Assignment 2: AI/AS ("AI Engineering") Essay

WASP Software Engineering Course (Module 2025)

Mathias Müller
KTH
matmul@kth.se

August 1, 2025

1 Introduction

I'm a researcher in the Department of Mathematics, in the Mathematical Statistics group at KTH in Stockholm. My research is primarily concerned with statistical inference in Bayesian models. In this area, we aim to derive the so-called posterior distribution of some underlying truth. Most of the time, this is done by using observations that give us information about the underlying variables. These could be parameters we want to estimate, latent states, or both. All of this requires significant compute, relying on intensive numerical methods and large-scale statistical simulation.

My specific topic at the moment deals with time-series data, where the data arrives sequentially. The difficulty here is that, because the data comes one at a time, we need to update the distribution at each step. Since the usual methods for deriving such approximations are very expensive, it is almost always infeasible to recalculate everything from scratch. Hence, there is a need for algorithms that can update the current approximation in a cheaper and more efficient way. One of the most important tools we work with is online variational inference. In these algorithms, one "guesses" a family of distributions with unknown parameters and optimizes them against a suitable metric (ELBO/RMSE). After receiving a new observation, we use ideas from Markov chain theory to update the distribution so that it gets closer to the true one. A key step in these algorithms is the proposal mechanism: we propose new samples from a distribution and, based on a comparison, either accept/reject them (as in MCMC) or reweight them (as in importance sampling/particle methods). In our research, we are exploring how to leverage neural networks for this proposal mechanism.

2 Lecture Principles

2.1 Behavioral Software Engineering

The key idea of behavioral software engineering is that it is people who both create and use software. The field is primarily concerned with the psychological and social aspects of software engineering as performed by individuals, small groups, or entire organizations. Since modern software engineering is incredibly complex and requires a large and diverse workforce, it is just as important to consider the behavioral and social aspects of these interactions as it is to focus on technical improvements. Simply pushing for better technology does not necessarily guarantee better outcomes. On the other hand, paying attention to the psychological and organizational aspects of how people collaborate on projects can often be more effective in improving results.

In academia, however, the situation is somewhat different. Much of the work is still carried out by individual researchers, sometimes with guidance from a supervisor, but rarely in large and coordinated teams. In my case, I spend a great deal of time coding, but my coding serves primarily as a proof of concept for theoretical developments in statistical inference and Bayesian methods. This means that many of the large-scale requirements of software engineering — such as maintainability across teams, strict code reviews, or long-term deployment pipelines—do not directly apply in my case. Instead, the relevant behavioral aspects often reduce to the psychological level of the individual researcher. For example, because I am usually the only one who has to read and maintain my code, I often end up writing messy implementations that prioritize speed of exploration over clarity or structure. This is a common trade off in research-driven software development, where the primary goal is advancing theory rather than building robust products.

However, the lecture introduced the concept of Collective Intelligence (CI), which I found especially interesting. CI suggests that certain tasks can be solved faster or more effectively in groups with high collective intelligence. This form of intelligence is only moderately correlated with the individual intelligence of the members, but strongly correlates with cognitive empathy and the ability to understand others' perspectives. Tests like Theory of Mind (ToM) correlate well with group CI. When I think about research in mathematics and statistics, it often feels overly individualistic: the culture expects each researcher to develop theory, write code, and publish more or less on their own. Yet, many of the complex challenges we face—such as developing efficient online learning algorithms for streaming data or ensuring reliable inference methods in high-dimensional settings—could arguably be advanced much faster in collective, interdisciplinary teams.

The hiring process in academia, particularly in mathematics, still focuses almost exclusively on individual excellence. Based on research in CI, one could argue that institutions should also pay attention to a candidate's ability to contribute to group-level intelligence. This perspective is rarely considered but could bring substantial improvements to collaborative research environments, especially in areas like mine where both theory and implementation are crucial and benefit from different skill sets.

2.2 Science vs Engineering and ML Engineering

The lecture on Science vs. Engineering and Machine Learning gave us a clear overview of the differences between science and engineering. Science is primarily concerned with understanding why something works and, based on that understanding, making predictions. Engineering, on the other hand, takes well-understood concepts and builds reliable tools to put them into practical use. Machine learning, however, is a relatively new field that does not fit neatly into either category.

