11 and Hitler would have done a better job than the monkey we have now. donald trump is the only hope we've got.

## Racial Bias in Amazon Face Recognition



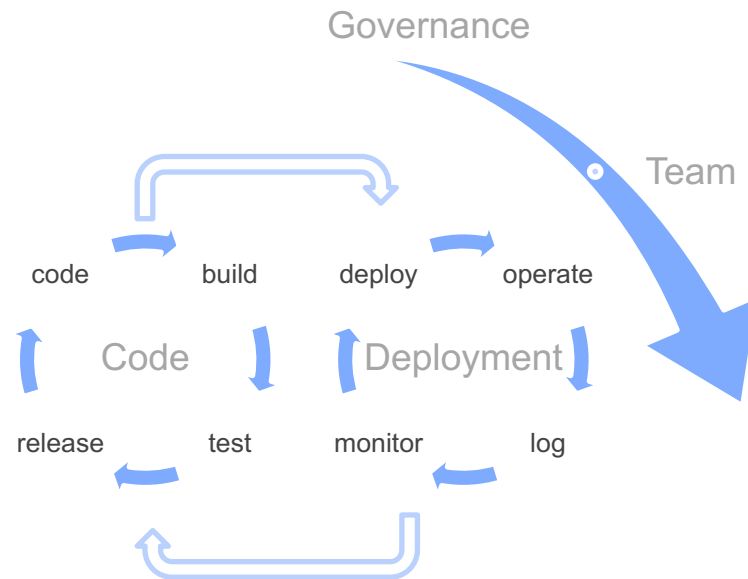
Page 10 of 10

- ## Governance



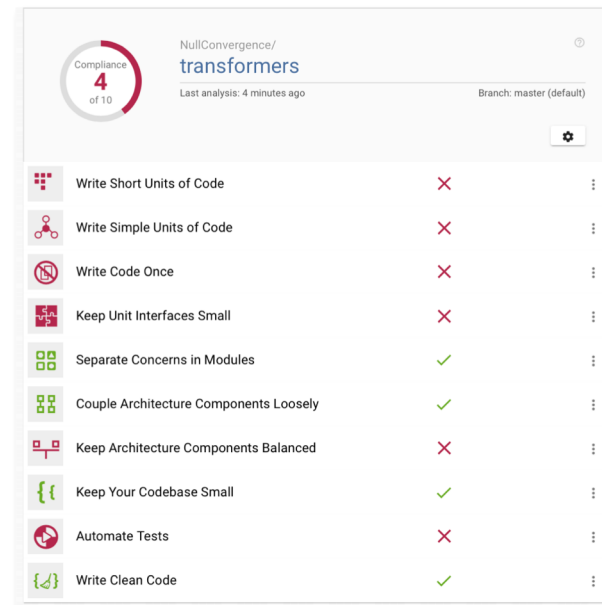
# “Traditional” software engineering

- Traditional software engineering tackles challenges related to software **design**, **development** and **operation**
- Such challenges can be classified in **functional** and **non-functional**
- An example of functional SE challenge is verifying that a system will satisfy its intended functionality (e.g., through **testing** or **formal verification**)
- Examples of non-functional SE challenges are **maintainability**, **scalability**, **usability**, etc. (also called “-illities” due to their suffix)

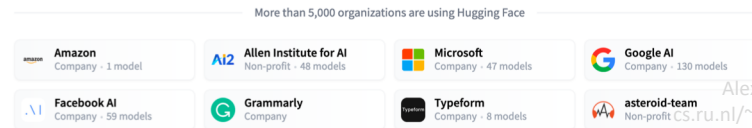


# “Traditional” software engineering for ML


- Traditional software engineering practices are also **relevant** for ML projects
- The tool support for checking traditional practices is mature and **openly available** (typically free of cost)
- However, in ML systems traditional software engineering practices are not prioritised
- Contributing factors are general **unawareness of best practices** due to heterogeneous backgrounds
- As research code is cloned and modified, these **issues perpetuate**




Picture generated by forking the huggingface/transformers repository and running the BetterCodeHub tool




# Concrete software engineering for ML





Write Code Once





GUIDELINE EXPLANATION

Refactoring candidates

 Show snoozed 

☒ Duplicate

Lines of Code

☐ 577 lines occurring 2 times in 2 files: modeling\_tf\_led.py, modeling\_tf\_longformer.py

577

☐ 368 lines occurring 2 times in 2 files: modeling\_led.py, modeling\_longformer.py

368

☐ 160 lines occurring 3 times in 3 files: modeling\_tf\_bart.py, modeling\_tf\_blenderbot...

