JuliaHub

Dyad

WHITE PAPER

SOFTWARE DEFINED MACHINES WITH DYAD

AI & Digital Twins drive rapid iteration in industrial engineering

info@juliahub.com

www.juliahub.com

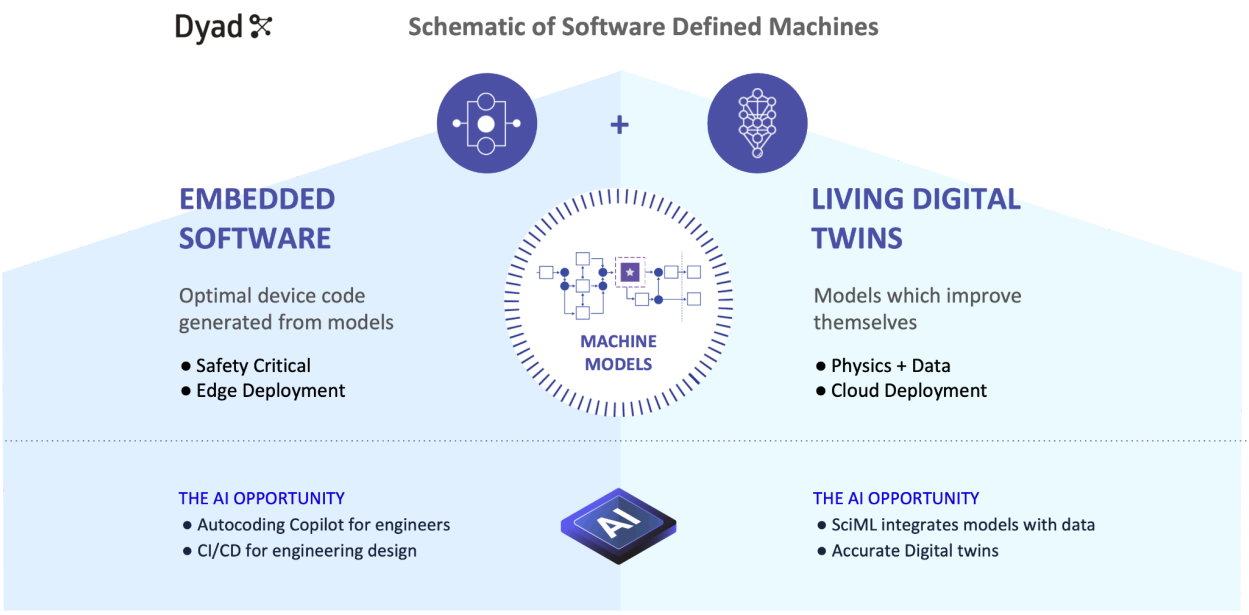
Table of Contents

1. Industrial Engineering is Transitioning to a Software-Defined Environment

Computing is becoming more powerful and pervasive, and is reshaping every product, from toothbrushes to rockets. Every physical object has a digital equivalent - a twin, used as a testing ground for new designs,rapid deployment, and new experiences. Every car, airplane, turbine, power plant, data center, and more, is being investigated – all the way down to the ball bearings – in order to improve efficiency and reliability.

We are entering the world of Software Defined Machines - where *software* is used to design every piece of hardware, runs on the hardware itself, and, once the hardware is deployed, functions as its digital twin, learning from real-world feedback. Software Defined Machines use models as the single source of truth across the product lifecycle. Software Defined Machines will make engineering design as fast as software development.

New features are being added to automobiles through over-the-air software updates, while sensors cover every airplane to continuously recalibrate predictive maintenance programs. Products can achieve higher satisfaction and greater consumer lifetimes while reducing the cost of materials and solutions, simply by adjusting software to the changing environment. For example, the Rivian driver experience was greatly upgraded recently, not through a recall, but instead by a software update to the controllers in the suspension [3], which used data gathered from sensors to refine the parameters that were previously calibrated using lab data. One can imagine pushing this world even further, where models are re-trained based on the data for each specific vehicle in order to automatically improve the experience, based on that vehicle's wear and tear, the inclinations of a specific driver, and the typical road conditions its driver faces. This revolution, which is digitalizing the physical world into software-defined machines, has a major potential for savings in manufacturing costs – all while achieving higher quality products, especially when considering advances in AI and machine learning that enable integration of data into digital twins. With the right software, even cheap materials can give a more luxurious experience if they automatically tune and adapt to the environment.



Schematic of software defined machines. Software defined machines are the merger of embedded software with digital twins. At their core are machine models, which are system level models of the physical interactions, the electrical systems, and the control systems involved in the real-world device. The software defined machine extends the model using the data about the system to refine the model to higher fidelity: the precision of the model begins to capture features only seen outside the lab such as defects in the manufacturing process and the wear-and-tear of individual vehicles. The controllers on the vehicle, due to being software-defined via embedded software, can then be semi-automously updated based on the refined physical knowledge of the system, effectively molding the hardware to the improved physical understanding learned through the AI.

A true marriage between the learnable behaviors and the constraints of modern engineering needs copious amounts of care and attention to detail in order to not derail the progress that has been made over the last century. In particular, standard engineering practices are heavily focused around aspects such as robustness, safety, and validation. Engineers are excited about the promise of AI tools, but also concerned about the pitfalls. For example, is replacing a hand-tuned, safe, and inspected automatic braking system with an AI-powered image sensor, even though it includes new sensor modalities which may improve autonomous operation? Given these sorts of challenges, the major question that must be posed to the field is: what is the right way for AI and SciML advancements to be integrated into the engineering models and workflows in order to best accentuate the advantages while mitigating the risks?

To address this challenge and transform the field, we must accept that industrial engineers are extremely conservative. And this is rightfully so, as the safety and well-being of millions of people can hinge on their decisions. However, we must also understand that advancements in AI have provided an opportunity for new tools to leapfrog old workflows. AI tools will provide Copilots for Engineers helping them build product twins quickly, accelerate running times with surrogates, use Scientific AI techniques to collect data from the field and refine models and autocomplete missing physics, translate models from legacy systems to new representations, and make natural

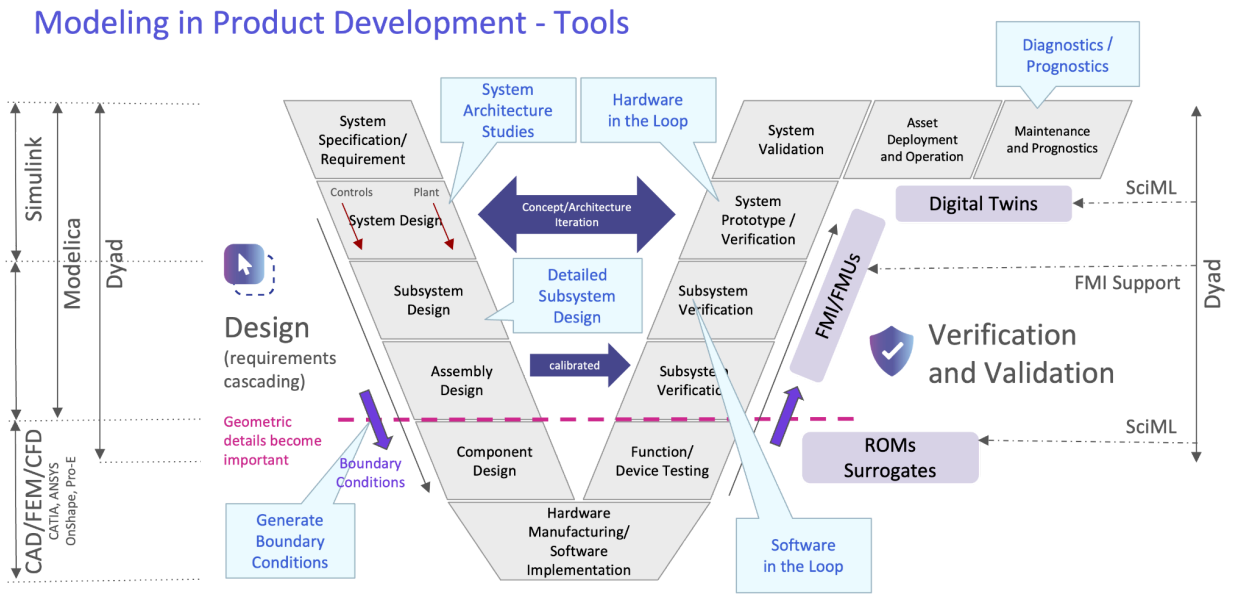
language a first class part of the design and deployment experience. Software Defined Machines will lead to disruption in the $30B Modeling and Simulation industry [18] and the $73B Digital Twin opportunity [17]. Software Defined Machines are the merger of AI with modern engineering software, done correctly and safely.

