Software Engineering for AI

Willeke Martens

Department of Computing Science, Umeå University, wms@cs.umu.se

1 Introduction

My project harnesses theoretical computing science tools to constrain out-of-the-box large language models (LLMs).

1.1 Background

LLMs, a well-known example of generative AI, have received significant attention outside of the techno-science community with the recent successes of LLM-driven applications such as ChatGPT.

The user interacts with ChatGPT through prompts: a prompt is a sequence of letters. Before being fed into the LLM, the letter sequence is transformed into a sequence of larger textual units called tokens, which in turn are mapped onto binary vectors. For instance the prompt

> Life is an unending stream of extemporaneous problems.

corresponds to the token sequence

> [Life][_is][_an][_un][ending][_stream][_of][_ext][empor][aneous][_problems][.]

which is eventually fed to the LLM as the binary vector stream

> [26833, 374, 459, 653, 2518, 4365, 315, 1327, 53471, 18133, 5435, 13].

As a response, the LLM produces a probability distribution over all possible next binary vectors, such that higher-scored vectors intuitively correspond to the more likely continuations based on the training data. To select the next token, a separate sampling unit is consulted. Subsequently, the entire process is repeated until reaching the reserved end-of-sentence token.

1.2 When Statistics Are Not Enough

LLMs are powerful computational devices, yet they suffer from two limitations. First, LLMs require large amounts of compute and data. Being data-intensive, most out-of-the-box LLMs are trained on data scraped from the internet, meaning that the training data is of varying, unreliable quality. Second, each neural network architecture has inherent expressivity bounds, meaning LLMs cannot solve, let alone learn to solve every language problem.

Now suppose, an organization wants to automatically format user-provided input to comply with the JSON-format, which then later can be used by some downstream module. A first - rather costly -step could be to fine-tune an out-of-the-box LLM on custom designed corpora. However, given that the pipeline should be automated in its entirety, using even a fine-tuned LLM might not be sufficient, given that transformer-based LLMs for instance struggle with structural recursion and parenthesis matching.

1.3 Formal Methods To The Rescue

However, it is what constitutes a valid program in a particular programming language is often specified with the help of formal grammars. Instead of adapting the model, we can use other tools to guide the generation to be compliant with those grammars.

Fortunately, what constitutes valid JASON is completely specified by formal grammars. Instead of modifying the LLM itself, we can use external formal tools to guide the sampling process so that outputs comply with the specified format. My project develops mathematical frameworks to steer out-of-the-box LLMs toward producing outputs that are formally correct. One of the challenges lies in converting the formal requirements over grammar tokens to formal requirements over LLM tokens.

2 Lecture Principles

Being familiar with both philosophy of science and technology, I know that most theories trying to strictly delineate these practices, fall short. The lecturer tried to *distinguish scientists from engineers* among others by type of questions respective profession asked: "why does this work?" versus "how can we make it work reliably?". This is of course too simplistic since for example to ensure that something works reliably, we must at least have some idea as to why it works at all. In other words, scientists and engineers often ask overlapping questions and use similar methods. Even framing the distinction as a matter of discovery versus construction is fruitless, certainly in the context of computing science, which is fundamentally constructive in nature – yet distinct from among others software engineering.

The ambiguity made me consider the position of my project along the scale from basic research to R&D pipeline. Interestingly, my project is concerned with an important aspect of software engineering, namely *verification* (albeit at the moment only syntactical verification). Namely, given specific syntactic requirements, I aim at developing a method to ensure that the LLM complies with said requirements. Yet, the verification of my own proposed solution is not handled through static, dynamic or process evaluation techniques, but instead via proofs and theoretical analysis. So while the goal of project has a clear practical usefulness, it is not embedded in its entirety in the software engineering paradigm.

3 Guest Lecture Principles

Julian Frattini spent considerable time on distinguishing the *problem space* from the *solution space* in software engineering. In particular, he focused on how requirement engineering can aid with elicitation and refinement of the problem space in a more systematic and structured way. I believe these concepts to be equally useful in the context of research, especially in fields closely tied to practical domains. Having a clear grasp of the problem we are trying to solve, ensures that we are aiming our research at relevant solutions and are climbing the right hill.[1]

However, the boundary between the problem and solution space is not as sharp as indicated during the lecture. In fact, the delineation depends largely on the framing and the level of abstraction. For instance guided generation is often discussed in the context of program synthesis - the task of automatically generating computer code from a set of requirements, specified using mathematics, logics, or even natural language. Incorporating LLMs into the program synthesis flow, already situates us in the solution space, and is far from the only potential starting point. Yet, once settling on LLMs, the problem space shifts to for example making LLM-assisted program synthesis more reliable. Adapting formal tools with this goal in mind is just one of the possible solutions, fine-tuning models another. Additional requirements beyond reliability, such as limited computational resources and data availability, can further constrain the solution space, e.g., eliminating fine-tuning as an interesting avenue.