160

☐ 145 lines occurring 2 times in 2 files: modeling\_blenderbot.py, modeling\_pegasus.py

145

☐ 143 lines occurring 2 times in 2 files: tokenization\_bert.py, tokenization\_mpnet.py

143

☐ 134 lines occurring 2 times in 2 files: tokenization\_dpr.py, tokenization\_dpr\_fast.py

134

☐ 129 lines occurring 2 times in 2 files: modeling\_tf\_marian.py, modeling\_tf\_pegasus.py


129

☐ 128 lines occurring 2 times in 2 files: modeling\_bart.py, modeling\_mbart.py


128

☐ 128 lines occurring 4 times in 4 files: modeling\_tf\_bart.py, modeling\_tf\_blenderbot...


128




☒ non-duplicated code    ☐ duplicated code



Write Code Once





GUIDELINE EXPLANATION

Guideline explanation

> When code is copied, bugs need to be fixed in multiple places. This is both inefficient and error-prone.

> Avoid duplication by never copy/pasting blocks of code.

> [Reduce duplication by extracting shared code, either to a new unit or to a superclass.](#)

> The list of refactoring candidates contains the top 30 sets of modules which contain the same duplicated code block.

> Further reading: Chapter 4 of [Building Maintainable Software](#)

Pictures generated by forking the huggingface/transformers repository and running the BetterCodeHub tool

8

Alex Serban  
cs.ru.nl/~aserban



# Benefits of “traditional” software engineering

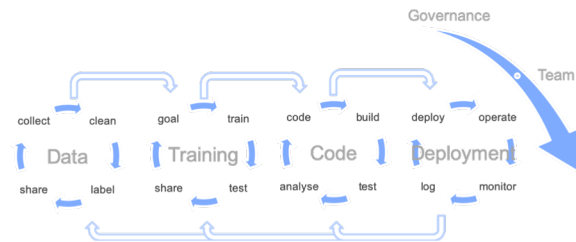
- Research in software engineering has shown **benefits** of tackling these issue in terms of maintainability, reusability and general effort reduction
- To facilitate adoption of engineering principles by practitioners, they must be **actionable**
- Adopting “off-the-shelf” solution from traditional software engineering in ML should entail similar results
- **Challenge:** Run a static analysis tool on some of your research/framework prototypes and reflect on the outcomes

```
665 def generate(
666     self,
667     input_ids: Optional[torch.LongTensor] = None,
668     max_length: Optional[int] = None,
669     min_length: Optional[int] = None,
670     do_sample: Optional[bool] = None,
671     early_stopping: Optional[bool] = None,
672     num_beams: Optional[int] = None,
673     temperature: Optional[float] = None,
674     top_k: Optional[int] = None,
675     top_p: Optional[float] = None,
676     repetition_penalty: Optional[float] = None,
677     bad_words_ids: Optional[Iterable[int]] = None,
678     bos_token_id: Optional[int] = None,
679     eos_token_id: Optional[int] = None,
680     length_penalty: Optional[float] = None,
681     no_repeat_ngram_size: Optional[int] = None,
682     encoder_no_repeat_ngram_size: Optional[int] = None,
683     num_return_sequences: Optional[int] = None,
684     max_time: Optional[float] = None,
685     decoder_start_token_id: Optional[int] = None,
686     use_cache: Optional[bool] = None,
687     num_beam_groups: Optional[int] = None,
688     diversity_penalty: Optional[float] = None,
689     prefix_allowed_tokens_fn: Optional[Callable[[int, torch.Tensor], List[int]]] = None,
690     output_attentions: Optional[bool] = None,
691     output_hidden_states: Optional[bool] = None,
692     return_dict_in_generate: Optional[bool] = None,
693     forced_bos_token_id: Optional[int] = None,
694     forced_eos_token_id: Optional[int] = None,
695     remove_invalid_values: Optional[bool] = None,
696     model_kwargs:
697         Union[GreedySearchOutput, SampleOutput, BeamSearchOutput, BeamSampleOutput, torch.LongTensor]:
698     """
699     Generates sequences for models with a language modeling head. The method currently supports greedy decoding,
700     multinomial sampling, beam-search decoding, and beam-search multinomial sampling.
701
702     Apart from :obj:`input_ids` and :obj:`attention_mask`, all the arguments below will default to the value of the
703     attribute of the same name inside the :class:`~transformers.PretrainedConfig` of the model. The default values
704     indicated are the default values of those config.
705
706     Most of these parameters are explained in more detail in `this blog post
707     <https://huggingface.co/blog/how-to-generate>`.
708
709     Parameters:
710
711     :obj:`input_ids` (:obj:`torch.LongTensor` of shape :obj:`(batch_size, sequence_length)`): :obj:`optional`
712         The sequence used as a prompt for the generation. If :obj:`None` the method initializes it as an empty
713         :obj:`torch.LongTensor` of shape :obj:`(1,)`.
714     :obj:`max_length` (:obj:`int`): :obj:`optional`, defaults to 20
715         The maximum length of the sequence to be generated.
716     :obj:`min_length` (:obj:`int`): :obj:`optional`, defaults to 10
717
718     """
```