2. Issues with Current Engineering Software

2.1 Legacy engineering tools do not facilitate modern agile workflows

Modern engineering practice is dominated by workflows that revolve around physical modeling and designing control systems. Some tools, like computational fluid dynamics (CFD) tools and detailed 3D spatial digital twins of elements such as battery cells, are meticulously developed around specific physical processes and focus the engineer on key physical processes which can be understood and better exploited for efficiency gains. While these tools occupy a crucial space in the engineer's toolkit, the expense (both human and computational) of isolating a component to achieve such high fidelity is too high to build a complete picture of the entire system. When trying to understand the system-level dynamics, how different optimized components will integrate and understanding for example the predicted performance of an entire vehicle in tandem, engineers resort to system-level simulation tools where the models strike a balance between simplified physics but high enough fidelity to make constructive decisions about complete systems.

What this means is that the domain of systems level modeling is heavily dependent on the engineer to sit down and make choices about the right way to build the model, what features need to be captured and what is not needed, and what elements of the physics to include while deciding which elements it is safe to assume is not important. This is a manual and iterative process that can take years. It is not an exaggeration to suggest that the majority of the time engineers use the modeling tool is spent trying to understand whether the model sufficiently matches the real world.

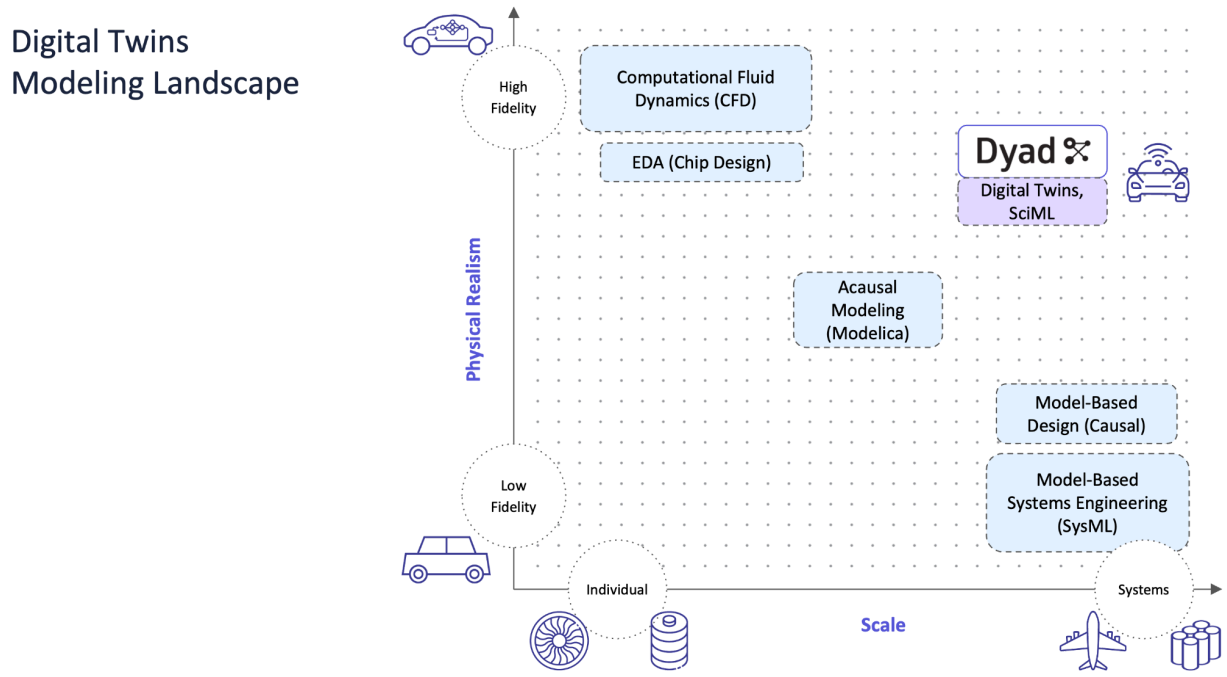


The Traditional V-diagram of modeling and simulation in product development. Above the dashed red line are the system modeling tools and the role they play in system design in verification. Below the dashed line are the high-fidelity 3D digital twins such as CFD software. Systems modeling tools must integrate with such tools but ultimately must simplify to capture the complexity of the entire system, though today much of this integration is manual.

To top it all off, the systems modeling tools which are in play have not fully kept up with the major changes to compiler technology which accelerated through the 2010's. We have seen a boom in agile development platforms, continuous integration testing and deployment (CI/CD), and the adoption of Git-based version control greatly accelerate the pace of software, but these workflows are not integrated into the core of traditional engineering platforms. Additionally, major advancements in underlying tooling include the LLVM compiler for high-performance just-in-time compilation which can target many platforms, the explosion of new tooling around automatic differentiation (AD) and machine learning requires entirely new solvers and compilers in order to achieve full integration. This means that recent advancements in domains such as physics-informed machine learning (PIML) and scientific machine learning (SciML) have seen major growth as potentially new workflows for modelers to build better models faster, but there are some serious technical hurdles for fully retrofitting into the existing technologies. And similarly, the stumbles of Microsoft Office vs other tools such as Google Docs shows the difficulty of migrating traditionally desktop-based software into a fully-collaborative cloud-based environment where the source of truth is always shared and handled by multiple users simultaneously. This shows that the next generation of engineering software needs to start from an entirely new foundation.

2.2 The traditional divide between models at different scales does not generalize to digital twins

Tools developed for system-level simulation have traditionally been separated from those of high-fidelity spatial 3D modeling. We tend to put the types of problems for which a systems modeling tools are used in a different bucket from those that would use tools like Computational Fluid Dynamics (CFD) or Computer-Aided Design (CAD). However, with the rising construction of digital twins, there is an increasing push to achieve higher and higher fidelity within the systems-level models in a way that is equivalent to embedding the high fidelity models within the systems model and its control circuits. This is compounded by the fact that traditional control systems were greatly limited by the capacity of embedded controllers, but with the ever increasing improvements in compute power and efficiency, modern controllers can often make use of ARM chips which are capable of running an entire smartphone. This means that even when targeting real-time embedded applications, higher fidelity model-based control or multi-frequency control where a lower-frequency higher-fidelity prediction is used is becoming more normalized.



Schema of the modeling landscape with respect to software-defined machines. In the top left there are the high-fidelity modeling systems of single assets, such as computational fluid dynamics which models every detail of airflow over an airplane wing and EDA tools which are a full specification of chips. In the bottom right you have tools which model the entire system but to low fidelity. For example, SysML uses natural language requirements specifications of hardware systems, and model-based design (causal modeling tools for embedded control systems) adds mathematical descriptions of control systems. In the middle you have acausal modeling tools which blend some of the accuracy of the component modeling tools while achieving a higher level system description, but require making trade-offs on both fronts. This highlights the advantage of the Dyad digital twin approach, which uses SciML in order to elevate the realism of system level models to almost achieve that of the individual component modeling tools, while being able to represent the entire system and the artifacts for its software-defined embedded control systems.

