Software Engineering Assignment

Alex Ionescu

1. Introduction

My research is in the area of language-based security, which consists of applying programming language-based techniques to analyze and improve the security of software.

My project involves applying language-based security techniques in the domain of cloud computing. More specifically, I'm working on *choreographic programming*, a programming paradigm that allows for designing distributed systems such that communication-related deadlocks are caught at compile-time (a property referred to as "deadlock-freedom by construction").

To achieve this, I'm leveraging HasChor [1], a library that enables choreographic programming in Haskell through an embedded domain-specific language (DSL). HasChor relies on Haskell's strong type system and clear separation between pure and effectful code to provide the deadlock-freedom guarantees at compilation time.

2. Lecture principles

Property-based testing

One of the concepts presented in the lectures that I find particularly useful is property-based testing (PBT). It allows developers to define the *properties* that are being tested as simple functions returing booleans, instead of having to manually design test-cases. (However, designing properties can also prove complex sometimes.) The PBT framework then automatically generates random test-cases and checks whether they satisfy the property. Those that fail are further *minimized* (reduced to the smallest size that still fails) and reported to the user, providing minimal error-cases that are easy to debug.

As I mostly use Haskell for my research, property-based testing is my preferred approach to testing, thanks to the excellent QuickCheck [2] library.

It allows me to test complex properties with relatively little testing code.

Behavioral software engineering

Another relevant concept presented in the lecture is that of behavioral software engineering (BSE), which calls for focusing not just on technical solutions and tools, but also on the psychological and human aspects of software engineering. I think these aspects are very important, but also often overlooked. Furthermore, some parts of behavioral software engineering are applicable when conducting research as well, especially being mindful of cognitive biases when collecting and analyzing data, or even when deciding which ideas are worth pursuing.

3. Guest lecture principles

Cost of defect removal

An important idea presented during Julian Frattini's guest lecture is understanding (and mitigating) the cost of defect removal. He mentions that the cost of removing a defect scales approximately by a factor of 10 for each phase the defect survives. As my research focuses on programming language-based techniques, this notion is very familiar to me, and is one of the main motivators for improving safety and security at the programming language level, since it leaves no (or very little) room for defects (such as deadlocks, in the case of choreographic programming) and security issues to "escape" farther into the lifecycle of the software product.

Problem vs. solution space

Another concept I found useful from the guest lecture is knowing the difference between the problem- and solution-space. The lecture made me realize that, as a developer, I mostly reason in terms of the solution-space (e.g. with statements such as "Tool/library/programming language X is better than Y"), and much more rarely in terms of the problem-space. However, this could lead me to use tools that are poorly-fit for the task or problem I'm trying to solve. This idea is also useful to keep in mind when designing domain-specific languages (which I do as part of my research), as their primary goal is to allow users to reason as closely to the "domain" (i.e. problem-space) as possible.

4. On Data Scientists vs. Software Engineers

On essential differences between these roles

I agree with the differences between Data Scientists and Software Engineers presented in the MLIP book [3]. The two roles place their focus on different aspects when developing an ML-enabled system or product, as a result of their different skillsets and (usually) backgrounds. Data scientists tend to focus purely on optimizing model performance, while software engineers take a more holistic view of the system being developed, balancing different (and often conflicting) considerations such as maintainability, performance, and cost.

Will the roles converge or specialize further apart?

In my opinion, we will see both phenomena play out. On the one hand, due to the rapid pace of evolution in ML, the skills required to architect competitive models will likely become much more specialized. At the same time, advancements in computing infrastructure (such as cloud and edge computing) and deployment scenarios (IoT devices, autonomous vehicles, VR/AR headsets etc.) will require more expertise to navigate their trade-offs, especially in the areas of DevOps/DevSecOps/MLOps.

