Editor in Chief: **Ipek Ozkaya**
Carnegie Mellon Software Engineering Institute,
ipek.ozkaya@computer.org

What Is Really Different in Engineering AI-Enabled Systems?

Ipek Ozkaya

ADVANCES IN MACHINE learning (ML) algorithms and increasing availability of computational power have resulted in huge investments in systems that aspire to exploit artificial intelligence (AI), in particular ML. AI-enabled systems, software-reliant systems that include data and components that implement algorithms mimicking learning and problem solving, have inherently different characteristics than software systems alone.[1] However, the development and sustainment of such systems also have many parallels with building, deploying, and sustaining software systems. A common observation is that although software systems are deterministic and you can build and test to a specification, AI-enabled systems, in particular those that include ML components, are generally probabilistic. Systems with ML components can have a high margin of error due to the uncertainty that often follows predictive algorithms. The margin of error can be related to the inability to predict the result in advance or the same result cannot be reproduced. This characteristic makes AI-enabled systems hard to test and verify.[2] Consequently, it is easy to assume that what we know about designing and reasoning about software systems does not immediately apply in AI engineering. AI-enabled systems are software systems. The sneaky part about engineering AI systems is they are "just like" conventional software systems we can design and reason about until they're not.

I argue that our existing design techniques will not only help us make progress in understanding how to design, deploy, and sustain the structure and behavior of AI-enabled systems, but they are also essential starting points. I suggest that what is different in AI engineering is, in essence, the quality attributes for which we need to design and analyze, not necessarily the design and engineering techniques we rely on. In some respects, the junction we are at is no different than when we realized security had to be treated as a primary quality concern in software systems; when we remembered that if we do not design for the users and architect for usability, systems fail; or when privacy concerns started to dominate our discussions.

Today, security, usability, and privacy are among the many other mainstream architectural concerns; we have common vocabulary and analysis methods to design and check for such attributes. Similar progress needs to be made in regard to explainability, data centricity, verifiability, and change propagation at a minimum because these attributes are critical in successfully designing the structure and behavior of AI-enabled systems. There is work to be done, but we are not starting from scratch.

Process of Building AI-Enabled Systems Is Different

The process of building AI-enabled systems does, in fact, differ from the process of building software systems. Industry teams recognized quickly that the role of the data scientist is a critical addition to the software engineering teams.[3] The development of an ML model requires the data scientist to engage in exploratory analysis and can require several iterations before an appropriate model is produced.[4]

Digital Object Identifier 10.1109/MS.2020.2993662
Date of current version: 18 June 2020

CONTACT US

AUTHORS

For detailed information on submitting articles, visit the "Write for Us" section at www.computer.org/software

LETTERS TO THE EDITOR

Send letters to software@computer.org

ON THE WEB

www.computer.org/software

SUBSCRIBE

www.computer.org/subscribe

SUBSCRIPTION CHANGE OF ADDRESS

address.change@ieee.org
(please specify *IEEE Software*.)

MEMBERSHIP CHANGE OF ADDRESS

member.services@ieee.org

MISSING OR DAMAGED COPIES

help@computer.org

REPRINT PERMISSION

IEEE utilizes Rightslink for permissions requests. For more information, visit www.ieee.org/publications/rights/rights-link.html

Once a model is developed a continuous sustainment and evolution cycle is not only necessary; it is inevitable for the viability of the system.

The tasks required to manage AI systems that include ML models are not always aligned with software development tasks. Although continuous evolution and iterative development are not new to software engineering, the uncertainty introduced by the volatility of the data and ML model predictions is certainly not common. An early lesson learned is that a number of mismatched assumptions can lead to significant failure of the AI-enabled system.[5] Such failure is often a consequence of disjointed development and sustainment processes among the data scientists, software engineers, and operations staff.