However, the compiler infrastructure of the systems level modeling tools have not kept pace with these requirements. Existing tools are well known for limitations in both memory and compute as the size of the systems grow, but even systems known for efficiency can have scalability issues when dealing with thousands of states [9]. Connections with these model types are therefore black-boxed. This means that when integrating higher-fidelity sources of truth—such as CFD tools and other domain-specific digital twins—the system uses black-box formulations through protocols like the Functional Mock-up Interface (FMI). FMI embeds the complete simulation code as a discrete block within the modeling system. As such, these models use different solvers, time steppers, etc. which are disconnected from the rest of the model, and then stepped in a lock-step pattern known as cosimulation which leads to many artifacts in the numerical stability, performance, and accuracy [16]. The inability for the modeling compilers to fully optimize the simulators across boundaries thus limits the ability for this model combination to be fully scalable.

2.3 General AI Tooling is not sufficiently trustworthy for Safety-Critical Engineering

While some Silicon Valley AI startups would lead you to believe that machine learning will replace all other forms of computation in the next 5 years, the majority of mechanical, aerospace, and automotive engineers are rightfully skeptical that there will be a complete replacement in these domains anytime soon. One major reason for this is, as emphasized in the previous section, the process of understanding what is necessary in a model is an iterative process that is refined. It is not entirely captured in the computer: it's a process of building a model, checking the real system, finding disconnects, making decisions about what is okay to be kept and not, understanding what new sensors could be helpful to further refine the model, and repeat.

Instead, machine learned solutions are black-boxes that are hard to understand and modify. While one can receive new data to retrain them, due to phenomena such as local minima it can be hard (if not impossible) to have any guarantee whether new iterations of a model have gotten closer to this global idea of the true system which can be difficult to capture in data and loss functions. Standard machine learned models do not have a sense of physical truth, and thus we do not have a guarantee that their predictions match physical principles such as conservation of energy and momentum, meaning that predictions can drift away from reality over time and the boundary to which they are not trustworthy is ill-defined.

With all these points together, it's clear that the job of the modeler is very unlikely to be replaced wholesale by a purely machine learned process, especially in regulated domains for which these model building aspects are closely checked in order to achieve safety in consumer systems such as automotive and aerospace. That said, the future can certainly have engineers in loop, taking

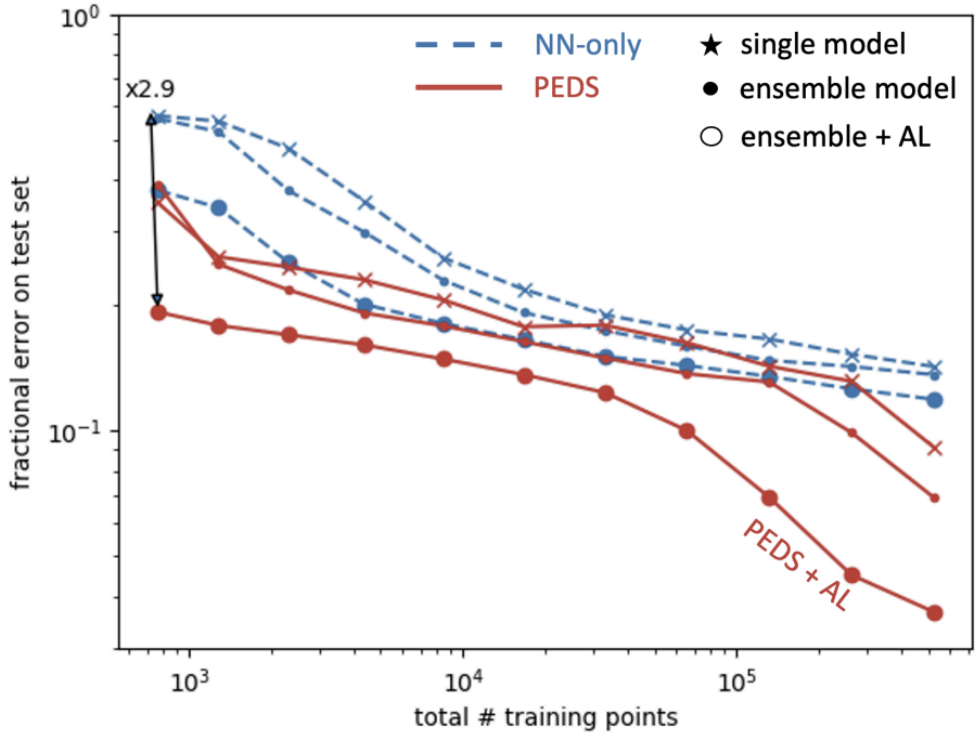
tools such as Large Language Models (LLMs) and AI chatbots to accelerate the usage of system modeling. However, such integration must be done with care because these tools are known to have difficulties with accuracies, known as hallucinations, and it's well-known that just one small error in a model can make the predictions completely incorrect, and thus unlike an essay a model has almost zero tolerance for such errors. Therefore any integration needs to be carefully thought through in order to highlight the areas which the modeler should second guess for the inevitable debugging phase. In addition, systems modeling has very specific sources and is thus not likely to be part of the core corpus of training data in tools such as ChatGPT or Google's Gemini: this domain requires specific API integrations to construct domain-specific word embeddings for the foundation models to understand system modeling will be necessary to have any level of accuracy. As such, a successful solution likely would need to integrate agentic AI tooling, which would need deep integration into the system modeling tool and change some of the standard workflows.

2.4 Scientific AI Has Not Transitioned to Industry

While standard machine learning tooling does not look like a viable alternative to traditional systems-level modeling and simulation and has thus not made any inroads into the industry over the last two decades of major ML advances, a subdomain known as scientific machine learning (SciML) shows promise by overcoming many of the previously mentioned shortfalls. In particular, SciML mixes the principles of the mechanistic models with data-driven elements, allowing for models which can learn from data but can be designed to ensure crucial properties such as conservation laws are kept. The burgeoning field of SciML has demonstrated that in many promising applications this can greatly improve the accuracy and reliability of ML predictions in the context of physical systems.

However, the growing space of SciML tooling has to this point been largely academic due to the early stage of the field. These tools assume deep familiarity with deep learning stacks like PyTorch, Jax, or Julia's Flux.jl, assume the author is comfortable designing neural architectures, is capable of debugging failures in the ML training processes themselves, and is comfortable with the scaling and deployment of the ML to distributed GPUs for the full training problems. While this problem can be largely handwaived in computer science circles by assuming that future university training programs will integrate machine learning knowledge into every discipline, it's unrealistic that every field integrates a complete ML engineering PhD core into every degree program. As such, there is a major barrier to the adoption of this tooling beyond the current academic research setting which requires both the simplification of the tooling and integration into workflows that engineers are already trained to understand.

In order to do this effectively, a systems modeling tool would need to integrate SciML directly into its core architecture. This means having all elements of the simulation system compatible with being a part of the training loops, which requires solving the compatibility with automatic differentiation that was previously alluded to as a major hurdle to retrofit into existing code bases. Additionally, most machine learning problems at the scale of realistic models cannot assume that the average user will have the right hardware for performing the training, as these can require multiple GPUs coupled together. Therefore the full system will require having deep integration with cloud-based compute resources and asynchronous workflows where jobs can be started and results can be analyzed. These issues are directly noted as in direct opposition to the workflows of the traditional systems modeling tools designed around the desktop workflow.



Example of how SciML (PEDS+AL) methods can perform on much smaller training data (two orders of magnitude reduction), compared to using a purely NN based approach. From "Physics-enhanced deep surrogates for partial differential equations" Pestoure et al.