On the other hand, neither of these roles operate in a vacuum, and a key component to effective collaboration is understanding the mindset and goals of each other, which I believe will incentivize developers on both ends of the spectrum to gain at least a minimal amount of training in the other role's domain. In my view, this is not unique to ML, and is also applicable to other specialized sub-domains of software engineering, such as video game development (with the analogy "Data Scientist : Software Engineer :: Game Designer : Game Developer") and compiler/PL tooling development.

5. Paper analysis

MLScent: A Tool for Anti-Pattern Detection in ML Projects

This paper by Shivashankar and Martini [4] introduces MLScent, an AST-based static analysis tool for detecting ML-specific code smells. It implements specialized detectors for multiple major ML frameworks including TensorFlow and PyTorch, and the data science libraries Pandas and NumPy, as well as general ML anti-pattern detection. The authors also perform both quantitative and qualitative studies that showcase the tool's effectiveness.

This paper is of importance to the engineering of AI systems because it highlights the most common machine learning antipatterns and code smells, and provides a tool that can automatically detect them, helping developers (and data scientists!) write more maintainable (and in some cases, even more *performant*) ML systems. Tying it to the notions presented during the guest lectures, this tool helps to "shift left" some potential ML software defects (i.e. move them earlier in the development lifecycle), thereby reducing the cost of fixing them.

I could see MLScent (and other similar tools) easily be integrated into any large project with a strong ML focus, for example as part of a CI/CD pipeline to automatically check pull requests for code smells, or even used as a pre-commit hook or integrated into IDEs to help developers lint their code as they are iterating on it.

The topic covered by this paper is tangentially related to my research, as it focuses on applying programming language-based techniques to improve software quality. If I were to steer my own research in this direction, I would look into designing a strongly-typed domain-specific language for ML models and components. This is because I feel that the biggest drawback of the current ML "stack" (which is primarily Python-based) is the lack of static typing. A high-level DSL with precise types could help ML developers express their intent more clearly and catch many defects (both general and ML-specific) at compile-time. There is some prior work in this area, for example the HaskTorch [5] library, which provides type-safe Haskell bindings to `libtorch` (the C++ library underlying PyTorch), making use of Haskell's advanced type system to track tensor shapes and dimensions at compile-time [6].

Privacy and Copyright Protection in Generative AI: A Lifecycle Perspective

This paper by Zhang et al. [7] highlights the issues and challenges related to data privacy and copyright infringement for Generative AI systems. It identifies and discusses important legal and privacy considerations, ranging from content licensing and data transparency to the acquisition and withdrawal of consent, centering on the legislative frameworks of the European Union's General Data Protection Regulation (GDPR) and the United States Copyright Law, and also discuses their potential consequences and side effects (for example, the emergence of fairness issues if certaina types of data are excluded for privacy reasons). The author focus primarily on the data

used for training GenAI systems and the outputs they produce, expanding upon the "data lifecycle" model introduced by Khan and Hanna [8] (which quite closely resembles the "machine-learning process" as described in the MLIP book [9]) and identifying which copyright and privacy challenges are relevant to each stage.

While this article doesn't introduce any technical solutions or tools, I find it highly relevant to the development of AI systems because it brings to attention salient legal and ethical aspects of machine learning, and also provides a comprehensive framework for analyzing and reasoning about these risks during the development (and deployment) lifecycle of ML projects. And while Generative AI is the primary focus of the article, most of the discussion is broadly applicable to ML systems in general.

These challenges and risks, and most importantly their consequences if ignored, directly increase with the scale of the ML system. For this reason, I find the ideas in this paper extremely valuable for large ML-enabled projects. The provided framework could serve as the basis for a set of guidelines and principles to inform the choices of high-level decision-makers on topics such as data sources, acquisition, and storage, and enable a more accurate assessment of legal and compliance risks. While not directly related to software engineering, these aspects are crucial for the success and longevity of large-scale AI-intensive projects.