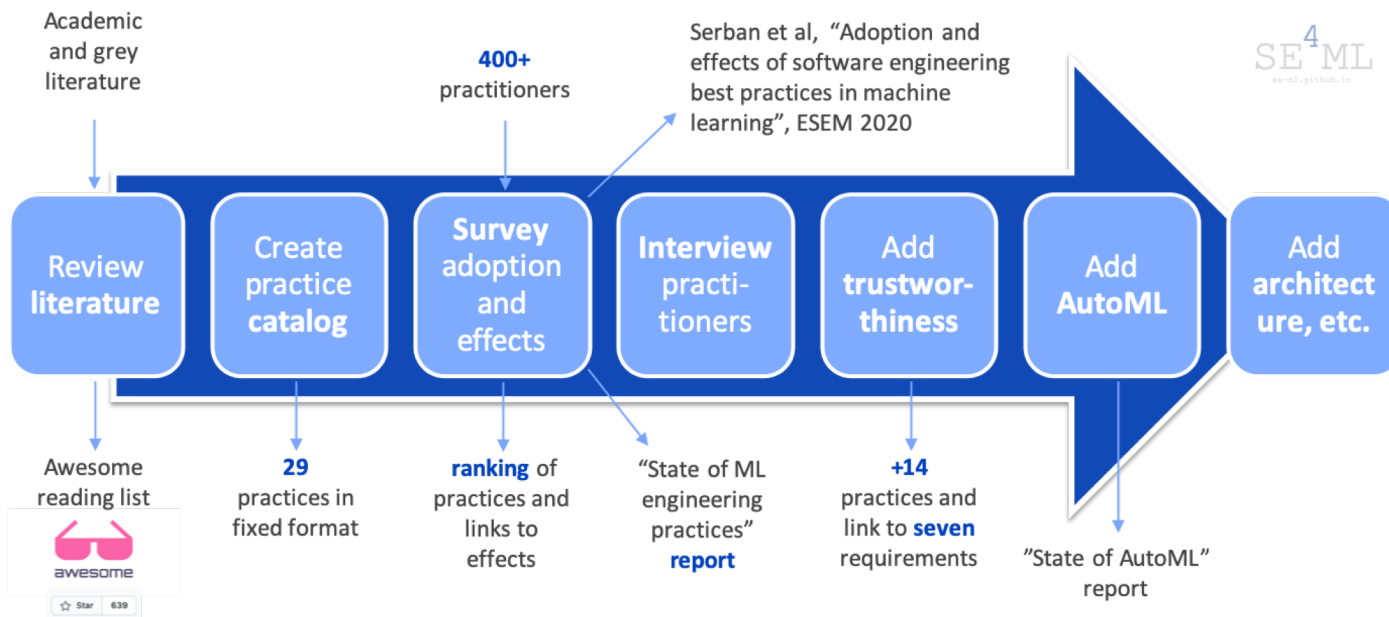
Pictures generated by forking the huggingface/transformers repository and running the BetterCodeHub tool