Finally, while the high-level goals of many of the SciML fields are largely aligned with this direction, many of the techniques are not designed to address the issues associated with industrial use. For example, while techniques like physics informed neural networks (PINN) and neural operators have the ability to merge physical information into predictive models trained with data, the exact mechanism by which these techniques are changing from the model's predictions is kept within the black box of the neural networks, where this opaqueness of the

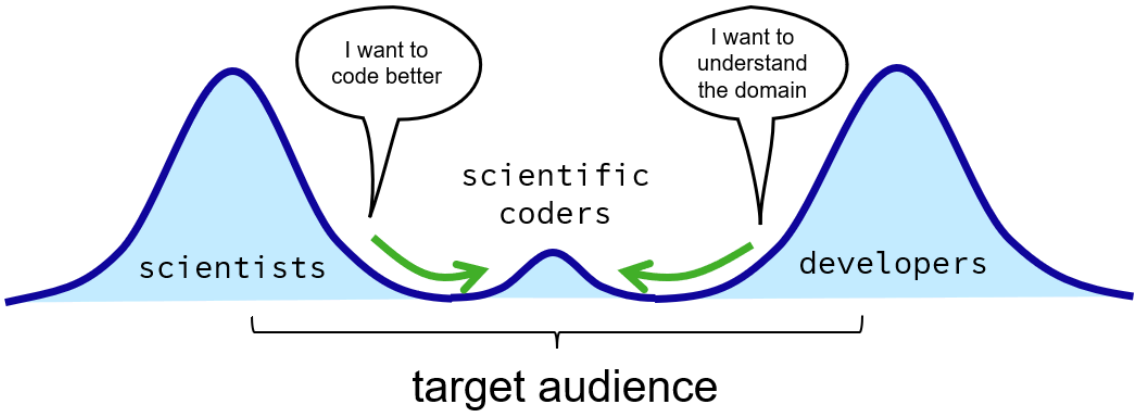
black box inhibits the engineer's ability to iteratively refine the model. Thus while these demonstrated some promise in their predictive power in the academic scenario that you have all of the data you will ever need and only need to use the model for one prediction, this does not translate to the real-world where model details are reused and refocused to new applications over time. In addition, it should be noted that there are major independent studies about the applicability of these methods which call into question their ability to accurately converge on non-toy examples, and the training of these models can be orders of magnitude slower than the traditional solving techniques.

Thus instead, the integration of SciML into system modeling tools requires a deep understanding of the domain in order to make use of the right types of methods, such as Universal Differential Equations [11], which can augment the modeling workflow, use less training data than purely data-driven methods [8], and move away from traditional neural architectures and training techniques with local minima behavior towards reservoir computing workflows which can guarantee reliable training and convergence to work the first time. With these kinds of changes and integration of hyperparameter tuning AutoML, the details of the ML processes can be masked from the user and directly integrated into workflows for engineers without an ML background.

3. Unlocking new hardware design experiences

3.1 Breaking the silos: Bringing engineers and developers onto a single source of truth

A fundamental issue with the previous era of engineering software was the split between engineers and developers. Engineers who focused on the physics, developing accurate models, and tuning controllers have developed a major preference for the GUI-based workflows of systems modeling tools. On the other hand, the software developers in charge of the deployment of code onto the hardware live in a different world, using low-level programming languages like C and Rust to hand-code the engineer-optimized solutions onto the device. While there are some tools which seek to overcome this hurdle by providing the ability to generate embedded code from the GUI, ultimately there is a divide in workflows. Modern software engineering relies on CI/CD automation, exploring diffs in textual formats, and using version controlled workflows with structured review processes. GUIs enable the engineers to design what to go on the hardware, but they get in the way of deploying that design in an agile but reproducible manner.



Matthijs Cox writes about the Two Cultures Problem [4]. Scientists would love to write better code whereas developers would love to have a better understanding of the domain. These communities are separated largely by tools which encourage working in silos, and lead to two separate cultures. Bringing these two cultures together would lead to tremendous benefits for the organization, and a crucial component in doing so is to have a common set of tools for everyone.

The future of engineering software must discard this dialectical divide by giving a single source of truth. The model files are simultaneously representable within graphical modeling environments and textual workflows. As such, the future modeling software must return all of software engineering tooling to the domain by naturally interfacing in the textual form without sacrificing the engineer's ability to visualize the same source in a graphical way. This allows for the hand-off between silos to be seamless, eliminating translation steps, and ensures that a single software artifact can describe the entire process from conception to value creation.

Substantial ROI for customers across personas

CXO Manufacturer	Engineer	Developer	CXO Asset Operator
• Breaking silos increases team productivity	• One model for design, analysis, controls, digital twin	• Dislike GUI-only tools used by engineers	• Digital Twins increase product value for operators
• Shorter product iteration cycles and faster time to market	• High performance modeling capabilities allow incorporating more physics	• Textual representation for software-like velocity: VS Code, Git, CI / CD	• New revenue streams after asset deployment through maintenance contracts

Software defined machines across the manufacturing process. By having a single system that connects design, deployment, and operation, allowing feedback and refinement, all aspects of a device's lifecycle can be transformed and remove the transformation friction between what were previously separate teams.

3.2 Rapid Deployments for Over-The-Air-Updates

As modern devices are becoming increasingly software integrated, one of the major features being touted is the ability to continually improve devices via over-the-air-updates. By simply shipping a car with a bit of extra computing power, your 2025 model vehicle can get the perception system of the 2027 vehicle simply by downloading the latest version of the autonomy software in a fully automated fashion. This gives improved value and longevity to the customer over traditional manufacturing which requires an entirely new device in order to receive any improvements. This has vastly changed the landscape of design, where instead of only changing a vehicle or airplane at well-defined longer scale time intervals, small improvements can be continually made, treating our manufactured devices closer to the fast iterating space we see in software services.

However, the same monsters can rear their head in this space as real-world devices need to balance safety, reliability, and other such metrics as any mistakes in this process is not just a bug but a potential hazard that can lead to accidents and death. Because of this, a car manufacturer needs to be very careful with every update they push out: faster updates give a better experience to the user, but a bad update to an autonomous driving system could be fatal.

In the future of engineering software, all of the evolving models of a team can live in a cloud environment, and thus any of the changing versions can be selected to construct and test deployments. Version control systems would allow the developers of models to maintain multiple versions and branches of their model and component libraries. They can for example keep a “no AI” version, “stable” version, and a “bleeding edge” version. The team running the deployment can then use the integrated dependency management in order to maintain a model of the full vehicle where they test in piecemeal the effect of change: updating the version of each component one at a time, bringing in the AI enhancements only as needed for the fidelity required in the final build. The deployment team could then in isolation test the complete build and A/B test the effectiveness of specific AI model enhancements and report back their findings to the modeling engineers responsible for the given component. This gives a fast acceptance and rejection mechanism for the validation of incorporating any AI elements, with the ability to fully log all of the choices for easier auditing.

3.3 Living Digital Twins: Scientific AI as an Evolving Model

Current engineering workflows are designed around an understanding of an engineer at the computer driving the model development. However, the SciML techniques can integrate with the growing streams of real-world sensor data in order to continually evolve the model

autonomously. Because of this, new engineering software designs could use the cloud based approach at its core, so that the life of the model continues beyond the point where the modeler closes their laptop. The ideal scenario can be imagined as a world where the model autocompletion is run autonomously in the background as data streams in, and when the engineer comes back from lunch, they open their laptop to find that their model of the car's suspension is likely missing a crucial friction term required for the performance loss and thermal output being higher than expected. The engineer can validate this SciML predicted model augmentation by gathering new data in the lab and report back to the design team about this previously unknown physical effect. In this way, the engineer is continuing to improve our digital understanding of the processes, but is not tied day-to-day in the minutiae of coding but is instead focused on the interpretation, validation, and communication that transforms models from math to value.

3.4 Individualization of Models and Predictive Maintenance

Once the modeling process is able to quickly and autonomously update from a baseline, we are no longer tied to simply modeling the idealized “average twin”. For example, while we can build a model of a jet engine, the processes can be set up to feed the SciML training specifically the data from a single engine, forcing the system to learn how flying over the Gobi desert has caused a specific device to diverge from the standard engine. This process would pinpoint the physical differences in the action of this specific component, allowing the engineers to identify failure modes for future design improvements, predict the degradation of performance to flag components to take out of the field for repair, and better ensure the safety of their devices.