On one hand, machine learning is widely applied to build end-to-end systems in the digital world. In this sense, it can be seen as an engineering tool, and ML systems must therefore meet the requirements of robust and reliable software. On the other hand, ML can also be used as a method for testing hypotheses, much like statistics or mathematics. In this role, it does not necessarily need to fulfill the strict requirements of an engineered system. Of course, most ML applications fall somewhere in between: a framework or model is used as part of a larger project, and so it must be integrated into an engineered system with its corresponding quality demands.

I would argue that there is also a third perspective on ML: research into the development of new ML methods themselves. This type of work falls even more into the scientific spectrum and has almost nothing in common with classical software engineering. This is the field where I do my research—developing new ML methods that rely on mathematical and statistical concepts to improve prediction and analysis. In my research, engineering requirements are almost irrelevant. It doesn't matter if the code is perfectly robust or production-ready. In fact, when data or assumptions change, the code often needs to be adjusted anyway. What does matter is reproducibility. In academic research, it is crucial to be transparent about how data and code are used, so that if another researcher tests the model, they should obtain the same results. Achieving this, however, is often arbitrarily complex, since many researchers make little effort to ensure their systems are easily understandable or executable.

3 Guest-Lecture Principles

3.1 Julian Frattini: Fundamentals of Requirements Engineering for Software (and AI) Systems

This lecture gave us a basic overview of requirements engineering, which is the systematic, iterative, and disciplined approach to developing an explicit requirements specification that all stakeholders agree upon.

A requirement generally consists of two parts:

- A need or constraint imposed by a stakeholder
- A capability or property that a system shall have

In an academic context, the PI or PhD student can often be seen as the internal stakeholder, since they are usually the ones defining the constraints, properties, and goals—such as which journals or conferences to target, etc. In other fields, particularly those involving costly experiments, external stakeholders may have a much larger influence. However, in mathematics, and in my own case, I am almost uniquely responsible for setting my own requirements and constraints.

At first glance, one might think this makes project development easier: with fewer stakeholders involved, there is less back-and-forth, and requirements can be defined in a small group, making development more closely aligned with those requirements. However, in my experience, this is rarely the case. Because so few people are involved, researchers often feel little need to carefully or explicitly define requirements—let alone write a requirements document. Worse, timelines are often not specified at all, leading to poor planning and inefficiencies.

The lecturer also emphasized the downside of neglecting requirements: the cost of removing defects increases significantly the later they are discovered. I think this lesson applies directly to research and academia as well, even though costs here are harder to measure since people work at different speeds and under different conditions. From my own experience, I can certainly confirm that poorly formulated requirements have led me to waste a great deal of time. Even more often, in academic projects one doesn't think about requirements or architecture at all, and instead jumps straight into implementation. This behavior frequently results in wasted effort later on, when things need to be restructured or redone.

4 Data Scientists versus Software Engineers

The first two chapters of Machine Learning in Production make it clear how hard it is to turn a research prototype into a production-ready system. Chapter 1 shows that, while ML powers impressive things—from voice assistants to medical diagnostics—most projects never make it past the lab. A reliable product needs far more than an accurate model: solid data pipelines, UX, security, operations, and, above all, real collaboration between data scientists and software engineers.

Chapter 2 continues by showing that the model is usually just one piece of a bigger system. The real challenge is stitching ML into non-ML components, handling failures gracefully, and aligning predictions with system-level goals like usability, fairness, and safety. From a software-engineering perspective, it is prudent to treat every machine-learning model as an unreliable function within the system—one that will inevitably return unexpected results from time to time. That way the focus shifts to designing infrastructure that can absorb such unpredictability.