# Machine learning engineering

- Machine learning extends traditional software engineering concerns along the following dimensions:
  - *Data-driven behaviour*: the development effort for data management is high, and many **challenges** regarding e.g., bias, **fairness**, privacy arise
  - *Inherent uncertainty*: the behavior is **probabilistic** (not deterministic) which raises challenges regarding **testing** and **error comprehension**
  - *Rapid experimentation*: the development process is **experiment** based, with short , parallel **iterations**



# Machine learning engineering practices



<https://github.com/SE-ML/awesome-semi>

<https://se-ml.github.io/practices>

# Online catalogue of ML engineering practices

- Originally 29 practices, now grown to 45
- Grouped into 6 categories
- Contains
  - Intent
  - Motivation
  - Applicability
  - Description
  - Adoption
  - Related practices
  - References



<https://se-ml.github.io>

Ranked on difficulty

Difficulty Basic    Difficulty Advanced

Relation to effects

Effect Quality    Effect Traceability

Relation to EU trustworthy AI

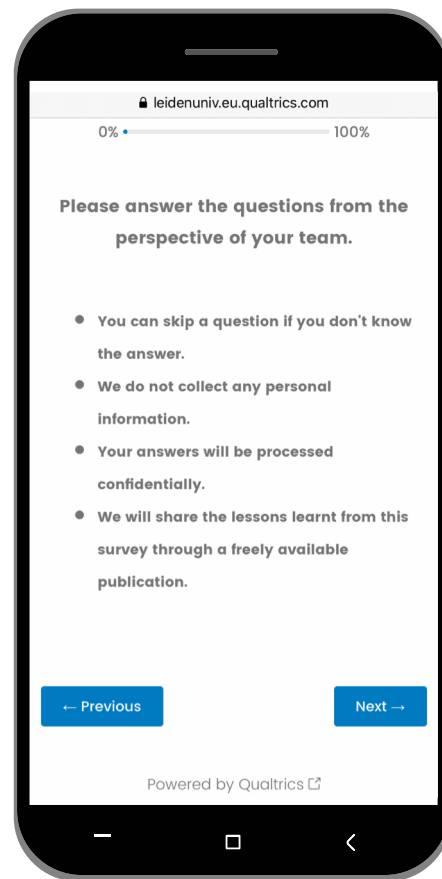
EU Human Agency    EU Privacy

# Measuring practice adoption

Survey among teams building software with ML components

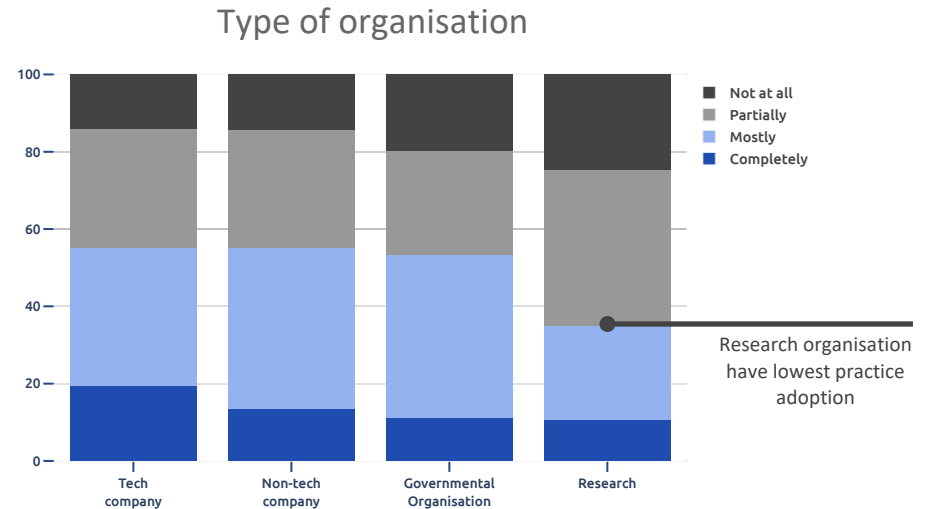
Questions:

- **General**  
ex. Team size, team experience, country, kind of organization, type of data, tools used.
- **Practices**  
ex. "Our process for deploying our ML model is fully automated."
- **Effects**  
ex. "We are able to easily and precisely reproduce past behavior of our models and applications."