3.5 Managing a Cadre of SciML-Enhanced Models

As machine learning becomes more integrated into engineering workflows, engineers will find themselves spending more time managing the understanding and cataloguing of the trained models. For example, when building a surrogate of a fluid dynamics model, every trained surrogate will have different properties. One surrogate may be trained in the range of pressures from 1 bar to 10 bar, while another is trained to be accurate in the range of 5 bar to 50 bar. Another set of surrogates might both be trained over the same space, but due to the effects of local optima, one surrogate might give better results on viscous fluids while the other might give better results on non-viscous fluids. Instead of running an expensive re-training process tools for every new scenario, the tooling can help catalogue the trained models and store reports of the validations in order to better help the modeler understand their AI archive and better pull the appropriate model out of the traceable and version controlled repository as needed, or understand when a new training may be necessary.

4. Dyad is built from the ground up for Software-Defined Machines

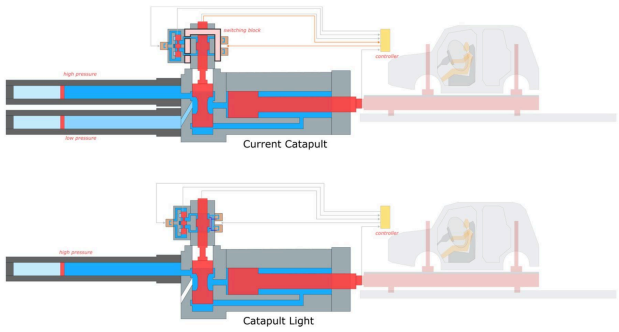
Dyad[1] is a new AI-enabled system modeling tool which is being designed to unlock the workflows of the future that are detailed and speculated above. By being a cloud-first, differentiable, and extendable system, Dyad is enabling a future where over-the-air updates of AI-integrated models are enhancing product performance.

1 Dyad is the new name for the product previously known as JuliaSim.

Dyad Underlying Technology Success Story

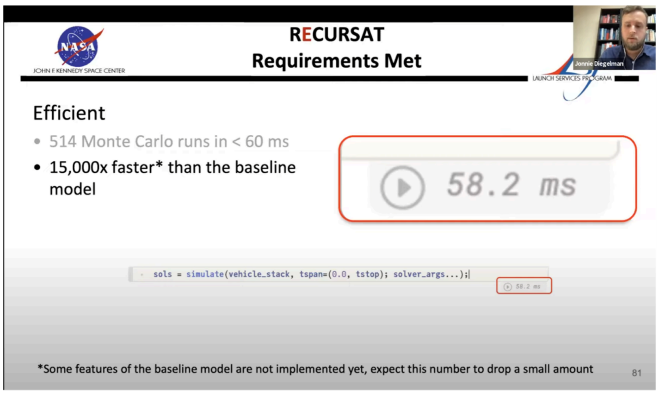
INSTRON

Leader in the materials testing industry and developer of the Hydroplus Catapult System



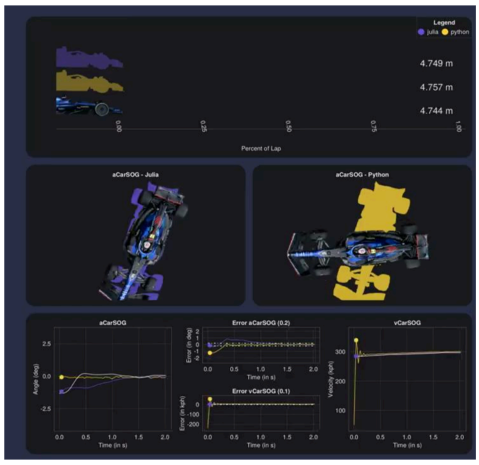
The Catapult Light redesign from Instron. **Dyad's underlying technology enabled a 500x speedup for this workflow** [19], cutting simulation time from months to hours. This made it possible to explore full system changes to find the best performing solution with a single mode of operation. The optimizations resulted in a simpler lower cost configuration, "Catapult Light", that provides performance and precision at a fraction of the cost by removing expensive low-pressure hydraulic arms and instead relying on improved control strategies.

NASA Launch Services, the launch analysis team for the NASA Kennedy Space Center



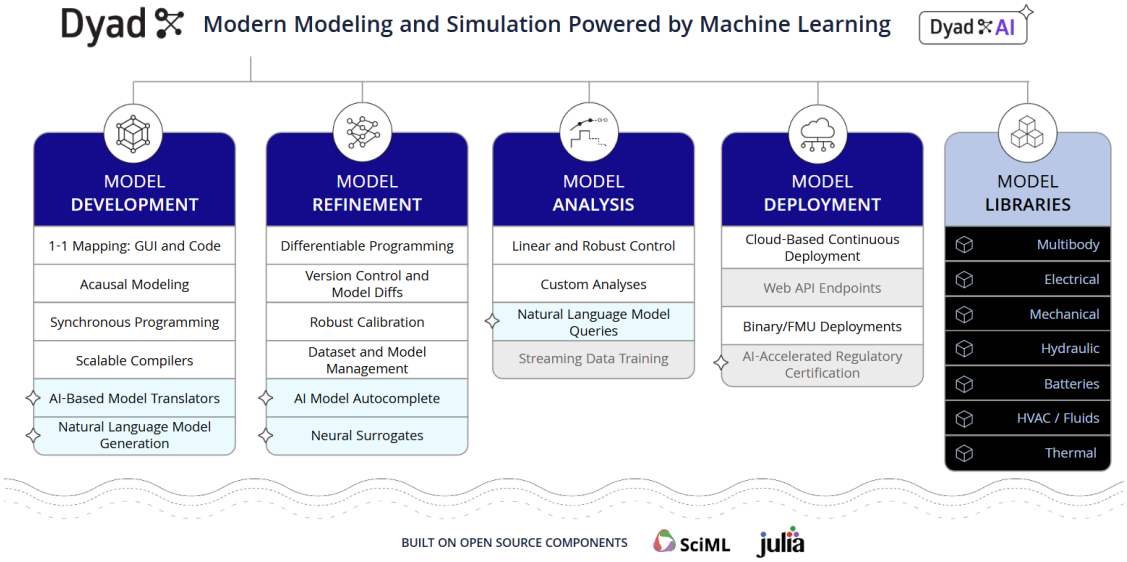
The public results of the RECURSAT project from NASA Launch services from engineer Jonnie Diegelman [14]. Showcased is the slide demonstrating the launch services simulation performance went from 15 minutes per run with Simulink to 58.2ms, or 15,000x faster. The previous Simulink-based workflow required that launch decisions were made at the end of the day to give engineers a full 7 hours to prepare analyses. After the change, analyses were performed in an interactive

Legendary F1 team known for innovation, speed, and championship-winning excellence.



Williams Racing employed Dyad to create a digital twin for a physical Speed over Ground (SoG) sensor [20]. The digital twin provides in-lap insights without the negative impact of extra weight and poor aerodynamics that come with running a race with the physical sensor. In the past, Williams Racing tackled this problem using classic machine learning techniques (Gaussian processes in PyTorch) depicted in Yellow. Dyad reimagined and improved the approach by implementing SciML techniques shown in Purple. The image of the car depicted the true orientation of the car, which demonstrates the Dyad model achieved approximately 50% less error in predictions over the original ML technique, evaluated 4x faster, and captured high-frequency features commonly found in vehicle control inputs.. Dyad deployed the model as an FMU for easy integration with

their real-time race analysis computer.



Dyad is built upon an open source foundation and makes fundamental strides in model development, refinement, analysis and deployment, which are all the key pillars of software defined machines (SDM)

4.1 Model Development

4.1.1 Dyad Modeling Language

The Dyad Lang is the modeling language at the heart of Dyad, providing a one-to-one textual representation of the model graphics and model analyses. This enables version control of the models themselves and code-based workflows. The textual representation of the model analyses means that every plot and result can be fully and exactly reproduced.

4.1.2 Seamless Workflows Between GUI and Code

Dyad's model development workflow is designed to achieve maximum composeability between models while achieving maximal scalability. Dyad Lang is designed to work with the Julia JIT compiler stack, and supports bi-directional editing of both the graphics and the code. This textual formulation integrates with modern software engineering workflows via compatibility with Git and continuous integration and deployment (CI/CD). These features have been used at companies like Instron [19] to design and maintain digital twins of sophisticated hydraulic crash equipment.