What I found far less convincing was the claim that models can sometimes be "too good" and therefore not adopted. The example given was that if a travel model predicts too accurately, it might appear "creepy" to the user and hurt sales. Honestly, this explanation feels flimsy. There is no clear causal link established between accuracy and sales, and brushing it off with a "creepiness" argument comes across as hand-waving. Accuracy is simply a metric of how close predictions are to the truth. If the true outcome is what the customer wants and we predict something else and they don't buy it, then the accuracy is apparently not that high. If a more accurate model coincides with lower sales, the real issue is likely that users prefer cheaper offers, which may hurt margins. The misalignment is between what the customer wants (which is likely what the model was trained to optimize) and what is best for the company. I also found the "From Traditional Software to Machine Learning: Number of Systems vs. Risk" framing a bit arbitrary—why should traffic planning be considered less risky than chatbots? That ranking doesn't fully convince me.

The book closes the section by contrasting data scientists and software engineers. Data scientists excel at statistics, feature design, and exploratory modeling. Engineers focus on requirements, architecture, reliability, and long-term maintenance. In my view, this difference also explains why so much data-science code is of relatively low software quality. The core mission of a data scientist is not to produce clean, maintainable codebases but to explore data, run experiments, and extract insights, very much in the tradition of statisticians. Their "code" is really a vehicle for testing ideas, trying hypotheses, and building models, not for shipping robust production systems. Expecting it to look like production-ready engineering code misses the point. The real challenge (and opportunity) is building workflows and collaboration models where exploratory, experimental code from data science can be translated and hardened by engineers into something reliable and scalable.

I agree with the authors: both perspectives are necessary if you want ML products that last. In normal companies teams where ML is only a small part, for example a shopping website that recommends products, the pressure will be toward hybrid roles that bridge data science and software engineering because the model is not the product. On the other hand I expect more AI first companies where the model is the product. These will need very specialized, complex, and customized ML with separate research and engineering tracks. In practice the market will split: hybrid generalists when ML is just a feature, and deep specialists when AI is the product.

5 Paper Analysis (two CAIN papers)

Choose two full/long CAIN papers and for each cover the four prompts below.

5.1 Paper A: Addressing Quality Challenges in Deep Learning: The Role of MLOps and Domain Knowledge [3].

1. Core ideas and SE importance. Deep Learning systems often excel in specific tasks, however, engineering DL systems pose unique software-engineering challenges for quality attributes (QAs) like correctness and resource efficiency. So one should treat the whole DL system, not just the model, as an engineering object. This paper shows, in a chess-broadcasting system running on edge devices, that (a) basic MLOps (experiment tracking, energy/latency monitoring, automated data collection) provides traceability and fast feedback, and (b) wrapping the model with domain knowledge (legal chess moves, turn logic, special handling for captures/castling) dramatically improves system quality attributes.

2. Relation to my research. In my research the setting is different. We are not running algorithms live on an edge device, and since my work is concerned with streaming data, the training and test procedures are less clearly defined. One can view the algorithm as receiving live data points and incrementally learning a better approximation of a distribution (the "training" period). After running the algorithm, we evaluate the learned distribution on selected metrics (the "testing"). During operation, we also collect metrics and intermediate values and could, in principle, include a live feedback loop that improves performance. In this context, experiment tracking is especially helpful. I also agree that domain knowledge is an important complement; in my case this means using mathematical and statistical structure to guide and improve the distribution learning.

3. Integration into a larger AI-intensive project. The topics of MLOps and domain knowledge are broad and apply to many projects that involve ML on edge devices or, more generally, AI systems under resource constraints. A large software project where these tools should be used is medicine. One could imagine a device that conducts diagnostic tests locally in a doctor's office and returns results immediately. Here, latency and accuracy are critical, and embedding domain knowledge (anatomical rules, known disease patterns) helps prevent nonsensical predictions. My research would fit a context where the test is not a single momentary measurement but a sequence in which new information arrives over time and updates the final result. In such situations, online variational inference can be efficient. For example, a test could run for a few minutes, with predictions progressively approaching the true distribution, a simple feedback loop can monitor convergence and automatically return the result as soon as the estimate is good enough.

4. Adaptation to my research. Experiment tracking would be a beneficial improvement in my workflow. Concretely, I would collect and store not only the final results but also intermediate data during the run (e.g., convergence behavior, variances), and track design changes (e.g., via MLflow). Energy monitoring may be less important since I am not targeting live deployment, but convergence speed and memory usage should be first-class metrics. Creating domain aware algorithms is also central in my work: for example, choosing specific activation functions, parameterizations, and architectures for proposal neural networks that respect the constraints of the underlying posterior distribution and thereby stabilize and improve the inference.

