WASP – Software Engineering and Cloud Computing
Assignment Module 2: Software Engineering

Willem Meijer (willem.meijer@liu.se)
June 13, 2025

Introduction

The overarching goal of my research is to improve quality assurance methods in systems that integrate machine learning (ML) components. This is predictive ML, so not generative ML, like LLMs. This type of software is increasingly present in modern systems and is expected to grow substantially in the future. However, even though companies and individuals *want* to integrate ML components and the technology is generally there, *how* they integrate them effectively is generally underexplored. This squarely placed my work inside SE4ML, where we explore the application of verification and validation principles to ML development.

ML algorithms commonly make very stringent assumptions about their data, like formats, distributions, or the type of information represented in the data. However, although most ML algorithms will naturally run on data that does not conform to these assumptions (during training and inference), the algorithm's effectiveness will generally be negatively affected. This could be obvious, like the prediction quality dropping substantially. However, this impact is generally much more subtle (e.g., a gradual decline over time) or only present in a particular subset of your data (e.g., people of a particular age group or data generated by specific devices). Specifically, our project addresses if and how ML engineers currently address these issues, develop tooling to support them in this process, and ultimately integrate this tooling into development pipelines.

At a high level, our research questions are the following: What are the development and integration issues faced by ML engineers? How can tooling detect these issues while writing ML code? And how can such tooling be integrated into ML engineering pipelines to detect regressions? Overall, this will allow companies, institutions, etc. that deploy systems with ML components to safeguard their systems' quality.

Lecture principles

Because my background is in software engineering, with a great interest in human aspects of software engineering and some formal education in psychology, most of the module's topics were not new to me. However, some of the testing methods to verify and validate your ML components and algorithms were quite insightful. Specifically, invariant/metamorphic and anchor testing, and data slicing, which could be applied in the context of intersectional biases in data/ML algorithms; a topic I might soon start a new collaboration on, but have not yet had the time to read up on. Invariant and anchor testing allow us to evaluate whether an algorithm does or does not latch onto specific features. In the context of fairness and discrimination in ML algorithms, this is incredibly important as it allows us to identify whether the algorithm classifies according to, for example, sensitive properties (gender, socio-economic status, etc.) if these are present in raw data. Additionally, if a party (company, institute, etc.) makes a case that particular sensitive feature is imperative to their use-case (e.g., as income is to bank loans), we could potentially leverage an anchor-like approach to systematically evaluate whether this is indeed one of the core features of the model, or merely an

auxiliary one, and judge whether its use is warranted. Finally, as was mentioned in the lecture, data slicing could be useful to identify whether a ML algorithm performs better on cases with particular properties compared to others, allowing us to, for example, identify misclassified edge cases that we could emphasize during future training or use as a candidate population for additional data collection. A benefit of these three methods is that they can be applied to test many ML solutions, as they can be deployed as black-box testing methods. Considering ML algorithms (and pipelines) change frequently, this would make it a powerful and sustainable tool.

Guest lecture principles

Although not entirely new, one idea I took from the guest lectures is Per Lenberg's description of risk-averse decision-making when introducing new components into existing and new systems. Clearly, his case is specific to aeronautics, a highly regulated discipline where it is inherently challenging to verify the quality and correctness of your system from a technical perspective. However, the importance of social factors, to which ML components are especially subject, as distrust in them and the fact that they challenge the status quo, is easy to forget. In some sense, this affects my work twofold. First, because I will develop a tool that could be adopted in company settings, which will be subject to such resistance. Second, because the tool provides a series of guarantees about components that might be integrated into the larger system, making them more trustworthy, thus reducing distrust.

Data scientists versus software engineers

Data scientists and software engineers. Although I am certain it holds for some people, I think the book's description of data scientists and software engineers is quite reductive. It is certainly not sustainable as, for example, the number of (university) programs on data science increasingly teach a combination of software engineering and data courses. I am sure there are global differences; there have to be, as *"software engineer"* and *"data scientist"* are two extremely broad terms. The definition of a software engineer ranges from web (and cloud) engineering (which is more or less what the book uses) to embedded systems engineering to requirements engineers (both of which the book disregards completely) Similarly, a data scientist could do anything from reinforcement learning to pure math to basic statistics. Consequently, I think that comparisons between these classes of people are not very useful when they are not placed in context. If people choose to specialize in their respective domains, the differences will clearly be great. However, if they do not, the difference is more or less meaningless. It is only a simple decision and a matter of priorities, after which it will be much more useful to call them a data pipeline operator, embedded RLOps developer, applied statistician, or ML product owner — names that are specific to what they actually do.