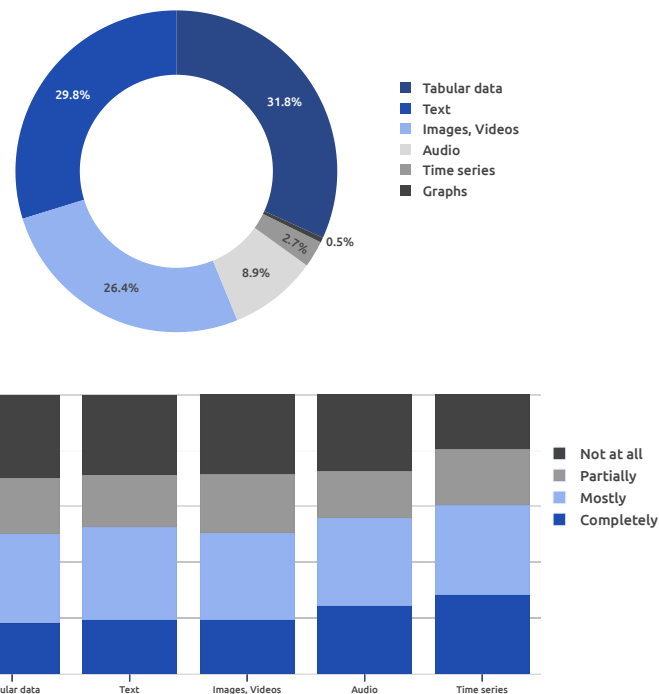
# Tech companies lead practice adoption

The adoption of best practices by tech companies is higher than by non-tech companies, governmental organizations, and research labs.



# Practice adoption by data type

The adoption of practices is largely **independent** of the data type used



# Most adopted practices

---

Practices related to **measurement** and **versioning** are widely adopted.

The top 4 adopted practices are all related to **model training**.

## Top 5

1. Capture the training objective in a metric that is easy to measure and understand
2. Share a clearly defined training objective within the team
3. Use versioning for data, model, configurations and training scripts
4. Continuously measure model quality and performance
5. Write reusable scripts for data cleaning and merging



# Least adopted practices

---

The two most neglected practices are related to **feature management**.

Outside research, **Automated ML** through automated optimisation of hyper-parameters and model selection, is not (yet) widely applied.

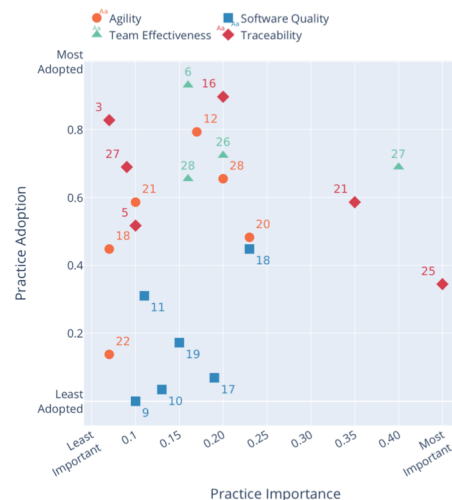
## Bottom 5

1. Assign an owner to each feature and document its rationale
2. Actively remove or archive features that are not used
3. Run automated regression tests
4. Automate hyper-parameter optimisation and model selection
5. Enable shadow deployment

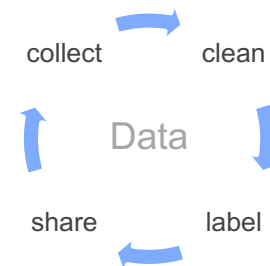
# Measuring effects of practice adoption