The topic of this paper is only loosely related to my research, insofar as discusses privacy, which is interconnected with security. However, language-based security techniques such as information-flow control (IFC) and taint tracking could be applied to annotate and track copyright agreements and user consent as data flows through an ML system. An example of similar work is Meta's Policy Zones [10], which rely on dynamic information-flow control to enforce privacy requirements across Meta's systems. Tying this into the previous section, it would be worthwhile to add such IFC capabilities to the ML DSL I discussed, allowing privacy and copyright requirements to be tracked at the type-level to ensure that violations are caught by the compiler.

6. Reflection on research ethics & synthesis

Search and screening process

For the screening and selection of papers to analyze, I applied the following process: Looking through the list of accepted papers for the last 3 editions of

CAIN [11], I selected a number of papers based on their title, that were either related to my research or that otherwise seemed important or interesting. Then I further refined the list based on reading the abstract and a quick skim of the papers' content, cross-referencing with CAIN's researchr website [12] to ensure the papers were "Long" [1]. Finally, I narrowed it down to 2 papers based on which sparked most interest when considering the "how would I adapt my research to this" question.

Pitfalls and mitigation

I did not encounter any papers or abstracts that seemed misleading. A possible pitfall of my screening process might be that I over-indexed on the wrong (or less-relevant) aspects of the papers, due to my limited familiarity with the area of AI/ML.

Ethical considerations

No LLMs or Generative AI systems were used in the production of this work. This work is Brainmade [13].
To ensure originality, I did not consult other students' submissions or any resources on these topics other than those cited in the bibliography.

Bibliography

[1] G. Shen, S. Kashiwa, and L. Kuper, "HasChor: Functional Choreographic Programming for All (Functional Pearl)," *Proc. ACM Program. Lang.*, vol. 7, no. ICFP, Aug. 2023, doi: 10.1145/3607849

[2] "QuickCheck: Automatic testing of Haskell programs." Available: https://hackage.haskell.org/package/QuickCheck

[3] C. Kästner, "Machine Learning in Production: From Models to Products." Available: https://mlip-cmu.github.io/book/

[4] K. Shivashankar and A. Martini, "MLScent: A Tool for Anti-Pattern Detection in ML Projects," *2025 IEEE/ACM 4th International Conference on AI Engineering – Software Engineering for AI (CAIN)*. 2025. doi: 10.1109/CAIN66642.2025.00026

[1] Despite this, I chose a "Short" paper for my second analysis, because I felt that the subject it covers (copyright and privacy) is highly important and consequential to the development of ML systems.

[5] "HaskTorch: Functional Differentiable Programming in Haskell." Available: http://hasktorch.org

[6] "HaskTorch Tutorial - Typed Tensors." Available: https://hasktorch.github.io/tutorial/07-typed-tensors.html

[7] D. Zhang *et al.*, "Privacy and Copyright Protection in Generative AI: A Lifecycle Perspective," *2024 IEEE/ACM 3rd International Conference on AI Engineering – Software Engineering for AI (CAIN)*. 2024. doi: 10.1145/3644815.3644952

[8] M. Khan and A. Hanna, "The Subjects and Stages of AI Dataset Development: A Framework for Dataset Accountability." 2022. doi: 10.2139/ssrn.4217148

[9] C. Kästner, "From Models to Systems: Traditional Model Focus (Data Science) - Machine Learning in Production." Available: https://mlip-cmu.github.io/book/02-from-models-to-systems.html#traditional-model-focus-data-science

[10] L. Waye, W. Dong, B. Shulman, Y. Yan, and Y. Zhang, "Policy Zones: How Meta enforces purpose limitation at scale in batch processing systems." Available: https://engineering.fb.com/2025/07/23/security/policy-zones-meta-purpose-limitation-batch-processing-systems/

[11] "IEEE/ACM Workshop on AI Engineering - Software Engineering for AI (WAIN)." Available: https://ieeexplore.ieee.org/xpl/conhome/1842224/all-proceedings

[12] "CAIN conference series." Available: https://conf.researchr.org/series/cain

[13] "The Brainmade Mark." Available: https://brainmade.org/