Role evolution. Whether roles will evolve depends greatly on context as well. While everybody does everything in start-ups, you could specialize in your favorite niche when you work at a huge organization. Sure, once operationalization and deployment of ML software become mainstream, you will need people with a different set of skills. A company that toys around with ML pipelines will not need an ML monitoring expert, whereas a company with 30 live ML services certainly does. In my experience, after talking with several companies, the maturity of ML systems differs wildly right now. However, as these systems generally mature, new data engineers will increasingly need

specialized skills, which inevitably involves learning from *"the other side."* An ML monitoring expert will, for example, need to be able to interpret trends in both ML-specific and software-specific performance metrics, like classification error and CPU utilization, contextualize these with respect to the task they fulfill and the greater system they are part of, and design interventions when and where necessary.

Paper analysis

Data smells. The work of Recupito et al. (2024), which builds on the work of Foidl et al. (2022), lies close to my work, proposing a taxonomy of *"data smells"* in learning software. These are errors in ML pipelines' input data, which can have negative consequences on the system's quality. In this context, they relate these smells to dataset quality, measured using metrics of uniqueness, consistency, readability, and completeness. To some extent, these metrics indicate the amount of information contained in the dataset and the dataset's cleanliness. They find a total of 50 smells, comprising eight overarching themes that range from low-level issues, like storing data in the wrong format or missing values, to higher-level issues, like outliers, correlated features, and inclusion of sensitive features. They continue to identify how frequently these smells occur in datasets, and, ultimately, whether the presence of the top five smells affects dataset quality. Unsurprisingly, most of these smells relate to dataset quality; however, they affect it in different ways, as some smells positively affect these quality metrics.

This paper takes a very data-oriented perspective on ML engineering — which is completely fair. However, my work (Meijer, 2024) builds upon it by evaluating whether such smells are present in ML pipelines, and, more importantly, whether they are dealt with. For example, if you remove some of the correlated features from your dataset, it does not matter whether they were originally present, as you dealt with them accordingly. Specifically, we currently work on a static analysis tool for learning software, evaluating incompatibilities between datasets and algorithms.

Static analysis. Various solutions exist already that claim to detect these issues. For example, the static analyzer shared in the lecture (Quaranta et al., 2024), or the second CAIN paper I want to discuss (Shivashankar and Martini, 2025). The work of Shivashankar and Martini (2025) proposes a linter for ML smells, capturing 76 smells distributed across five popular ML libraries by analyzing the abstract syntax tree (AST) of ML pipelines written in Python. They performed a small-scale evaluation, finding that its precision is 1 for all classified notebooks and its recall ranges between 0.75 and 1, suggesting a limited number of false flags (which static analyzers are notorious for). Further, their small-scale user study suggests the usefulness of such a tool in practice, scoring between 3 and 4 with 0.3 to 0.8 variance on a Likert scale of 1 to 5.

However, one major drawback of these tools is that they generally do not comprehensively address the relationship between algorithms and data, but instead re-implement primitive (and long-known) rules, like placing your dependencies at the top of your script (Quaranta et al., 2024), or make strong assumptions about the the relationship between different function calls, for example, using whether the number of unique elements in any feature of a dataset is printed as an indicator of class imbalance (Shivashankar and Martini, 2025). Consequently, in my experience, the tests done are either not specific to ML at all or yield an unnecessary number of false reports and misses due to unruly assumptions. Although the reported evaluation by Shivashankar and Martini (2025) is positive, I am not convinced in the slightest that it will translate well to real development environments, as printing unique values is by no means an indicator of the imbalance of your

classes. Arguably, it is a good practice and could lead to the healthy conclusion that there is no class imbalance. I zoom in on class imbalance, as I consider it is a fundamental issue in many ML algorithms, however, I noticed similar issues with their rules regarding improper feature selection, which draw a strange relationship with validation strategies, rules regarding large dataset loading, which require users to print the file size to enable the rule, and the generally very strong assumptions about variable and file names (for example, any file that has a name starting with `test_` or `helper` will never violate a rule).