4.1.3 Acausal Without Sacrificing Control Systems

Dyad uses an acausal formulation pioneered by tools such as Modelica. Acausal modeling allows for the user to specify the high level physics of the system and allows an automated symbolic process to perform the derivation in order to arrive at the fundamental equations which determine the simulation. Previous work has demonstrated that such acausal modeling tools can be a major boost to productivity and model composability over causal system simulation tools such as Simulink, such as the improvements seen by Instron [19], and NASA launch services[14]. However, the causal modeling tools have typically been the dominant tool in areas such as control design as these systems are naturally causal and the acausal tools have lacked many of the control-systems features required in this domain, including auto-coding to embedded targets. The Dyad language includes specific extensions to the traditional acausal languages, such as analysis points and linearization capabilities, along with having a well-supported causal component language subset which seamlessly allows for the control-systems applications to integrate into the workflow, bridging a gap that traditionally divided the domain. Along with synchronous programming features and state machines, Dyad is able to bridge the gap between regulated flight controls and detailed physical models, and generate binaries for a number of embedded targets. Additionally, Dyad can leverage capabilities from the Julia language, such as novel state estimation methods to aid control system design, as done at Mitsubishi Electric Research Lab [6].

4.1.4 Synchronous Programming

Dyad has the ability to build and represent discrete-time components which can interact with continuous-time components. This allows for complex controllers and state machines executed on multiple different clocks, which are then interconnected with continuous-time plant models. The plant models used in model-based controllers can be automatically discretized to give fully discrete controllers which are compatible with the code generation and hardware deployment capabilities.

4.1.5 Scalable Compilers and Julia Solver Integration to Bridge Scales

Systems modeling tools have traditionally been focused on lower fidelity models due to the extensive issues involved with the compilation process. Equation-specific code can be manually written to be highly performant in specific domains, which is then the backbone of much of the 3D spatial simulation software such as those for computational fluid dynamics (CFD). The compilation issues have thus limited the types of models and components which can be expressed within systems such as Simulink or the Modelica tools. These models must be simplified, generally omit spatial details, and aim for low or intermediate fidelity.

However, Dyad is a new generation of compilers that solves the issues of scaling to large-scale systems. Its infrastructure is not tied to a single solver but instead is able to make use of the entire Julia SciML stack with hundreds of different techniques [12], where some are optimized for small 8 ODE systems and others are GPU-parallel distributed and optimized for millions of equations spread over a super computer. Dyad's compiler is able to target this large class of numerical infrastructure to re-specialize the approach based on different systems that are being modeled.

This is all possible because Dyad's infrastructure is structure-preserving, meaning that code from structures such as arrays or repeated loops can be kept. This removes the limitations of many previous compilers, like that of Dymola and other Modelica acausal compilers, which relied on passes like flattening and scalarization, which ultimately leads to the amount of code being generated growing linearly with the number of equations. With the structure-preserving tooling, Dyad is able to have constant code size for structured equations, which ultimately means that the compilation time is able to be better bounded on such models, bringing them within the domain of system modeling. The types of models which fit this domain are partial differential equation discretizations, such as CFD and finite element models, which means that with the new infrastructure of Dyad, this world can be brought together with the simpler system models. When combined with the specialized solvers, Dyad can present an alternative to the hand-tuned CFD codes and can generate simulators for these types of equations all within its composable modeling system.

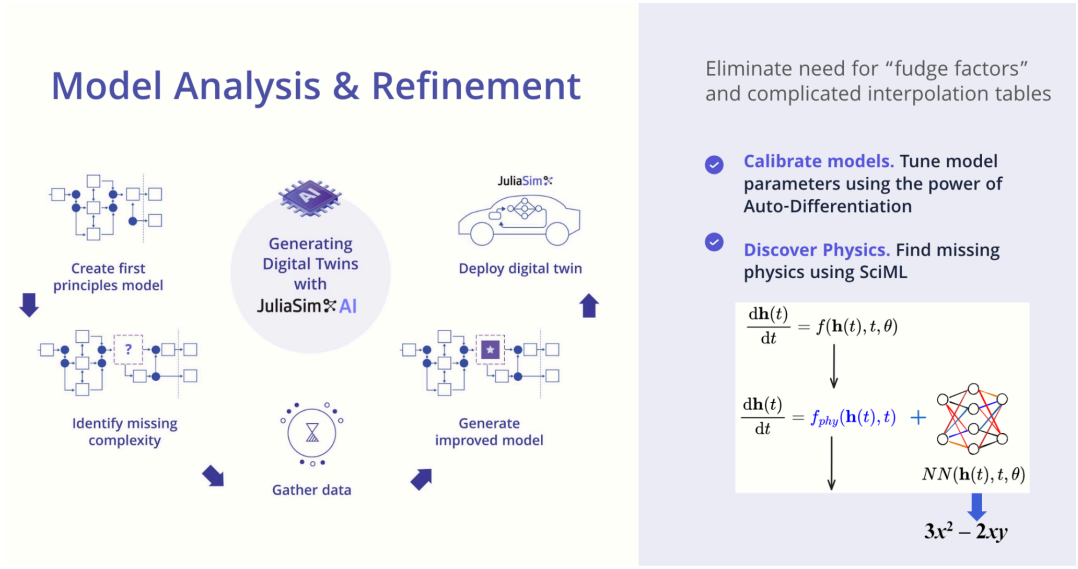
While in isolation at the start of the project we do not expect a pure Dyad CFD model to be competitive with a code like Ansys Fluent that is hand-optimized for exactly that model form, the ability to seamlessly integrate and compose models will mean that combinations, like a spatial battery model cooled by a chiller, with a CFD model of the resulting airflow, can be greatly improved over situations which attempt to co-simulate independent simulation codes. Over time, by focusing all modeling domains and equations through a single compiler stack, we also expect to achieve compounding benefits which could enable even a Dyad generated CFD model to compete with the hand-tuned versions because of the increased scale of utility provided by this design.

4.1.6 AI: Generative AI for Model Generation and Translation

The starting of modeling a new phenomenon can give a mental block as to where to start. Engineering would typically have to spend hours scouring the literature in order to find how others have approached the problem and start by recreating similar models before venturing into the unknown. By using generative AI mixed with AI-based translation of models from the

corpus of a multitude of modeling languages, Dyad can present natural starting points to the modeling process and overcome the activation energy of the starting writer's block.

4.2 Model Refinement



Schematic of SciML-based automated model refinement from data. This showcases how recent advances such as differentiable programming and universal differential equations can be synchronized with streaming data sources in order to produce models which learn previously unknown physics on the fly, iteratively improving themselves at the pace of computing. This transforms the modeling workflow from being

4.2.1 Differentiable Programming Integration for Fast Calibration and Design Optimization

The modeling and simulation tools of the Dyad stack are fully compatible with modern tooling for automatic differentiation (AD), enabling differentiable programming (dP) workflows. All forms of inverse problems, which includes the solving of problems like calibrating models to data and performing design optimizations, rely on a form of gradient-based optimization as the central calculation, and the calculation of the gradient is the core technical hurdle. Dyad's AD integration allows for the automatic construction of adjoints which leads to orders of magnitude faster and more accurate gradients and thus optimizations. This compounds as the size of the models increases as the cost adjoint approach is linear with respect to the model and parameter sizes while the naive non-adjoint solutions are multiplicative, meaning that models which were once outside the realm of a deep data integration are now computationally possible. One of the key aspects to note about this enabling technology is that it can be very difficult to retrofit into legacy programs, meaning that this advantage (which enables all of the other model refinement features) is a technical moat for the Dyad design against the legacy competitors.

4.2.2 Version Control and Model Diffing for Iterative Refinement

Because the model's form is saved directly in a textual representation, Dyad integrates directly with Git-based workflows. Dyad's project features are able to track the versioning of the models over time and show "diffs" which highlight what changed between versions. In addition, because there is a one-to-one mapping between GUI representations and the Dyad code, these diffs are representable in the GUI with coloring and other tricks able to highlight what has changed between model versions. These features are especially crucial in order to enable the AI-based model autocompletion in order to help modelers understand and track the exact changes proposed by the learning system.