Most of the process differences between developing AI-enabled systems and software systems are driven by the central role played by data in AI systems. Data collection, cleansing, management, and continuous updates add additional tasks. Resource and execution planning of AI-enabled systems, therefore, should take into account these tasks. These tasks also drive additional roles and responsibilities as well as additional process steps that need to be considered during resource and execution planning. Those who collect, curate, and manage data may or may not be part of the responsibilities of the data scientists or software engineers. Data science is forging its own methods for addressing the data centricity of ML development, for example the Team Data Science Process by Microsoft, but their alignment with software engineering processes are yet to be put to test.

Software engineers are familiar with working with large volumes of data as well as all the compliance concerns. However, in AI-enabled systems, data have multiple facets: data are both an enabler and a constraint. Resources required for data management are often not estimated correctly. System elements that interact with data in any way (including create, read, update, delete, and store) become key system components in the deployed system. Moreover, data need to be updated continuously, which will impact several architectural concerns for the system, including its latency and prediction accuracy.

Data scientists and software engineers, respectively, understand the process steps of data science and software engineering; however, we have yet to define what an integrated, well-managed, AI-enabled system development process looks like. In particular, how data and model evolution impact the overall systems sustainability is an area where we need to make significant progress. Organizations that have demonstrated success in applying AI techniques to their data-driven problems automate anything and everything they can and rely on robust infrastructures to allow for managing multiple versions of deployed software, reverting back as needed.

These techniques call for a level of scale and resource management that many organizations are not equipped with. Robust infrastructures and automation frameworks are essential for the successful development of AI-enabled systems. The availability of infrastructure, computing resources, and tools should be complemented by development methods and processes that clearly define the new roles and tasks that are fit for AI engineering.

Our Misconceptions Exasperated by AI Engineering

The differences in the process of engineering AI-enabled systems do not

necessarily translate to contrasts in analyzing and designing for such systems. Deploying and sustaining successful software-reliant systems require constantly managing competing business and architecture tradeoffs. These tradeoffs dictate how systems are structured as well as how they behave and, consequently, can effectively be evolved and sustained.

There are a number of misconceptions we hold for software systems that do not include AI components. Before we can make progress in improving how we design and analyze for AI-enabled systems, we need to revisit some of the current misconceptions we have about engineering software-reliant systems.

- *We can specify systems*: A significant difference listed for AI-enabled systems is that they cannot be specified up front, whereas software systems can be. There are systems whose requirements we can and do know up front or can easily discover iteratively. Those tend to be manageably sized systems that we have learned how to develop over the years. In reality, many software systems, even without AI components, struggle with uncertainties and "unknown unknowns" throughout their development. What changes in engineering AI-enabled systems, however, is the fact that uncertainty is their dominant characteristic. In particular, learning from data and the discovery process introduced with ML-modeling activities both introduce many uncertainties. AI specifications are specifications of problems, not systems.
- *System correctness can be verified*: If we cannot specify systems, challenges in verifying them, of course, are not surprising. There are countless examples of failure due to mismatched specifications, especially between as-designed and as-deployed software systems. Verifying at requirements or design time against predefined requirements does not suffice to safeguard systems from runtime failure. Software engineers have long sought the quest of developing systems that are correct by design. Model-based software engineering has made progress but not to the extent that systems can be modeled end to end, driving improved formalism and verification. Verification challenges are inevitably exacerbated in AI-enabled systems given their inherent uncertainty.
- *We can avoid hidden dependencies*: The hard-to-trace dependencies, in particular those induced by data dependencies, become a significant source of failure in ML systems. These hidden dependencies make applicability of the known architectural patterns to manage system evolution and separation of concerns challenging. Introduced by Google engineers as the Changing Anything Changes Everything (CACE) principle, systems with ML components in particular not only become highly coupled but also more complex.[6] However, the reality is that hidden dependencies exist in all systems.[7] Hidden dependencies have always been a challenge to manage, in particular runtime dependencies, because software engineers lack tools to analyze, model, and visualize these dependencies. Data dependencies that are inherent in AI-enabled systems suggest that we need the tools we already lacked even sooner.