Adaptation. Both of these lines of research (Recupito et al., 2024; Shivashankar and Martini, 2025) are essential to correct ML development. It is already easy to make mistakes in *"regular software."* In my opinion, this is only amplified in ML software due to its inherent complex relationship between code and datasets — especially when you consider that a large portion of *"bugs"* in ML software do not cause crashes but present themselves as bias, poor generalization, etc., which exclusively occur after you have trained your model. Obviously, training is a very time-consuming and costly task, making the detection of these issues before training economically interesting; beyond the social benefits they provide by preventing the deployment of buggy ML software.

Because of its importance and the limited depth of existing solutions, my work (Meijer, 2024) tries to push the envelope by actively involving data in static analysis for ML code. Going back to the class imbalance example, this will allow us to highlight this problem exclusively when such an imbalance is problematically present in the data. This allows us to test the problems given by Recupito et al. (2024), but also allows us to go much further, testing for grouping factors, distribution assumptions, cardinality, dimensionality, etc. as well. This allows us to find functional incorrectness in ML software, but also non-functional issues, as some algorithms *"only"* become much less time or memory-efficient given particular input data.

Going into a mild amount of detail, we use ASTs of ML code written in Python (like Shivashankar and Martini (2025)) and translate it to a construct similar to to control-/data-flow graphs. These graphs have real datasets at their roots, which can be accessed to calculate data properties (data ranges, means, correlation, etc.; essentially, any property you can think of). Then, for each ML method (e.g., the `fit` method in `sklearn`'s `LogisticRegressionClassifier`), we can identify its input data and hyperparameters, what properties we expect the data to have with respect to those hyperparameters, and how they have been transformed throughout the pipeline to see if the data still has those properties. Of course, this process is still subject to assumptions and will inevitably cause false classifications; however, compared to existing solutions (Shivashankar and Martini, 2025; Quaranta et al., 2024), our assumptions are at the very least inherent to ML and are directly connected to the thing we make the assumptions about: the data and how it is processed.

Research ethics and synthesis reflection

My search strategy was fairly straightforward. Because my research is closely related to the articles published in CAIN, I picked papers that are directly relevant to my work and emphasized recent ones. First, I searched through the papers in my reference manager, specifically ones I cited previously. Through Meijer (2024), where I cited various CAIN papers, I found Recupito et al. (2024). After, I sieved the titles of the 2025 Research and Experience Papers[1] for relevant works,

[1] https://conf.researchr.org/track/cain-2025/cain-2025-call-for-papers#Papers-Accepted

finding Shivashankar and Martini (2025), which is the only one directly related to my work. The work by Vonderhaar et al. (2025) is likely closely related to a future study, so I could have discussed that too.

Although I did not read all abstracts, I did not spot any misleading titles/abstracts. Although I think that Shivashankar and Martini (2025) is not a great paper in either its solution or its evaluation, it is one of the more complete ML static analysis tools I have seen as of now.

Beyond Grammarly as a spelling/grammar checker and ChatGPT to find some synonyms, I have not used LLMs for this assignment whatsoever.

References

Foidl, H., Felderer, M., and Ramler, R. (2022). Data smells: categories, causes and consequences, and detection of suspicious data in ai-based systems. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*, pages 229–239.

Meijer, W. (2024). Contract-based validation of conceptual design bugs for engineering complex machine learning software. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, MODELS Companion '24, page 155–161, New York, NY, USA. Association for Computing Machinery.

Quaranta, L., Calefato, F., and Lanubile, F. (2024). Pynblint: A quality assurance tool to improve the quality of python jupyter notebooks. *SoftwareX*, 28:101959.

Recupito, G., Rapacciuolo, R., Di Nucci, D., and Palomba, F. (2024). Unmasking data secrets: An empirical investigation into data smells and their impact on data quality. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI*, pages 53–63.

Shivashankar, K. and Martini, A. (2025). Mlscent a tool for anti-pattern detection in ml projects. In *Proceedings of the 4th international conference on AI engineering: software engineering for AI*.

Vonderhaar, L., Elvira, T., and Ochoa, O. (2025). Generating and verifying synthetic datasets with requirements engineering. In *Proceedings of the 4th international conference on AI engineering: software engineering for AI*.