4.2.3 Specialized Loss Function Generation for More Robust Training

Model refinement requires the solution of difficult nonlinear optimization problems. These are known to be prone to many phenomena such as local minima that make the process generally extremely difficult. There is an extensive literature on defining specific cost functions that can be used in order to avoid these issues, though they are not commonly used due to the complexity added to the process. Dyad includes a litany of these options in order to generate optimization problems, which mix with the differentiable programming functionality, in order to have a much higher chance of being well-defined and giving accurate results. This results in more robust model calibrations and other automatic refinements.

4.2.4 Dataset and Model Management for Logging Training Results

The integration of data into models and the resulting solution of inverse problems (neural network training, model calibration, etc.) can be a difficult computational task. Therefore it is necessary to track all of the experiments that were run, hyperparameters of the algorithms used (for example, results of different fitting choices, initial guesses, etc.) in order to get a full picture of the landscape. Dyad includes not only features for asynchronous execution of these long running jobs, i.e. the ability to spin up cloud runners to run these tasks in the backend, but also a complete logging system for investigating what types of analyses were previously performed and build comparisons between them. This lab tracker is essential to the AI functionality by allowing for comparative analyses for the AutoML features, making it so the automatic choices of neural network architectures can be easily tracked and understood by advanced users, while simplifying the process for beginning users who just want to pick out the best result out of a hat.

4.2.5 Streaming Data Sources and Event Triggered Model Training

Dyad is served through JuliaHub, which provides connectors to many popular datalakes and datastores. After connecting with the appropriate data source [5], Dyad models can be deployed as live apps which can be retrained either at regular time intervals, or when a new batch of data

enters the datastore, or externally via an API.

4.2.6 AI: Model Autocomplete via Scientific Machine Learning

As the model development progresses [13, 20, 21], engineers must make a myriad of modeling choices in order to balance the fidelity of the model against the requirements of simplification. However, when the simulations diverge from reality, it can be difficult to identify the aspects of the model which need to be re-evaluated. The Dyad AI tooling for model autocomplete can use the real-world data to suggest to the modeler the precise model changes required to fix the predictions towards reality, and thus transform a tedious guess-and-check process of model refinement into a structured task of confirming and validating the predicted model extensions.

4.2.7 AI: Neural Surrogates for Accelerated Model Analysis

Sometimes, a high fidelity source truth can be helpful to integrate into a system model. For example, precise simulators for fluid or structural properties can serve as a digital source of truth. The Dyad AI tooling for automated surrogate generation allows for slimming the twin of the component [1, 6] in a way that retains the required fidelity to integrate with the rest of the system without introducing heavy computational burden. This reduced computational burden has been leveraged to design control systems in both commercial [10] and automotive HVAC systems, and even for solving large scale inverse problems when evaluating the erosion of public infrastructure such as bridges.

4.3 Model Analysis

4.3.1 Built-In Control-Systems Synthesis and Analyses

Dyad includes a complete control-systems analysis suite which covers a wide range of traditional uses. This includes many features such as linear analyses (linearization, PID autotuning, etc.), robust control (H_∞ control), model-predictive control (MPC), and more. This functionality bridges the gap between what was traditionally covered by several different product offerings. Generally, the acausal modeling tools (such as Modelica) are known to be the better systems for building complex plant models, while tools like Simulink have been known to have more complete control-systems capabilities. Dyad covers this gap by both being a scalable acausal system for efficient construction of complex models, while at the same time offering a complete ecosystem for control-systems.

4.3.2 A General System For Adding and Sharing Custom Analyses

One major differentiating factor of the Dyad system is its flexibility. This flexibility is not just within code and models but is also demonstrated in the analyses available in the GUI because Dyad includes a general system for hooking into the GUI for custom analyses. All of the internal analyses (controls, simulation, AI, etc.) are built to a documented API that exposes these model analyses to the user in the GUI. This API is designed to be open and accessible to users as well, meaning that the user base not only can share libraries of models but also libraries of analyses. This means that specialized control-systems analyses, optimal experimental design techniques, etc. can be implemented as GUI extensions by external parties and shipped as part of our model library system in order to allow further growth of the Dyad platform. But this also allows for company specific analyses, like the construction of specific plots or reports about models for regulatory deliverables, to be constructed and shared within a company and become a standard GUI button for a specific user group. This greatly grows the customizability of the GUI-based features and thus removes many of the limitations that are traditionally discussed about the legacy GUI-based system modeling tools.

4.3.3 AI: Large Language Models for Natural Language Model Analytics

Understanding what a simulation is saying is a language all to its own, where engineers must dig through mountains of documentation in order to force modeling systems to produce the visualizations which are informative. By integrating natural language queries into the modeling system [7], modelers can ask for visualizations in a format they already understand, cutting through the barriers to truly understand their digital twin.

4.4 Model Deployment

4.4.1 Cloud-Based Continuous Deployment

Dyad runs natively in the cloud, and thus all models that are developed are naturally available in the cloud file system. This means that all models can be hooked into cloud-based continuous deployment systems for generating hardware-specific binaries and run automated testing where the results can be accessible at a team-wide level. This means that teams do not need to wait on the modeler to go to their computer to generate say an FMU binary for downstream usage, other modelers who have library access can sign into Dyad and click a few buttons to generate the binaries they need, and write scripts to automatically trigger build systems using the live version of the model as needed.

4.4.2 Web API Endpoints for Model Interactions and Predictions

Dyad is served through JuliaHub, a cloud-based platform specialized for on-demand computation. Dyad models can be deployed as live “apps” which run in real-time and can be queried for model predictions via a REST API. This deployment strategy is leveraged to provide predictive and preventive maintenance services based on Dyad models. This is an active area of interest in commercial aerospace [1] and public utilities, where jet engines and pumps act as live assets whose failure or degradation leads to loss of services.

4.4.3 Binary Deployments Model in the Loop, Software in the Loop, and Hardware in the Loop (MiL/SiL/HiL)

Dyad’s compiler architectures are built on Julia and LLVM which is able to target binaries to embedded devices via the new juliac features [2]. This means that Dyad’s models and controllers can be used to generate embedded code for a number of microprocessors and microcontrollers (via a real-time environment). This allows Dyad to be central to embedded control-systems applications, along with supporting workflows for MiL/SiL/HiL testing.

Additionally, this same workflow would enable algorithms written in Dyad to be compiled into a library which can then be deployed and linked against firmware running on commercial-grade equipment, such as at ASML [15].

4.4.4 Compliance with Regulatory Practices and Certification

Dyad will also generate human-readable C-code with the mapped requirements, maintaining the digital thread from high-level requirements to C-code that is compiled onto the target, as specified by the DO-178C regulatory guidance for the commercial aerospace industry.

4.4.5 AI: Retrieval Augmented Generation for Accelerated Regulatory Certification

When considering the use of the model in regulated contexts, such as DO178C qualified workflows for commercial aerospace, the engineer needs to ensure that their model conforms to the required processes and correctly integrates the natural language requirements. Dyad AI allows for highlighting potential areas where the model diverges from the assumed requirements description, helping to accelerate the process by which the model is refined to match the validation requirements.

Together, these workflows enhance the modeler to empower them to be more productive while not compromising or sacrificing their expertise at ensuring the end system is robust for real-world usage.