The question of whether we are actually climbing the right hill, or *solving the right problem using appropriate means*, is then again a matter of framing and the level of abstraction. In the context of LLM-assisted program synthesis, I believe guided generation is the right paradigm – but I am far less convinced that we are solving the right problem: at a higher level of abstraction the real issue could be insufficient developer capacity. In that case, improving hiring practices or employee retention might be a more effective solution than pursuing program synthesis at all.

4 Data Scientists versus Software Engineers

That data scientists and software engineers possess different knowledge and skills, and have distinct immediate interests and concerns, is straightforward. It reminds me of the distinction between mathematicians and scientific computing engineers: mathematicians may know which derivation to compute and how to compute it accurately on paper, but implementing the same method on a computer can lead to catastrophic errors. Both the mathematician and the scientific computing engineer are essential and may even address the same problem, yet they operate at different levels of abstraction and carry distinct responsibilities.

Rather than merging the roles of data scientist and software engineer, I plead for even stronger interdisciplinarity. We should not only bridge communication between these two groups, but also include other disciplines such as interaction design and ethics. The idea

[1] Reference to the iconic paper by Bender and Koller (2020).

that a single short course - for example, the WASP ethics course - could turn participants into experts in ethical reasoning is laughable (a point underscored by some participants openly questioning its relevance during the lectures). Each of these professions requires years of dedicated study, and their expertise cannot simply be collapsed into a hybrid role. A more realistic and productive approach – at least to me - is the T-shaped team model mentioned in the assigned literature where individuals have deep expertise in one domain, but also have cultivated enough breadth to communicate across disciplines. The end goal is that all members can work effectively together toward a common goal.

5 Paper Analysis

5.1 Generating and Verifying Synthetic Datasets with Requirements Engineering, Vonderhaar et al. (2025)

That more is better, is a common assumption within the deep learning community. Seeking to improve your machine learning model? Train it on more (preferably higher-quality) data and for longer. Ignoring for a second the required additional storage and compute, obtaining the required data often is already a prohibitive expensive step – often outsourced to companies with murky human labor practices. However, today, with the successes of LLMs for text generation and diffusion models for visual representations, synthetic dataset generation and augmentation has gained significant traction. With the practice becoming more popular and wide-spread, systematic processes for dataset quality assurance have to be developed. For traditional dataset manufacturing, there are several known metrics and routines in place, yet in the automatic counterpart, quality assurance often remains ad-hoc. This paper is inspired by the software engineering validation-verification cycle to move towards a more transparent and rigorous framework.

The authors use requirement engineering to both steer data generation and subsequently verify the results. The dataset is treated as a piece of software itself such that the requirements are dictated by the requirements on the downstream model (in training). The authors showcase this in the context of developing an object detector. First, we need to specify the requirements for the object detector. One such requirement could be that "the object detector shall detect a Toaster class when the class is close up in the image." This requirement can then be translated into a prompt for the generating model ("a photo of a toaster close up") and traceable verification test cases (whether the artifact actually contains a toaster in close up).

Interestingly, the authors propose dual verification, both with the help of a model simulating the trained downstream model, and a human in the loop. The assumption is that if the simulating model manages to locate a toaster in the image, the downstream model will be able to do so as well. The human is tasked with verifying that the toaster actually is shown in close-up. Only the images passing both the automatic and manual verification are supposed to be included in the final dataset.