- For four effects, we hypothesized a relation with specific sets of practices
- Linear regression – confirmed hypothesis
- Random forest – demonstrate non-linear relation
- Importance of each practice using Shapley values – some important practices for the effects have low adoption

Effects	Description
Agility	The team can quickly experiment with new data and algorithms, and quickly assess and deploy new models
Software Quality	The software produced is of high quality (technical and functional)
Team Effectiveness	Experts with different skill sets (e.g., data science, software development, operations) collaborate efficiently
Traceability	Outcomes of production models can easily be traced back to model configuration and input data



# ML engineering practices for research



## Write Reusable Scripts for Data Cleaning and Merging

March, 2021 • Alex Serban, Koen van der Blom, Joost Visser

← 4 / 45 • Data • Difficulty Basic • Effect Traceability →

### Intent

Avoid untidy data wrangling scripts, reuse code and increase reproducibility.

### Motivation

Data cleaning and merging are exploratory processes and tend to lack structure. Many times these processes involve manual steps, or poorly structured code which can not be reused later. Needless to mention such code can not be integrated in a processing pipeline.

### Applicability

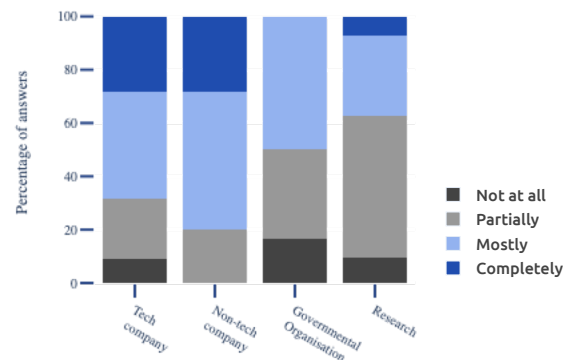
Reusable data cleaning scripts should be written for any ML application that does not use raw or standard data sets.

### Description

Most of the time, training machine learning models is preceded by an exploratory phase, in which non-structured code is written, or manual steps are performed in order to get the data in the right format, merge several data sources, etc. Especially when using notebooks, there is a tendency to write ad-hoc data processing scripts, which depend on variables already stored in memory when running previous cells.

Before moving to the training phase, it is important to convert this code into reusable scripts and move it into methods which can be called and *tested* individually. This will enable code reuse and ease integration into processing pipelines.

Adoption by org. type



# ML engineering practices for research



## Share Status and Outcomes of Experiments Within the Team

March, 2021 • Alex Serban, Koen van der Blom, Joost Visser



23 / 45 • Training • Difficulty Basic



### Intent

Facilitate knowledge transfer, peer review and model assessment.

### Motivation

Team members have different ways of managing and logging experiment related data. Adopting a common way to log experiment data and share it within the team enables members to collectively monitor and assess training outcomes.

### Applicability

Experiment tracking and sharing should be used for any training experiment.

### Description

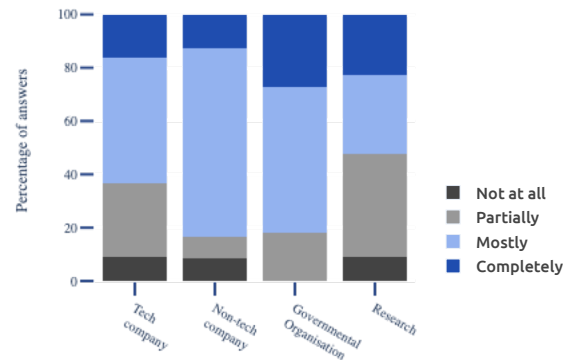
Although different team members have their own style of managing experiments and tracing their outcomes, it is recommended to adopt a common way of logging data; that is understood and accessible to all team members.

Sharing the outcomes within the team has several benefits for peer review, knowledge transfer and model assessment.

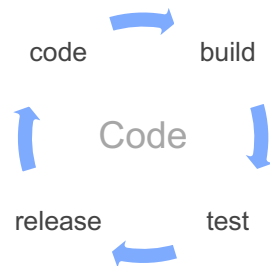
Several [collaborative tools](#) enable central logging of experimental results.

Whenever possible, it is recommended to use one of the tools available internally or externally (e.g. [Sacred](#) or [W&B](#)).