4.5 Model Libraries

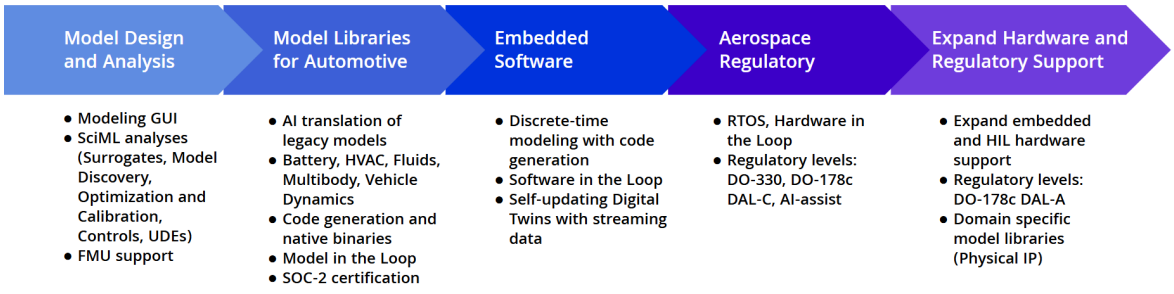
4.5.1 Extensive Pre-Built Component Libraries

Dyad comes with extensive pre-built component libraries which allows users to easily start constructing digital twins. This standard library includes many domains such as electrical, mechanical and hydraulic components, and includes discrete-time components to serve as central elements in controllers. Dyad also includes detailed libraries for specific subdomains on a case-by-case basis, including lithium ion batteries, aerial vehicles, and multiphase flow models of HVAC systems, to further accelerate the process of generating digital twins in high demand areas. Dyad also includes multibody libraries for the construction of rigid and flexible bodies, along with a visualization system, for modeling devices like industrial robots and vehicles.

4.5.2 Community Model for Library Sharing and Licensing

Unlike the traditional systems modeling systems whose business model is built from the top-down focusing on model libraries as the key source of revenue, and thus building a preference for vendor-built libraries into the system, Dyad is focused on being a platform by which all developers build and share models. Therefore, central to the design of Dyad is that every model a modeler builds is in a model library. Dyad includes a package management system for sharing and depending on other model libraries. The cloud-based file system then allows for teams to share models within a company, and for users to declare models as open to be shared by all users of Dyad. This makes Dyad a marketplace for models, with plans to allow for proprietary models to be released and marketed through the system for external model library vendors to supply other Dyad users directly through the package management system.

5. Roadmap



Dyad software roadmap. Shown is a simplified trajectory for Dyad's development towards achieving software-defined machines. It includes both the aspects required to achieve embedded software capabilities while also achieving the aspects required for digital twins.

6. Conclusion

The integration of AI into digital twins for industrial applications needs to be done carefully in order to ensure the safety and reliability of traditional engineering is brought into the new age. Dyad is specifically designed to incorporate AI and SciML into the right areas that allow for accelerating the process towards more accurate digital models of the physical world, while allowing for engineer-in-the-loop workflows. We envision that as the adoption of AI progresses, this will dramatically transform the modeling workflows, and have described a world where models are autonomously improving themselves in the cloud via streaming data and engineering teams work to validate the changing model and rapidly deploy the advances for over-the-air-updates and predictive maintenance. Every aspect of the engineers productivity can soar an order of magnitude above where it is today when all of these elements come together, leading to more efficient and performant devices, all without sacrificing an ounce of safety.

7. References

[1] Anas Abdelrehim, Dhairya Gandhi, Sharan Yalburgi, Ashutosh Bharambe, Ranjan Anantharaman, and Chris Rackauckas. 2025. Active learning enhanced surrogate modeling of jet engines in JuliaSim. *arXiv [cs.CE]*. https://doi.org/10.2514/6.2025-2323

[2] Fredrik Bagge Carlson, Cody Tapscott, Gabriel Baraldi, and Chris Rackauckas. 2025. C codegen considered unnecessary: go directly to binary, do not pass C. Compilation of Julia code for deployment in model-based engineering. *arXiv [eess.SY]*. Retrieved from https://scholar.google.com/citations?view_op=view_citation&hl=en&citation_for_view=1kyW6dwAAAAJ:4fGpz3EwCPoC

[3] Jose Castillo. 2024. Rivian Suspension software update. *RivianTrackr - All things Rivian, EVs, charging, and roadtrips*. Retrieved April 10, 2025 from https://riviantrackr.com/2024-19/

[4] Matthijs Cox. 2023. How to solve the two language problem? *The Scientific Coder*. Retrieved April 10, 2025 from https://scientificcoder.com/how-to-solve-the-two-language-problem

[5] Deep Datta. 2024. Quick and Easy Data Migration with JuliaHub's New Tool. Retrieved March 12, 2025 from https://juliahub.com/blog/easy-migration-with-juliahub-import-tool

[6] Mitsubishi Electric. 2024. Improved HVAC Diagnostics. Retrieved March 12, 2025 from https://juliahub.com/industries/case-studies/improved-hvac-diagnostics-juliasim-case-study

[7] Panagiotis Georgakopoulos. 2023. Ask AI with ChatGPT on JuliaHub. Retrieved March 11, 2025 from https://juliahub.com/blog/ask-ai-chat-gpt-juliahub

[8] Raphaël Pestourie, Youssef Mroueh, Chris Rackauckas, Payel Das, and Steven G. Johnson.

2021. Physics-enhanced deep surrogates for partial differential equations. *arXiv [cs.LG]*. Retrieved from https://www.nature.com/articles/s42256-023-00761-y

[9] Adrian Pop, Per Östlund, Francesco Casella, Martin Sjölund, and Rüdiger Franke. 2019. A New OpenModelica Compiler High Performance Frontend. In *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*, February 01, 2019. Linköing University Electronic Press. https://doi.org/10.3384/ecp19157689

[10] Chris Rackauckas, Maja Gwozdz, Anand Jain, Yingbo Ma, Francesco Martinuzzi, Utkarsh Rajput, Elliot Saba, Viral B. Shah, Ranjan Anantharaman, Alan Edelman, Shashi Gowda, Avik Pal, and Chris Laughman. 2022. Composing Modeling And Simulation With Machine Learning In Julia. In *2022 Annual Modeling and Simulation Conference (ANNSIM)*, July 18, 2022. IEEE, 1–17. https://doi.org/10.23919/ANNSIM55834.2022.9859453

[11] Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, and Alan Edelman. 2020. Universal Differential Equations for Scientific Machine Learning. *arXiv [cs.LG]*. Retrieved from http://arxiv.org/abs/2001.04385

[12] Christopher Rackauckas and Q. Nie. 2017. DifferentialEquations.Jl – A performant and feature-rich ecosystem for solving differential equations in Julia. *J. Open Res. Softw.* 5, (May 2017), 15. https://doi.org/10.5334/JORS.151

[13] Christopher Rackauckas, Roshan Sharma, and B. Lie. 2021. Hybrid mechanistic + neural model of laboratory helicopter. *en. In* (March 2021). https://doi.org/10.3384/ECP20176264

[14] The Julia Programming Language. 2021. Modeling Spacecraft Separation Dynamics in Julia - Jonathan Diegelman. Retrieved December 5, 2023 from https://www.youtube.com/watch?v=tQpqsmwlfY0

[15] The Julia Programming Language. 2024. Keynote: Software at ASML: the Force behind making microchips | du Mee | JuliaCon 2024. Retrieved March 12, 2025 from https://www.youtube.com/watch?v=uYhQHMtHJrU

[16] K. Wang, C. Greiner, John Batteh, and Lixiang Li. 2017. Integration of complex Modelica-based physics models and discrete-time control systems: Approaches and observations of numerical performance. *Int Model Conf* (July 2017), 132:059. https://doi.org/10.3384/ECP17132527

[17] 2023. What is digital-twin technology? Retrieved April 10, 2025 from https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-digital-twin-technology

[18] Simulation Software Market Size, Share & Trends Analysis Report By Component (Software, Service), By Deployment, By Application, By End-use (Conventional Automotive, Healthcare, Aerospace & Defense), By Region, And Segment Forecasts, 2024 - 2030. Retrieved April 10,

2025 from https://www.grandviewresearch.com/industry-analysis/simulation-software-market

[19] Auto Crash Simulation - JuliaHub. Retrieved December 6, 2023 from https://juliahub.com/case-studies/auto-crash-simulation/

[20] Williams Racing Unlocks SciML using JuliaSim. Retrieved December 6, 2023 from https://juliahub.com/case-studies/williams-racing-unlocks/

[21] Scientific Machine Learning for Thermo-Fluid System Design in Transport Refrigeration Systems. *(in progress)*.