Digital Object Identifier 10.1109/MS.2020.2972661

> We are still climbing the hype curve in our expectations of what AI can accomplish.

- *We can manage system change propagation*: The implications of the CACE principle are that, in AI-enabled systems, there is a dire need for improving how change propagation is managed, both to reduce the uncertainty of the expected results and to improve the engineer's ability to debug systems. Software engineering tools and techniques still need to make progress to improve analyzing and safeguarding for change propagation. The inability to manage change propagation leads to technical debt, which is also recognized as a challenge in AI-enabled systems.[7]
- *Frameworks do it all*: Tools that help engineers analyze data, experiment with learning algorithms, and create initial versions of ML models are quickly becoming a commodity. The ability to generate an initial model falsely gives the impression of having a successful AI application prototype. The challenge starts when these quick and dirty models need to be integrated into deployed, functioning systems. Existing frameworks, model libraries, and toolsets do not replace a well-architected system that can scale, minimize the impact of change propagation, and address the AI-enabled system and relevant quality attribute concerns such as explainability, data centricity, and verifiability.
- *We can build reliable systems from unreliable and unpredictable subcomponents*: We see the challenges of analyzing and designing for reliable systems in embedded real-time systems that rely on integration of many disparate software and hardware components, some of which may be developed and owned by different parties. The mismatched assumptions propagate to all levels of design. AI components developed independently are yet another set of subcomponents whose behavior needs to be reliably predicted.

We are still climbing the hype curve in our expectations of what AI can accomplish. Objectively analyzing what is similar and different in engineering such systems will guide how and where research can focus on improving designing, analyzing, and verifying AI-enabled systems. Progress is needed in defining new development process models for AI engineering. Meanwhile, we need to appreciate that the uncertainty introduced by the behavior of AI systems is nothing new to software systems—we already did not know how to manage uncertainty. The rush to deploy AI-enabled systems just increased the urgency of making progress on how to model, analyze, and safeguard against the inherent uncertainty of our systems.

Acknowledgments

The ideas I present in this article are influenced by conversations with Mary Shaw and many colleagues at the Software Engineering Institute.

References

1. A. Horneman, A. Mellinger, and I. Ozkaya, *AI Engineering: 11 Foundational Practices*. Pittsburgh: Carnegie Mellon University Software Engineering Institute, 2019. [Online]. Available: https://resources.sei.cmu.edu/asset_files/WhitePaper/2019_019_001_634648.pdf
2. J. Bosch, I. Crnkovic, and H. H. Olsson, Engineering AI systems: A research agenda. 2020. [Online]. Available: https://arxiv.org/abs/2001.07522.
3. M. Kim, T. Zimmermann, R. DeLine, and A. Begel, "The emerging role of data scientists on software development teams," in *Proc. ICSE 2016*, pp. 96–107. doi: 10.1145/2884781.2884783
4. S. Amershi et al., "Software engineering for machine learning: A case study," in *Proc. ICSE (SEIP)*, 2019, pp. 291–300.
5. G. A. Lewis, S. Bellomo, and A. Galyardt, "Component mismatches are a critical bottleneck to fielding AI-enabled systems in the public sector," presented at AAAI FSS-19: Artificial Intelligence in Government and Public Sector, Arlington, VA, 2019. [Online] Available: https://arxiv.org/abs/1910.06136v1
6. D. Sculley et al., "Hidden technical debt in machine learning systems," in *Proc. 28th Int. Conf. Neural Information Processing Systems (NIPS'15)*, vol. 2, Dec. 2015, pp. 2503–2511.
7. R L. Nord, I Ozkaya, R S. Sangwan, R J. Koontz, "Architectural dependency analysis to understand rework costs for safety-critical systems," in *Proc. 36th Int. Conf. Software Engineering (ICSE Companion 2014)*, May 2014 pp. 185194. doi: 10.1145/2591062.2591185.