The requirement-engineering aspects indeed results in a transparent, traceable verification process. However, upon reading, I had several questions. On the one hand, there

is still the heavy reliance on humans in the loop – but this is probably unavoidable. Human labor savings occur during the gathering data phase, not the data verification phase. On the other hand, more problematically, is the role of the simulating downstream model. The authors for instance mentioned that the simulating model had a problem recognizing toasters and regularly mistook them for mobile phones, boxes or refrigerators, while the human in the loop okay-ed said images. As a result these images were discarded, making it questionable whether the real model ever would be able to surpass the simulant. And lastly, the engineering of the requirement specification was not really expanded upon, while this must be one of the most vital aspects for ensuring a high-quality dataset. Obviously the requirement that "the object detector shall detect a Toaster class when the class is in the image" would not be sufficient to ensure the required diversity. Yet, it is left in the open how we actually can make sure the requirements are sufficiently but not overly specific.

The verification process proposed by the authors works reasonably well for single-step generation tasks (ignoring the questions above), such as image synthesis, where each sample can be checked after the fact. However, this strategy is much less effective when dealing with sequential generation, such as producing code snippets for training an LLM-based code completion system. In such cases, the model must generate a complete snippet before verification is even possible. If the output fails to meet requirements, there is no guarantee that repeated attempts will succeed, leading to wasted computation. My own research addresses this gap: by using formal tools to constrain generation online, I can ensure that intermediate outputs - such as partial code snippets - are at least syntactically correct. While additional semantic checks would still be necessary afterwards, this approach reduces inefficiencies and shows why offline verification alone is insufficient for sequential generation tasks. In a sense, my research acts complementary to the proposed loop.

5.2 ML-On-Rails: Safeguarding Machine Learning Models in Software Systems – A Case Study, Abdelkader et al., (2024)

In traditional software engineering, established methodologies and processes are used to ensure that an application including its different components are robust. The authors however highlight that while integration of machine-learning components into applications has steadily increased, those components are inherently data-driven and probabilistic, and therefore significantly different: traditional routines are not sufficient nor appropriate. Instead a new framework is required to address the specific risks and concerns relevant to those machine-learning components. The authors introduce ML-On-Rails, a framework that not only indicates the essential aspects the model-provider must consider and implement, but also provides more clear-cut guidelines surrounding the responsibility of the model-provider versus the model-user.

The framework focuses on different aspects such as out-of-distribution detection, and model explainability, all necessary to safeguard the machine-learning component. Whilst the raw model tends to fail silently, the component should not. This should enable the model-users to perform error diagnostics and error handling. To offload these detection

services on the model-provider is intuitive, since most of these services require significant insight into the training data and underlying model architecture – details providers are often unwilling (or unable) to share. The model-user still might be required to add application-specific safeguards, which are not necessarily tied to the model itself.

As a case study, the authors apply their framework to the MoveReminder application, a mobile health tool with two ML components: an activity recognition model and an LLM. The activity recognition model processes sensor data to predict the user's activity, while the LLM combines these predictions with contextual factors such as weather and environment to generate personalized activity recommendations. The framework ensures that if, for example, a sensor malfunctions, the activity recognition model can signal to the application that its output is unreliable since the input is OOD. Instead of passing faulty data to the LLM and producing misleading recommendations, the system can then warn the user to for instance check their hardware.

From the perspective of my own research, ML-On-Rails makes a useful distinction between provider safeguards and application safeguards. My work fits more clearly within the latter: rather than expecting the model provider to guarantee syntactic compliance with formal requirements, I develop tools that allow application designers to constrain model output.

6 Research Ethics and Synthesis Reflection

6.1 How I conducted the Search

I know I did not exactly do as instructed, but I adapted the assignment to still deepen the material in the course, yet make it relevant and engaging for my own project and interests.

At first, I attempted a more systematic search using keywords related to guided generation, but this did not yield any immediate results – as it probably was too narrowly scoped. Instead, I started skimming through each edition in the hope of finding articles that sounded interesting and, with some imagination, could be relevant to the area of my field of work. Since I started with the most recent edition, article 1 was among the first I pre-selected for further consideration. Since it immediately triggered thoughts, I simply kept on reading. My choice of article 2 is more ad-hoc: I started reading several, but abandoned them since they were more technical in nature (and not really conversing on the nature of software engineering for ML) – or kept on drifting further away from the already tangential connection to my research. While article 2 is not closely connected to my topic, it tied in neatly with the software–ML discussions in class.

6.2 Ethical Considerations

To avoid copying from LLMs, I simply did not use them. (I know – magic.) Since I am familiar with paraphrasing, analysing and thinking, I also avoided the trap of copying from the papers.

Bibliography

Emily M. Bender and Alexander Koller. 2020. Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 5185–5198, Online. Association for Computational Linguistics.