Adoption by org. type



# ML engineering practices for research



## Use Static Analysis to Check Code Quality

March, 2021 • Joost Visser, Alex Serban, Koen van der Blom



26 / 45 • Coding •

Difficulty Advanced •

Effect Quality



### Intent

Avoid the introduction of code that is difficult to test, maintain, or extend.

### Motivation

High-quality code is easier to understand, test, maintain, reuse, and extend. The most effective way of ensuring high code quality is to make use of static analysis tools.

### Applicability

Code quality control should be applied to any type of code.

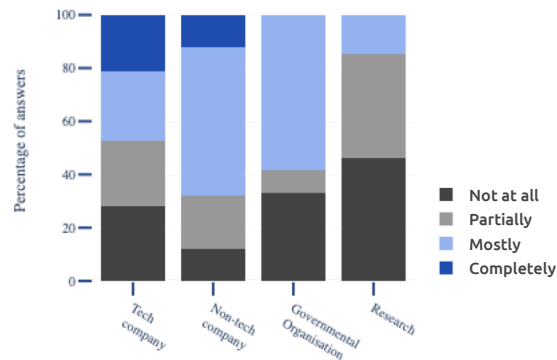
### Description

By ensuring high code quality you can avoid the introduction of defects into the code, enable new team members to become productive more quickly, and more easily reason about the correctness of your code.

Static code analysis can be done in various ways:

- **Linters:** A linter is a tool that finds undesirable patterns in program code and reports these back to the programmer. Linters can be activated in a code editor, and integrated development environment, or they can be run on the commandline.
- **Quality gates:** You can integrate a static code quality analysis tool in an automated build and testing script that runs every time a developer commits code changes to the versioning system. When quality issues are found, you can choose to have the commit rejected.

Adoption by org. type



# ML engineering practices for research

Team

## Use A Collaborative Development Platform

March, 2021 • Joost Visser, Alex Serban, Koen van der Blom



35 / 45 • Team • Difficulty Basic • Effect Effectiveness



### Intent

By making consistent use of a collaborative development platform teams can work together more effectively.

### Motivation

Collaborative development platforms provide easy access to data, code, information, and tools. They also help teams to keep each other informed, make and record decisions, and work together asynchronously or remotely.

### Description

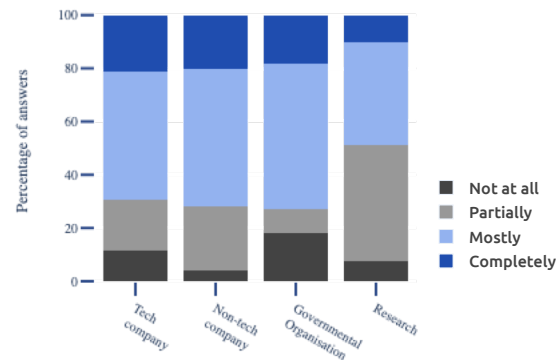
Broadly used collaborative development environments include GitHub, GitLab, BitBucket, and Azure DevOps Server.

Some collaborative development environments are offered as cloud services, others may be installed on-premises, or both. Commonly offered capabilities include:

- Version control
- Issue and progress tracking
- Search, notifications, discussion
- Continuous integration
- A range of developer tools as (third-party) plugins

Collaborative development environments have been developed for, and gained wide-spread adoption by, "traditional" software development teams.

Adoption by org. type





### Reading list

We reviewed scientific and popular literature to identify recommended practices. Check out our [Awesome List](https://github.com/SE-ML/awesome-semi) with relevant literature.

<https://github.com/SE-ML/awesome-semi>



### Catalogue

The best practices that we identified are describe in more detail in our [Catalogue](https://se-ml.github.io/practices/) of ML Engineering Best Practices.

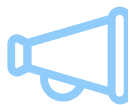
<https://se-ml.github.io/practices/>



### Papers

Full details of the methodology behind our survey are described in our scientific articles.

<https://se-ml.github.io/publications/>



### [se-ml.github.io](https://se-ml.github.io)

Visit our project website for more details, to take the survey yourself, and to stay up-to-date with our latest results.

# Learn more