5.2 Paper B: A Case Study on AI Engineering Practices: Developing an Autonomous Stock Trading System [2]

1. Core ideas and SE importance. This paper talks more generally about software that uses any ML components and how to ensure a certain quality of the developed system given their differences compared to traditional software. Specifically the paper is concerned with the development of an autonomous stock trading system that uses machine learning functionality to invest in stocks. They selected 10 AI engineering practices from the literature and systematically applied them during development, with the goal to collect evidence about their applicability and effectiveness. The ten proposed AI engineering practices and their effects are:

1. Standardize and automate data quality checks - (Caught subtle data issues early)
2. Use error validation and categorization - (Focused effort on the most harmful mistakes)
3. Capture the training objective in an interpretable metric - (Hard to operationalize)
4. Use cross-validation - (Removed overly optimistic results)
5. Continuously measure model quality, performance, and drift - (Kept the system within time budgets)
6. Review model training scripts - (Found real bugs)
7. Test all feature extraction code - (Uncovered subtle preprocessing errors)
8. Automate hyperparameter optimization and model selection - (Small but consistent gains)
9. Log prediction results with model version and input data - (Useful for debugging/auditing and ensembles)
10. Collaborate with multidisciplinary stakeholders - (reshaped objectives/constraints and improved the overall system direction)

Most applied practices improved the system, but to varying extent. By far the most important practice was the collaboration with multidisciplinary stakeholders.

2. Relation to your research. My work is research first (method development, not production), so I currently do not apply most of the practices. I don't run standardized data-quality tests beyond a few ad-hoc sanity checks. Cross-validation is occasional and not set up as a proper walk-forward protocol. There are no formal code reviews (single-author codebase). I rarely use automated model selection (manual tuning instead). I don't log predictions with versioned inputs and there's no multidisciplinary collaboration ritual. What I do keep is minimal: a lightweight note of seeds for key runs and a couple of sanity assertions in preprocessing. Given the exploratory nature and limited compute of my research, this trade-off is acceptable for now but I agree that some of the practices would be very helpful in the future.

3. Integration into a larger AI-intensive project. Since the ten practices are already good things to apply to every AI project the trading bot itself was only an example where to apply the uses. Generally speaking all of the above tricks should be used in every mid sized AI project no matter with what it is concerned.

4. Adaptation of your research. Going forward, I plan to adopt some of the ten practices to raise rigor without slowing exploration. I will set up a simple rolling CV protocol and start logging predictions together with the exact configuration, seed, and data version. I will introduce brief reviews of the scripts before trusting results, and schedule periodic check-ins with a software engineer to improve their quality—there is likely substantial room for improvement. In parallel, I'll add a few automated data/feature checks and minimal continuous measurement (e.g., ELBO and RMSE). This way I keep my workflow fast while making results better, reliable and easier to integrate into larger systems.

6 Research Ethics & Synthesis Reflection

6.1 Search and screening process

I just went to the CAIN conference website and scanned the names of all the papers in 2022 - 2025. Since my research is somehow related to finance, stock market etc. I found the first paper, A Case Study on AI Engineering Practices: Developing an Autonomous Stock Trading System, somehow interesting and relevant. The second paper, I chose based on the role of MLOps as I wanted to learn more about that. I found both papers portrayed the actual topic well in the abstract.

6.2 Ethical considerations

For the essay I mainly read the chapters of the book, the two papers and reread the slides of the lectures and wrote my thoughts about it in the corresponding chapters. I used an LLM in order to correct some grammar and language mistakes as english is not my native language.

References

[1] CMU, *Machine Learning in Production*, Ch. 1–2.

[2] M. T. Ribeiro, S. Martínez-Fernández, and X. Franch, "A Case Study on AI Engineering Practices: Developing an Autonomous Stock Trading System," *CAIN*, 2023.

[3] S. del Rey, A. Medina, X. Franch, and S. Martínez-Fernández, "Addressing Quality Challenges in Deep Learning: The Role of MLOps and Domain Knowledge," *CAIN*, 2025.